

VBA 开发手册

作者：郑立楷

目录

第一章 VBA 入门

了解嵌入和全局 VBA 工程

用 VBA 管理器组织工程

处理宏

用 VBA IDE 编辑工程

更多的信息

回顾 AutoCAD VBA 工程术语

回顾 AutoCAD VBA 命令

第二章 理解 ActiveX 自动操作基础..

理解 AutoCAD 对象模型

访问对象层次

通过集合对象操作

理解属性和方法

理解父对象

定位类型库

在数据库中返回第一个图元

在方法和属性中使用变体

使用其它程序语言

第三章 控制 AutoCAD 环境

打开、保存和关闭图形

设定 AutoCAD 参数

控制应用程序窗口

控制图形窗口

重置活动对象

设定和返回系统变量

精确制图

提示用户输入

访问 AutoCAD 命令行

工作于无打开文档状态

输入其它文件格式

输出到其它文件格式

第四章 创建和编辑 AutoCAD 图元

创建对象

确定容器对象

创建直线

创建曲线对象

创建点对象

创建实体填充区域

创建面域

创建阴影

创建实体填充区域

创建面域

创建阴影

编辑对象

工作于命名的对象

选择对象

复制对象

移动对象

删除对象

比例缩放对象

转换对象

延伸和修剪对象

分解对象

编辑多段线

编辑样条曲线

编辑阴影

使用图层、颜色和线型

使用图层

使用颜色

使用线型

分配图层、颜色和线型给对象

添加文本到图形中

处理文字样式

使用单行文字

使用多行文字

使用 Unicode 字符、控制代码和特殊字符

替换字体

拼写检查

第五章 标注与公差

标注的概念

创建标注

编辑标注

利用标注样式

在模型空间和图纸空间中标注

创建引线及注解

创建形位公差

第六章 定义菜单和工具栏

理解 MenuBar 和 MenuGroups 集合

加载菜单组

改变菜单条

创建和编辑下拉菜单和快捷菜单

- 建立并编辑工具栏
- 建立宏
- 对菜单项和工具栏项增加状态栏帮...
- 在右键菜单中增加条目

第七章 使用事件

- 了解 AutoCAD 中的事件
- 编写事件处理器的方法
- 处理应用程序级事件
- 处理文档级事件
- 处理对象级事件

第八章 在三维空间下工作

- 指定三维坐标
- 定义用户坐标系统
- 坐标转换
- 建立三维对象
- 在三维中编辑
- 编辑三维实体

第九章 定义布局及打印

- 了解模型空间和图纸空间
- 了解布局
 - 了解布局与块的关系
 - 了解打印配置
 - 决定布局设置
- 了解视口
 - 切换至图纸空间布局
 - 切换至模型空间布局
 - 建立图纸空间视口
 - 改变视口视图及内容
 - 在图纸空间中缩放线型样式
 - 在被打印视口中的消隐线
- 打印图纸
 - 执行基本打印
 - 在模型空间中打印
 - 从图纸空间打印

第十章-高级绘图与组织技术

- 使用光栅图像
 - 附着和缩放光栅图像
 - 管理光栅图像
 - 修改图像和图像边界
 - 剪裁图像
- 使用块和属性
 - 使用块
 - 使用属性

第一章 VBA 入门

本章将为你介绍 AutoCAD

VBA 工程及 VBA 交互开发环境(VBA IDE)。尽管大部分 VBA 环境在行为上都是相似的，但 AutoCAD

VBA IDE 还是有些独有的特性。在 AutoCAD 中还有些相关的命令可以用于装载工程、运行工程，或打开 VBA

IDE 环境。本章将概要介绍 VBA 工程、VBA 命令和 VBA IDE 的使用。

了解嵌入和全局 VBA 工程

AutoCAD VBA 工程是代码模块、类模块和窗体的集合，它们组合起来以执行给予的功能。工程可保存在 AutoCAD 图形中，或作为独立的文件保存。

嵌入工程是保存在 AutoCAD 图形中。当包含有这些工程的图形中 AutoCAD 打开时，他们可以自动地装载，这种方法可以很方便地分发工程给用户。嵌入工程也有它的极限，它不能打开或关闭 AutoCAD 图形，那是由于他们的函数只存在于工程所在的文档中。使用嵌入工程不需要在运行程序之前查找并装载工程文件。举个含有嵌入工程的图形的例子，当图形打开时，一个时间日志被触发。通过这样一个宏的应用可以登记并记录用户在该图形上所花费的时间。这时用户不必去记住在打开图形之前装载工程，这就是自动操作的一个很好的例子。

全局工程保存在独立的文件中，它更加通用，因为他们能在 AutoCAD 图形中运行，也能打开、关闭 AutoCAD 图形，但它在图形打开时不能自动装载。用户必须知道他们所需要的宏包含在哪个工程文件中。然而，全局工程非常容易与其它使用者共享，它可以将通用的宏做为很好的库而存在。举个例子，你保存在一个工程文件中的宏是有关多个图形的材料清单。这个宏可以在工作周期的末期由管理员运行，这样就可以收集到所有图形的信息。

在特定的时间，用户可以在同个 AutoCAD 进程中同时装载嵌入工程和全局工程。

AutoCAD VBA 工程与 Visual Basic 工程在二进制结构上是不兼容的。然而，其中的窗体、模块和类可以通过在 VBA

IDE 环境中使用输入和输出 VBA 命令来在工程之间进行转换。关于 VBA IDE 的更多信息，可以查看“用 VB

A

IDE 编辑工程”

用 **VBA** 管理器组织工程

你可以使用 **VBA** 管理器查看装载在当前 **AutoCAD** 进程的所有 **VBA** 工程。**VBA** 管理器一个 **AutoCAD** 工具，它允许你装载、卸载、保存、创建、嵌入和分离 **VBA** 工程。

打开 **VBA** 管理器的步骤：

1. 在工具菜单中选择宏-**VBA** 管理器。
2. 或者，在 **AutoCAD** 中调用 **VBAMAN** 命令。

本节的内容：

装载现存的工程

卸载工程

嵌入工程到图形中

从图形中分离工程

创建新的工程

保存嵌入工程

装载现存的工程

当你装载工程到 **AutoCAD** 中，所有的公用的子程序(也称为宏)都可以使用。

嵌入于图形中的工程在图形打开时就被装载。保存在 **DVB** 文件中的工程必须单独装载。

装载现存的 **VBA** 工程文件

1. 在 **VBA** 管理器中，使用装载项可弹出打开 **VBA** 工程对话框。

2. 在打开 **VBA** 工程对话框中，选择打开的工程文件。该 **VBA** 工程对话框允许你打开有效的 **DVB** 文件。如果你尝试打开不同类型的文件，你将会得到出错信息。
3. 选择打开。

你也可以使用 **VBALOAD** 命令直接出现打开 **VBA** 工程对话框以装载工程文件。

另外，当工程装载后，该工程所引用的其它工程将会自动地装载。

还有，**AutoCAD** 在启动时将自动装载名称为 **acad.dvb** 的工程文件。

相关主题：病毒警告

每次当你装载工程时，你会看到启用或禁用工程中的代码以阻止宏病毒的警告框。如果你选用启用代码，如果工程中含有宏病毒时病毒可能会发作。如果你禁用代码，工程同样会装载，但工程中的所有代码将不能运行。

想查看更多关于病毒保护的信息，请参考“设置工程选项”。

卸载工程

卸载工程以释放内存并保持装载的工程列表的长度以方便管理。

你不能卸载嵌入工程或由其它已装载工程所引用的工程。

卸载 **VBA** 工程的操作

1. 在 **VBA** 管理器中，选择要卸载的工程。
2. 选择卸载。

或者，使用 **VBAUNLOAD** 命令，它将提示你所要卸载的工程。

嵌入工程到图形中

当你嵌入一个工程时，你是将工程的一个副本置于图形数据库中。无论何时，当所包含工程的图形打开或关闭时，工程会同时被装载和被卸载。

一个图形只有在同一时间包含一个嵌入工程。如果图形已经包含有一个嵌入工程，你必须在同样地入其它工程之前将该嵌入工程分离出。

嵌入工程到 AutoCAD 图形中的操作

1. 打开 VBA 管理器并选择你所要嵌入的工程。
2. 选择嵌入。

从图形中分离工程

当你分离工程时，其实是将工程从图形数据库中删除，同时会提示你将工程保存为外部的工程文件。如果你没有将其保存为外部工程文件，该工程的数据将会删除。

从 AutoCAD 图形中分离工程的操作

1. 打开 VBA 管理器并选择要从图形中分离的工程。
2. 选择分离。
3. 如果你想保存工程信息为外部工程文件，可以在“你是否想在删除 VBA 工程前将其输出？”的提示下选择“是”，此时会显示另存为对话框，允许你保存文件。

如果你不想保存该工程信息到外部文件，可以在“你是否想在删除 VBA 工程前将其输出？”的提示下选择“否”，此时工程信息将在没有保存的情况下从图形中删除。

创建新的工程

新的工程将作为未保存的全局工程被创建。当工程创建时，你可以将工程嵌入图形中，或将其保存为工程文件。

创建新的 VBA 工程的操作

1. 打开 VBA 管理器。
2. 选择新建。

新创建的工程将使用默认的工程名称 **ACADproject**。你必须到 VBA IDE 中才可以更改工程名称。查看更多关于命名工程的信息，请参考“命名工程”。

保存嵌入工程

嵌入工程是在图形保存时同时保存的。全局工程必须使用 VBA 管理器或 VBA IDE 进行保存。

使用 VBA 管理器保存工程的操作

1. 打开 VBA 管理器并选择要保存的工程。
2. 选择另存为，此时另存为对话框打开。
3. 选择用于保存工程的文件。
4. 选择保存

处理宏

与设置 VBA 工程选项一样，宏对话框允许你运行、编辑、删除和创建宏。

宏是公用(可执行)的子程序。每一工程通常至少一个宏。

打开宏对话框的操作

1. 从工具菜单中选择宏-宏。
2. 或在 AutoCAD 中调用 VBARUN 命令。

在对话框中显示的是所有在有效范围内的宏名称。你可通过下拉列表来改变有效范围。该列表指定所要显示的宏所在的工程或图形。你可在以下列表内容选择所要显示宏范围

所有图形和工程

所有图形

所有工程

在当前所打开的单独图形

在当前所装载的单独工程

通过限制有效范围你可控制显示在列表中的宏名称的数量。它可帮助你注意到当前有多少宏装载到图形中或指定工程中有多少宏。

本节内容：

运行宏

编辑宏

逐语句运行宏

创建新的宏

删除宏

设置工程选项

运行宏

运行宏就是在当前 **AutoCAD** 进程中执行宏代码。当前活动图形指的是当宏执行开始时处于打开并处理激活状态。所有在全局工程的宏中所涉及的 **ThisDrawing** 对象将指向当前活动图形。在嵌入工程中，**ThisDrawing** 对象通常指向嵌入该宏的图形。

运行宏的操作

1. 打开宏对话框并选择要运行的宏。
2. 选择运行。

编辑宏

编辑宏将打开 **VBA IDE** 并打开所选定宏的代码窗口。详细的介绍请参考“用 **VBA IDE** 编辑工程”。

编辑宏的操作

1. 打开宏对话框并选择要编辑的宏。
2. 选择编辑。

逐语句运行宏

逐语句运行宏指的是开始运行宏并在第一行代码时暂停。**VBA IDE** 也随之打开所选定的宏的代码窗口并高亮显示所执行宏所在的行。

逐语句运行宏的操作

1. 在宏对话框中，选择要逐语句运行的宏。
2. 选择逐语句。

创建新的宏

你可以创建一个空的新宏。

创建新宏的操作

1. 打开宏对话框并输入新宏的名称。
2. 在宏位置下拉列表中选择所创建宏所要存在的位置。
3. 选择创建。

如果所指定宏的名称已经存在，系统将提示你是否替换现在的宏。

如果你在提示下选择“是”，则现在的宏代码将被删除，而一个新的而没有内容的宏也会以指定名称而创建。

如果你在提示下选择“否”，这时会返回宏对话框以输入另外的宏名称。

如果你在提示下选择“取消”，宏对话框将消失，宏也没有创建。

删除宏

你可以从工程中删除宏

删除宏的操作

1. 打开宏对话框并选择要删除的宏
2. 选择删除。系统会提示你是否确认删除。
3. 在提示下，选择“是”而删除宏，或者选择“否”以取消删除的操作。

设置工程选项

在 AutoCAD VBA 工程中可以设置三个选项：

启用自动嵌入

出错时允许中断

启用宏病毒保护

设置 AutoCAD VBA 工程选项的操作

1. 从工具菜单中选择宏-宏以打开 **VBA** 宏对话框。
2. 从 **VBA** 宏对话框中，选择选项以打开选项对话框。
3. 从选项对话框中，选择你所要设置的选项。
4. 选择确定。

本节内容：

启用自动嵌入

出错时允许中断

启用宏病毒保护

启用自动嵌入

自动嵌入特性将在打开图形时自动为所有图形创建嵌入的 **VBA** 工程。

出错时允许中断

在遇到错误时允许 **VBA** 进入中断模式。中断模式是在交互开发环境中暂时挂起程序的执行。在中断模式中，你可以检查、调试、复位、跳过或继续程序的执行。

当该选项启用时，在执行 **VBA** 宏的过程中如发现的未处理的错误，将会挂起执行的宏并在 **VBA IDE** 中显示宏的出错点。

当该选项禁用时，当执行 **VBA** 宏的过程中如发现出错时，将出现错误警告提示框，然后结束宏的执行。

启用宏病毒保护

病毒防护机制将在你打开可能含有宏病毒的图形时显示内建的警告信息。

用 VBA IDE 编辑工程

当工程装载到 AutoCAD 后，你可以使用 VBA 交互开发环境编辑其代码、窗体及进行引用。你也可以在 VB

A

IDE 中调试和运行工程。

本节内容：

打开 VBA IDE

查看工程信息

定义工程的部件

输入现存的部件

编辑部件

运行宏

命名工程

保存工程

引用其它 VBA 工程

设置 VBA IDE 选项

打开 VBA

IDE

当打开 VBA IDE 时，它就可以访问所有已装载的工程。

按需要打开 VBA IDE 的操作：

你可以从命令行或从菜单栏中打开 VBA IDE 从命令行，可输入 VBAIDE。或从工具菜单，使用宏-Visual Basic 编辑器。

在 AutoCAD 启动时自动打开 VBA IDE 的操作：

如果你想在每次开始 AutoCAD 时自动打开 VBA IDE，你必须在 acad.rx 文件中包含以下行：

```
acadvba.arx
```

[查看工程信息](#)

VBA IDE 包含名称为工程窗口的窗口，其中显示了所有已装载 VBA 工程的列表。它也显示包含在工程中的代码、类和窗体模块，还关联到该工程的文档，在该工程所引用的 VBA 工程，以及工程的物理位置(也称路径)。

工程窗口有它自己的工具栏，使用其工具栏可打开不同的工程部件进行编辑。使用查看代码按钮可打开选定模块的代码。使用查看对象按钮可显示如窗体之类的选定对象。

VBA IDE 工程窗口

工程窗口默认为可见。如果不可见，可从查看菜单中选择工程窗口，或按 CTRL+R。

[定义工程的部件](#)

每个工程可包含多个不同的部件。在工程中包含的不同部件可以是对象、窗体、标准模块、类模块和引用。

对象

对象部件声称了 VBA 代码可访问的对象或文档的类型。在 AutoCAD VBA 工程，该对象声称为当前 AutoCAD 图形。

窗体

窗体部件包含了当前你在工程中为了使用而建立的自定义对话框。

标准模块

该代码部件包含了普通程序和函数。标准模块也归类为代码模块或简单的模块。

类模块

类模块部件包含你所有定义为类的自己的对象。

引用

引用部件包含所有引用其它的工程和库。

增加新部件

增加新的部件是在工程中创建一空白的部件。你可在工程中增加新的模块、窗体和类模块。你有责任更新所有部件的属性(如部件的名称)并在其中写入适当的代码。当命名一个新的部件，请记住可能其它开发都会在以后的应用程序中用到你的部件。在你的开发项目中应该跟随适当的名称约定。

增加新的部件到工程的操作

1. 在 **VBA IDE** 的工程窗口中，选择你要增加部件的工程。
2. 从插入菜单中，选择用户窗体、模块或类模块以增加新的部件到你的工程中。

新的模块将被增加到你的工程中并出现在工程窗口中。

输入现存的部件

输入功能允许你增加现存的部件到工程中。你可以输入窗体、模块或类模块。输入的窗体文件为 **FRM** 文件，输入的模块文件为 **BAS** 文件，输入的类模块文件为 **CLS** 文件。

当你输入一个部件文件，文件的一个副本将输入并增加到工程中。而原来的文件也会保持不变。在输入部件中所做的
不会影响到原来的部件文件。

如果你用现存的同样的名称输入部件，该文件将添加到工程中相应的部件上。

从工程中输入现存的部件的操作

1. 在 **VBA IDE** 的工程窗口中，选择你要增加部件的工程。
2. 从文件菜单中，选择输入文件以打开输入文件对话框。
3. 从输入文件对话框中，选择所要输入的文件，然后按打开。

该输入部件将增加到你的工程中并在工程窗口中出现。编辑部件的属性，可选择工程窗口的该部件。所选部件的属性将在属性窗口中列出并可编辑。

编辑部件

在工程中编辑工程的操作

1. 在 **VBA IDE** 的工程窗口中，选择你要编辑的部件。
2. 选择工程窗口中的查看代码按钮以打开代码窗口。
3. 选择工程窗口中的查看对象按钮以打开用户窗体窗口和关联的工具箱。

你可同时打开你所有模块中的代码窗口，所以你可很容易地 不同窗体或模块中的代码，并在它们之间进行复制和粘贴。

访问关联在窗体中的代码的操作

在窗体窗口中双击任何控件。关联在该控件中的代码将会在代码窗口中打开。

使用代码窗口

代码窗口包含两个下拉列表、一个分隔条、一个边界标识条栏和全模块视图和过程模块视图的图标。

在代码窗口的顶部有两个下拉列表，它们显示当前对象和过程。你可通过更改在此下拉列表中的对象或过程来在工程中移动。在代码窗口的右侧有一个分隔条，它允许你在垂直方向分隔窗口。单独拖动该分隔条来创建另外的窗口窗格。该特性允许你同时查看相同模块中的两部分代码。关闭窗口格时，可将分隔条拖回其原先的位置。

边界标识条栏是在代码窗口的左侧。它用于显示在代码编辑和调试期间的边界标识条。

全模块视图和过程模块视图图标位于代码窗口的左下角，它可在单过程模块视图和全模块视图之间切换。

使用用户窗体窗口

用户窗体窗口允许你在工程中创建自定义对话框。

增加控件可简单地从工具箱中拖动想要控件并摆放在窗体中。你可从选项对话框中的常规选项卡中设置控件的对齐栅格。你可查看窗体栅格并可从选项对话框的常规选项卡中设定栅格线的大小。(参考“设置 VBA IDE 选项”以得到更多的信息。)

你设计的每一窗体都有最大化、最小化和关闭按钮。这些按钮已经为你准备好。

将代码增加到控件中，可简单地双击摆放在窗体中的控件。此时将打开相应控件的代码窗口。

运行宏

和在宏对话框中运行宏一样，你也可以在 VBA IDE 中运行宏。

从 VBA IDE 中运行宏的操作

从菜单中，使用运行宏菜单选项。

如果当前没有宏或窗体，将显示一对话框允许你选择所要运行的宏。

如果当前已经提供了宏(即光标已经在一过程中)，此时将直接执行宏。

命名工程

工程名称和保存工程的.dvb 文件名称是两个不同的值。当你保存工程时你所设定的为保存工程的.dvb 文件名称。而工程名称是在 VBA IDE 的属性窗口中设置的。

如果你没有设定工程名称和文件名称，AutoCAD 将自动地以以下默认的名称进行分配：

工程名称：ACADProject

文件名称：Project.dvb

更改工程名称的操作

1. 在 VBA IDE 的工程窗口中，选择要更改的工程。
2. 在属性窗口中，编辑工程的名称属性。

更改工程的文件名称操作

1. 在 VBA IDE 中，从文件菜单中选择保存项。
2. 在另存为对话框中，输入工程文件新的名称和路径。

保存工程

在 AutoCAD 的 VBA 工程中，没有直接的保存命令。取而代之，保存命令被置于 VBA

IDE 的文件菜单中和 VBA 管理器中。当发生以下事件时，VBA 工程中所进行的更改将访问一标准的保存 VBA 工程对话框：

你在 **VBA IDE** 中拾取了保存命令

你在 **VBA** 管理器中选择了另存为项

在 **VBA** 工程没有保存的情况下结束或退出 **AutoCAD** 进程

注意：当你保存工程时，它将分配一默认的文件名称 **project.dvb**。尤其重要的是在你保存工程时你应该为工程文件分配一新的名称。如果你使用默认的文件名称 **project.dvb** 保存工程，你将不能再创建新的空工程了。每一次你创建新的工程，你将得到的是装入已经存为 **project.dvb** 文件名称的工程。

引用其它 **VBA** 工程

从其它工程中引用 **VBA** 工程可开发者容易分享代码。开发者可使用宏来创建公用的库，然后在需要时引用库中的内容。这样可将共享代码公开并让众多的开发利用该代码。

引用其它 **VBA** 工程的操作

1. 在 **VBA IDE** 的工程窗口中，选择你要增加引用的工程。
2. 从工具菜单中，选择引用项以打开引用对话框。
3. 从引用对话框中，按浏览按钮打开添加引用对话框。
4. 从添加引用对话框中，选择你要引用的工程文件，然后按打开按钮。
5. 从添加引用对话框中，选择确定按钮完成引用的添加。

当另外的工程被成功以引用，你将会注意到在 **VBA IDE** 的工程窗口中有一新的文件夹。该新的文件夹是引用的标题，它包含着引用工程的名称。

当你引用了一个工程，你可使用工程中公用的代码或部件。

当引用了其它工程的工程被装载到 **AutoCAD** 中，被引用的工程也自动地装载到 **AutoCAD** 中。该被引用工程在引用它的工程没有关闭之前是不能关闭的。

你不能进行循环引用。也就是说，你不能引用包含第一个工程的工程。如果你不小心创建了一个循环引用，你将会得到 **VBA** 的一个警告提示。

注意：你不能引用嵌入工程或从其它应用程序中的 **VBA** 工程。

设置 VBA

IDE 选项

你可使用选项对话框更改 **VBA IDE** 的特征。可使用工具菜单并选择选项来打开选项对话框。

选项对话框包含四个选项卡：编辑器、编辑器格式、通用和可连接。

编辑器

编辑器选项卡指定代码窗口和工程窗口的设置。

代码设置包括

自动语法检测

要求变量声明

自动列出成员

自动显示快速信息

自动显示数据提示

自动缩进

Tab 宽度

窗口设置包括

编辑时可拖放文本

缺省为查看所有模块

过程分隔符显示

编辑器格式

编辑器格式选项卡指定 **Visual Basic** 代码的外观。

你可以

更改代码的颜色

更改文本列出项

更改前景

更改背景

更改边界标识条

更改文本字体和大小

显示或隐藏边界标识条

显示或隐藏你设置的示例文本

通用

通用选项卡指定当前 **Visual Basic** 工程的设置、出错处理和编译设置。

你可以

更改窗体栅格的栅格设置

显示或隐藏工具提示

设定工程折叠收起时自动隐藏窗口

在丢失当前状态前通知

决定出错时的处理方法

设定工程编译时为需要时编译或进行后台编译。

可连接的

可连接的选项卡允许你选择要进行泊留在主窗口上的窗口(也就是非浮动状态)。

进行介绍性练习

现在你可能已经学习了在 **AutoCAD VBA** 中进行编程的基础知识，那就试试创建一个简单的“**Hello World**”的练习示例。在本练习中你将创建一个新的 **AutoCAD** 图形，在图形中增加一行文本，然后保存图形，所有的这些操作都在 **VBA** 中完成。

创建“**Hello World**”文本对象

1 打开从 **AutoCAD** 的命令行中输入以下命令打开 **VBA IDE**：

命令：VBAIDE

2 在 VBA IDE 的视图菜单中选择代码项打开代码窗口。

3 通过从 VBA IDE 的插入菜单中选择过程项在工程中创建一个新的过程。

4 当提示需要过程信息时，输入如 **Hello World** 这样的名称。确定类型选定的是 **Sub**，而范围选定的是 **Public**。

5 选择确定。

6 在行 **Public Sub Hello World()**和行 **End Sub** 之间输入以下代码(功能是打开新的图形)：

```
ThisDrawing.Application.Documents.Add
```

7 紧接着第 6 步输入以下代码(它创建文本字符并定义它的插入位置)。

```
Dim insPoint(0 To 2) As Double '定义插入点
```

```
Dim textHeight As Double '定义文本高度
```

```
Dim textStr As String '定义文本字符
```

```
Dim textObj As AcadText '定义文本对象
```

```
insPoint(0) = 2 '设定插入点 X 坐标
```

```
insPoint(1) = 4 '设定插入点 Y 坐标
```

```
insPoint(2) = 0 '设定插入点 Z 坐标
```

```
textHeight = 1 '设定文本高度为 1.0
```

```
textStr = "Hello World!" '设定文本字符
```

```
'创建文本对象
```

```
Set textObj = ThisDrawing.ModelSpace.AddText _
```

```
(textStr, insPoint, textHeight)
```

8 紧接着第 7 步输入以下代码(保存图形)

```
ThisDrawing.SaveAs("Hello.dwg")
```

9 通过从 VBA IDE 的运行菜单中选择运行过程/用户窗体项运行你的程序。

当程序运行完成后，回到 AutoCAD 应用程序中，你可看到在图形中出现有“Hello World!”的文本。该图形名称为 Hello.dwg。

更多的信息

关于 VBA IDE 及 Visual Basic 编程语言的更多信息可在微软提供的帮助文件中找到。

访问微软 VBA IDE 帮助文件的操作

从 VBA IDE 的帮助菜单中，选择 Microsoft Visual

Basic 帮助。

回顾 **AutoCAD VBA** 工程术语

常规工程

保存在.dvb 文件的 VBA 工程。

嵌入工程

保存在 AutoCAD 图形中的 VBA 工程。

正常文档

没有包含 VBA 嵌入工程的 AutoCAD 图形。

活跃文档

包含 VBA 嵌入工程的 AutoCAD 图形。

当前工程

在 VBA IDE 中当前选定的工程。

ThisDrawing

ThisDrawing 是用于声称当前图形的 VBA 编程项目。在常规工程中，ThisDrawing 通常指向 AutoCAD 的活跃文档。在嵌入的工程中，ThisDrawing 通过指向包含该工程的文档。

VBA IDE

指的是 VBA 交互开发环境。该应用程序允许你编辑工程中的代码和窗体，或从其它工程中复制代码和窗体。它也允许你设定引用其它应用程序的对象模块。

VBA 管理器

VBA 管理器允许你管理工程。你可以创建、删除、嵌入或分离工程。你也可以查看工程是否嵌入于打开的图形中。

宏对话框

宏对话框允许你运行、删除和创建新的宏，也提供访问 **VBA** 工程选项。

回顾 **AutoCAD VBA** 命令

VBAIDE

打开 **VBA IDE**。

该 **VBA IDE** 允许你交互式地编辑、运行及调试程序。尽管 **VBA IDE** 只能在 **AutoCAD** 运行时才能调用，但它可独立于 **AutoCAD** 应用程序窗口进行最小化、打开和关闭。

VBALOAD

在当前 **AutoCAD** 进程中装载 **VBA** 工程。

VBARUN

从宏对话框中或从 **AutoCAD** 命令行中运行 **VBA** 宏。

VBAUNLOAD

从当前 **AutoCAD** 进程中卸载 **VBA** 工程。

如果 **VBA** 工程已被修改但还没有保存，系统会弹出保存工程对话框(或在命令行)提示是否保存。

VBAMAN

显示 **VBA** 管理器，允许你查看、创建、装载、关闭、嵌入和分离工程。

VBASTMT

从 AutoCAD 命令行中执行 VBA 语句。

第二章 理解 **ActiveX** 自动操作基础

如果想有效地使用 AutoCAD ActiveX 自动操作，你必须熟悉 AutoCAD 图元、对象和与你用于开发的应用程序类型相关联的特性。对于对象的图形和非图形属性认识越深，通过 AutoCAD ActiveX 自动操作对其进行操作就越容易。

记住你随时都可以调出 AutoCAD ActiveX 自动操作的帮助文件-只要按一下 **F1** 键就行。如果你对对象、方法或属性的细节有何不清楚之处，可在 VBA IDE 中选中相应对象、方法或属性并按 **F1** 键。

理解 **AutoCAD** 对象模型

一个对象就是 AutoCAD ActiveX 界面的一个主要组成块。每一暴露的对象描绘了 AutoCAD 的精确部分。在 AutoCAD ActiveX 界面中有许多不同类型的对象。例如

图形对象，如线、弧、文本和标注都是对象

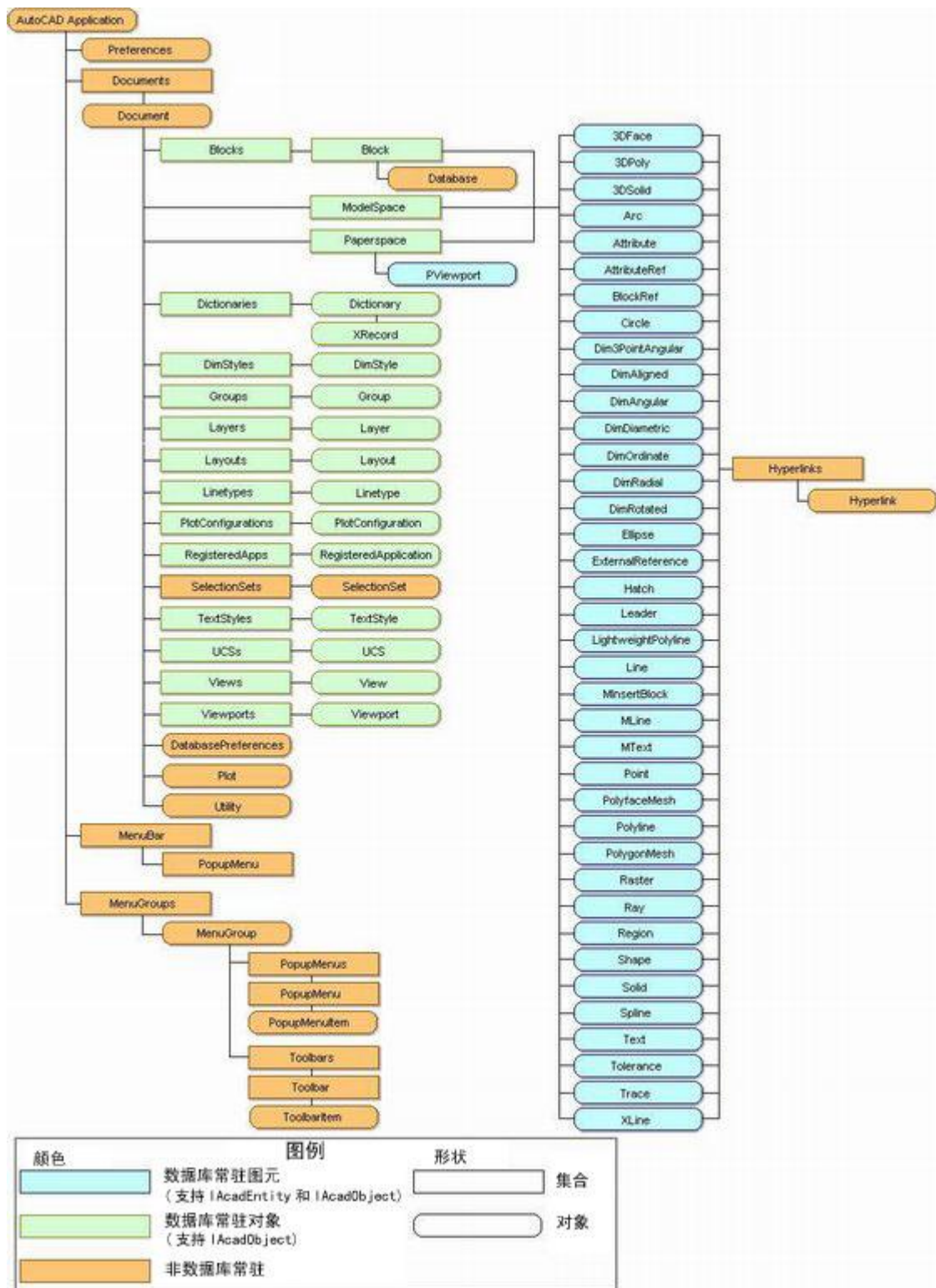
样式设置，如线型和标注样式均为对象

组织结构，如图层、组合和图块也是对象

图形显示，如视图和视口都是对象

甚至图形和 AutoCAD 应用程序本身也是对象

对象是通过分层方式来组织的，应用程序对象为根对象。这种分层结构的视图被归结为对象模型。对象模型提供了你访问下一层对象的途径。



本节内容：

简要介绍 Application(应用程序)对象

简要介绍 Document(文档)对象

简要介绍 Collection(集合)对象

简要介绍图形和非图形对象

简要介绍 Preferences(参数选择)、Plot(打印出图)和 Utility(实用工具)对象

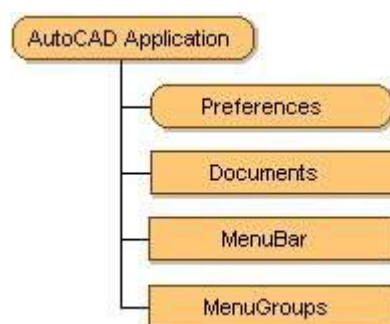
简要介绍 **Application(应用程序)**对象

应用程序对象是 AutoCAD ActiveX 自动操作对象模型的根对象。通过应用程序对象，你可访问其它的对象，或指派对象的属性和方法。

例如，应用程序对象具有 Preferences(参数选择)属性，它返回 Preferences(参数选择)对象。该对象提供访问在选项对话框中设定的注册信息。(图形信息设定包含在 DatabasePreferences 对象，它在后面会介绍到。)

应用程序对象的其它属性提供你访问应用程序指定数据，如应用程序的名称和版本、还有 AutoCAD 的窗口大小、位置和可见性等。应用程序对象的方法执行应用程序指定的动作，如列出、装载、卸载 ADS 和 ARX 应用程序，还有退出 AutoCAD。

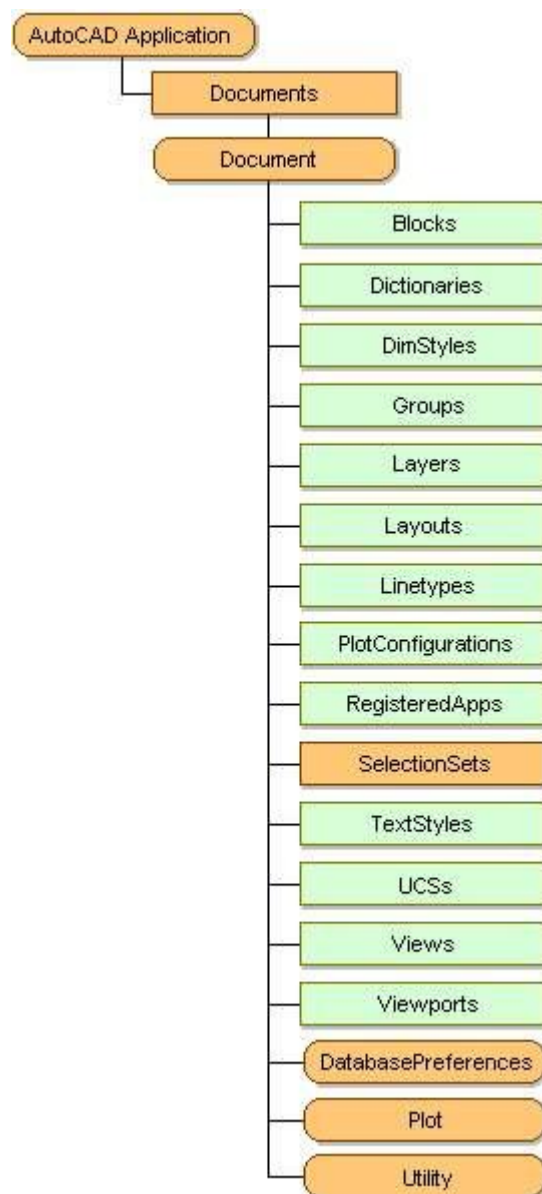
应用程序对象也提供通过 Documents(文档)集合链接到 AutoCAD 图形、通过 MenuBar 和 MenuGroups 集合链接到 AutoCAD 菜单和工具栏，还有通过称为 VBE 属性链接到 VBA IDE。



应用程序对象也是 ActiveX 界面的全局对象。也就是说应用程序对象的所有方法和属性在全局名称空间都是有效的。

简要介绍 Document(文档)对象

文档对象，实际上就是 AutoCAD 图形，它可在 Documents(文档)集合中找到，它提供访问所有图形还有大部分非图形的 AutoCAD 对象。通过提供的 ModelSpace(模型空间)和 PaperSpace(图纸空间)访问图形对象(线、圆、弧等)，通过提供的如 Layers(图层)、Linetypes(线型)和 TextStyles(字型)这样名称的集合访问非图形对象(图层、线型、字型等)。Document(文档)对象也提供访问 Plot(打印出图)和 Utility(实用工具)对象。



简要介绍 Collection(集合)对象

AutoCAD 组合大部分的对象在集合中。尽管这些集合包含不同类型的数据，但它们是通过相似的技术进行处理。每一集合都有添加对象到集合中的方法。大多数集合使用 **Add**(添加)方法以达到目的。不同的是，图元对象通常使用标头为 **Add<图元名称>**这样的方法来进行添加。例如，添加一条直线你必须使用 **AddLine** 方法。

集合也拥有共用的一些其它方法和属性。**Count**(记数)属性可用于获取集合中对象的数目。**Item**(项目)方法可用于获取集合中的对象。

简要介绍图形和非图形对象

图形对象，也称为图元，它是构成图形的可见对象(如直线、圆、光栅图像等)。创建这些对象，可用近似于 **Add<图元名称>**方法。修改或查询这些对象，可使用对象自身的方法和属性。每一图形对象都有方法允许应用程序执行大部分的 AutoCAD 编辑命令，如复制、删除、移动、镜像还有其它。这些对象也有相应方法可以设定及返回扩展数据(**xdata**)、高亮和更新、返回对象的边框范围等。图形对象具有如图层、线型、颜色和句柄这样典型的属性。它们也有其特殊的依赖于这些对象类型的属性，如圆心、半径和面积等。

非图形对象为不可见(指示性)对象，它们是图形的一部分，就如 **Layers**(图层)、**Linetypes**(线型)、**DimStyles**(标注样式)、**SelectionSets**(选择集)等。创建这些对象，可在父集合对象中使用 **Add**(添加)方法。而修改或查询这些对象，可使用对象自身的方法和属性。每一非图形对象均有其特殊的方法及属性以达到其目的；所有对象都有方法可以设定及返回扩展数据。

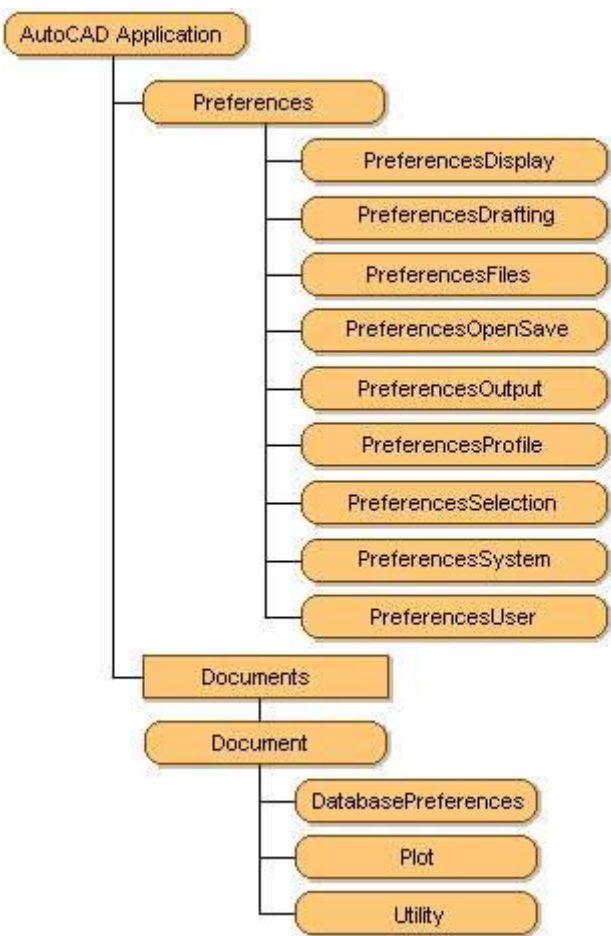
简要介绍 Preferences(参数选择)、Plot(打印出图)和 Utility(实用工具)对象

在参数选择(**Preferences**)对象下边为对象集，每一对象集都与选项对话框的一个选项卡相关。它们共同提供访问在选项对话框中所有的注册设置。图形设置包含在数据库参数选择(**DatabasePreferences**)对象中。你也可通过 **SetVariable** 和 **GetVariable** 方法来设定和修改选项(系统变量不是选项对话框的一部分)。设置选项的详细内容请参考“设定 AutoCAD 参数”。

打印出图(**Plot**)对象提供访问在打印对话框的设定，并且让其它程序可以使用不同的方法打印图形。图形打印的相关内容请参考“打印图形”。

实用工具(**Utility**)对象提供用户输入和转换功能。用户输入功能是在 AutoCAD 命令行中提示用户输入不

同类型数据的方法，如输入字符串、整数、实数、点等。转换功能是操作 AutoCAD 特有数据类型的方法，如点和角度，另外还有字符串和数字的处理。用户输入功能的相关内容请参考“提示用户输入”。



访问对象层次

在 VBA 内部访问对象层次是非常容易的。这是因为 VBA 是运行在 AutoCAD 进程的内部，所以不必通过附加的步骤与应用程序进行连接。

VBA 通过 ThisDrawing 对象链接到当前 AutoCAD 进程的活动图形。通过使用 ThisDrawing 直接获得访问当前文档对象和所有相关的方法和属性以及所有在该层次中的其它对象。

当使用全局工程，ThisDrawing 通常指向 AutoCAD 中的活动文档。当用的是嵌入工程，ThisDrawing 通常指向包含该工程的文档。例如，以下在全局工程中的代码行将保存当前 AutoCAD 中活动的图形：

`thisDrawing.Save`

在对象层次中引用对象

你可直接通过用户定义变量引用对象。直接引用对象，应包括对象的层次。例如，以下语句增加一条直线到模型空间。注意到层次状态为 **ThisDrawing**，然后是 **ModelSpace**(模型空间)对象，最后才是 **AddLine** 方法：

```
Dim startPoint(0 To 2) As Double, endPoint(0 To 2) As
```

```
Double
```

```
Dim LineObj as AcadLine
```

```
startPoint(0) = 0: startPoint(1) = 0: startPoint(2)
```

```
= 0
```

```
endPoint(0) = 30: endPoint(1) = 20: endPoint(2) = 0
```

```
Set LineObj = ThisDrawing.ModelSpace.AddLine(startPoint,endPoint)
```

通过用户定义变量引用对象，首先定义变量的类型，然后设定变量为适当的对象。例如，以下代码定义了一个类型为 **AcadModelSpace** 的变量(**moSpace**)并设定变量等于当前模型空间：

```
Dim moSpace
```

```
As AcadModelSpace
```

```
Set moSpace = ThisDrawing.ModelSpace
```

以下语句使用用户定义变量增加一条直线到模型空间：

```
Dim startPoint(0
```

```
To 2) As Double, endPoint(0 To 2) As Double
```

```
Dim LineObj as AcadLine
```

```
startPoint(0) = 0: startPoint(1) = 0: startPoint(2)
```

```
= 0
```

```
endPoint(0) = 30: endPoint(1) = 20: endPoint(2) = 0
```

```
Set LineObj = moSpace.AddLine(startPoint,endPoint)
```

访问应用程序对象

因为 **ThisDrawing** 对象提供与文档对象的链接，你可能会觉得奇怪，在文档对象之上对象层次的根对象(应用程序对象)是怎样访问的。其实文档对象中有一称为应用程序的属性可提供与应用程序对象的链接。

例如，以下代码行更新应用程序：

```
ThisDrawing.Application.Update
```

通过集合对象操作

集合对象是预先定义的对象，它包含所有相似对象的实例(即这些对象的父对象)。集合对象有以下的对象：

文档(**Documents**)集合

包含所有在当前 **AutoCAD** 进程打开的文档。

模型空间(**ModelSpace**)集合

包含在模型空间中的所有图形对象(图元)。

图纸空间(**PaperSpace**)集合

包含在活动图纸空间布局中的所有图形对象(图元)。

图块(Block)对象

包含在指定图块定义中的所有图元。

图块(Blocks)集合

包含在图形中的所有图块。

字典(Dictionaries)集合

包含在图形中的所有字典。

标注样式(DimStyles)集合

包含在图形中的所有标注样式。

组合(Groups)集合

包含在图形中的所有组合。

超级链接(Hyperlinks)集合

包含提供图元的所有超级链接。

图层(Layers)集合

包含在图形中的所有图层。

布局(Layouts)集合

包含在图形中的所有布局。

线型(Linetypes)集合

包含在图形中的所有线型。

菜单条(MenuBar)集合

包含当前显示于 AutoCAD 的所有菜单。

菜单组(MenuGroups)集合

包含当前装载到 AutoCAD 中的所有菜单和工具栏。

注册应用程序(RegisteredApplications)集合

包含在图形中的所有注册的应用程序。

选择集(SelectionSets)集合

包含在图形中所有的选择集。

字型(TextStyles)集合

包含在图形中所有的文字样式。

UCSs 集合

包含在图形中所有的用户坐标系统(UCS)。

视图(Views)集合

包含在图形中所有的视图。

视口(Viewports)集合

包含在图形中所有的视口。

本节内容：

访问集合

添加新成员到集合对象

在集合对象中循环

删除集合对象中的成员

访问集合

大多数集合对象是通过文档对象来访问的。文档对象包含每个集合对象的属性。例如，以下代码定义一个变量并将其设定到当前图形的图层集合中：

```
Dim layerCollection
```

```
as AcadLayers
```

```
Set layerCollection = ThisDrawing.Layers
```

文档集合、菜单条集合和菜单组集合是通过应用程序对象进行访问。应用程序对象为这些集合的每一个集合包含一个属性。例如，以下代码定义了一个变量并且设定该变量为应用程序中的菜单组集合：

```
Dim MenuGroupsCollection
```

```
as AcadMenuGroups
```

```
Set MenuGroupsCollection = ThisDrawing.Application.MenuGroups
```

添加新成员到集合对象

添加新成员到集合中使用的是 **Add** 方法。例如，以下代码创建一个新的图层并且将其添加到图层集合中：

```
Dim newLayer
```

```
as AcadLayer
```

```
Set newLayer = ThisDrawing.Layers.Add("MyNewLayer")
```

在集合对象中循环

选择集合对象中的一个指定成员，使用的是 **Item(项目)**方法。**Item** 方法需要一个标识符。该标识符可以是指定集合内部项目位置的索引号或者描述项目名称的字符串。

以下例子在集合中循环并显示集合中所有图层的名称：

在图层集合中循环

```
Sub Ch2_IterateLayer()
```

```
' 在图层集合中循环
```

```
On Error Resume Next
```

```
Dim I As Integer
```

```
Dim msg As String
```

```
msg = ""
```

```
For I = 0 To ThisDrawing.Layers.count - 1
```

```
msg = msg + ThisDrawing.Layers.Item(I).Name + vbCrLf
```

```
Next
```

```
MsgBox msg
```

```
End Sub
```

以下例子使用 **Item** 方法查找名称为“ABC”的图层：

查找名称为“ABC”的图层

```
Sub Ch2_FindLayerABC()
```

```
' 使用 Item 方法查找名称为"ABC"的图层
```

```
On Error Resume Next
```

```
Dim ABCLayer As AcadLayer
```

```
Set ABCLayer = ThisDrawing.Layers.Item("ABC")
```

```
If Err <> 0 Then
```

```
MsgBox "图层"ABC"并不存在。"
```

```
End If
```

```
End Sub
```

注意：当使用 **For Each** 机制同时在集合循环时，不要在对象上使用图元编辑方法(复制、阵列、镜像等)，你只能在完成循环之后才可试着去编辑对象，或者先创建一个临时的数组并且设定其与集合相等，然后才可以在该复制的阵列中循环并执行你的操作。

删除集合对象中的成员

删除指定的成员，可使用所找到成员对象的 **Delete**(删除)方法。例如，以下代码删除图层 **ABC**：

```
Dim ABCLayer
```

```
as AcadLayer
```

```
Set ABCLayer = ThisDrawing.Layers.Item("ABC")
```

`ABCLayer.Delete`

当对象被删除后，你再也不能试图用程序去访问该对象了。

理解属性和方法

每一对象都关联着属性和方法。属性描述着单个对象的外观，而方法是一种可在单个对象上执行的行为。

当对象创建后，你就可通过属性和方法查询和编辑对象。

例如，一个圆对象有圆心属性。该属性以三维世界坐标系统的坐标描述了圆的圆心。更改圆的圆心，你只要简单地将该属性设定为新的坐标。圆对象也有称为偏移(**Offset**)的方法。该方法可在相对于现存圆的指定偏移距离创建一个新的对象。关于圆对象所有属性和方法的列表，请参考 **AutoCAD**

ActiveX 和 **VBA** 参考中的圆对象。

理解父对象

每一个对象都有其永远不变的父对象。所有对象都源于称为根对象的单个父对象。你可以通过从根对象链接到子对象的方法访问所有在界面中的对象。还有，所有对象都有称为应用程序的属性以直接链接回根对象。

AutoCAD 界面的根对象为 **AutoCAD** 应用程序。

定位类型库

通过自动操作对象暴露的对象、属性和方法都包含在类型库中。类型库是一个文件或一个文件的一部分，它描述一个或多个对象的类型。类型库并不保存对象；它们只保存信息。通过访问类型库，应用程序和浏览器可确定对象的特征，如对象支持的界面和每一界面成员的名称和地址。

在使用通过应用程序暴露出来的自动操作对象，你必须引用其类型库。该引用已由 **AutoCAD**

VBA 自动设定好。而在其它的交互开发环境中你必须创建这个引用。

你可以在没有引用应用程序类型库的情况下使用应用程序对象。然而，由于以下理由，可考虑增加类型库引用：

全局可访问函数可无条件直接访问。

调用函数、属性和方法可在编译时检查其正确性，这样可以在运行时有更快的速度。

有可能的话可声明定义在库中的变量类型，这样可增加运行时的可靠性和可读性。

在数据库中返回第一个图元

以下例子返回模型空间中的第一个图元对象。对于图纸空间中的图元，代码略有不同：

返回模型空间中的第一个图元

```
Sub Ch2_FindFirstEntity()
```

```
' 这个例子返回模型空间中的第一个图元
```

```
On Error Resume Next
```

```
Dim entity As AcadEntity
```

```
If ThisDrawing.ModelSpace.count <> 0 Then
```

```
Set entity = ThisDrawing.ModelSpace.Item(0)
```

```
MsgBox entity.ObjectName + _
```

```
" 是在模型空间中的第一个图元。"
```

Else

MsgBox "在模型空间中没有对象存在。"

End If

End Sub

在方法和属性中使用变体

AutoCAD ActiveX 自动操作是使用变体传递数组数据。尽管这对于初学者来说有点弄不懂，但只要你有点基础知识就不会觉得困难。另外，AutoCAD ActiveX 自动操作提供了帮助你转换数据类型的工具。

本节内容：

什么是变体？

在数组数据中使用变体

转换数组为变体

解释变体数组

什么是变体？

变体是一种特殊的数据类型，它可包含除固定长度字符串数据和用户定义类型外的其它任何类型的数据。

变体也可包含特殊的值，如 Empty、Error、Nothing 和 NULL。你可通过 VarType 或 TypeName 这样的 Visual

Basic 函数来确定变体中数据。

你可使用变体数据类型来放置大多数的任何数据类型，来使工作途径更灵活。

在数组数据中使用变体

变体是用于传递数组数据进和出 AutoCAD ActiveX 自动操作。也就是说你的数组必须为变体以由 AutoCAD

ActiveX 自动操作的方法和属性所接受。另外，从 AutoCAD ActiveX 自动操作中输出的数组数据必须处理为变体。

注意：在 AutoCAD，VBA 中输入数组会自动转换为变体。也就是说你在 VBA 中使用的数组不必将其转换为变体数组后才输入 ActiveX 自动操作的方法和属性。然而，所有输出的数组将是变体的形式，所以请记住要进行适当的处理。

转换数组为变体

AutoCAD ActiveX 自动操作提供了实用方法以转换数组中的数据为变体。该方法为 `CreatTypedArray` 方法，它创建包含整数、浮点数、双精度数等数组的变体。你可传递这些结果的变体到任何接受作为变体的数据数值的 AutoCAD 方法或属性。

`CreatTypedArray` 方法接受在数组中输入的数值类型，和转换的数组数据。它返回数组数值为变体。以下代码使用 `CreateTypedArray` 转换有三个数组：样条曲线的拟合点坐标、样条曲线的起点切点和终点切点。然后传递这些变体到 `AddSpline` 方法以创建样条曲线。

使用 `CreateTypedArray` 方法创建样条曲线

```
Sub Ch2_CreateSplineUsingTypedArray()
```

```
' 这个例子在模型空间中利用 CreateTypedArray 方法
```

```
' 创建样条曲线。
```

```
Dim splineObj As AcadSpline
```

```
Dim startTan As Variant
```

```
Dim endTan As Variant
```

```
Dim fitPoints As Variant
```

```
Dim noOfPoints As Integer
```

```
Dim utilObj As Object ' 然后绑定在实用工具对象
```

```
Set utilObj = ThisDrawing.Utility
```

```
' Define the Spline Object
```

```
utilObj.CreateTypedArray _
```

```
startTan, vbDouble, 0.5, 0.5, 0
```

```
utilObj.CreateTypedArray _
```

```
endTan, vbDouble, 0.5, 0.5, 0
```

```
utilObj.CreateTypedArray _
```

```
fitPoints, vbDouble, 0, 0, 0, 5, 5, 0, 10, 0, 0
```

```
noOfPoints = 3
```

```
Set splineObj = ThisDrawing.ModelSpace.AddSpline _
```

```
(fitPoints, startTan, endTan)
```

```
' 缩放查看该新创建的样条曲线
```

```
ZoomAll
```

```
End Sub
```

解释变体数组

从 AutoCAD ActiveX 自动操作传递回的数组信息是以变体的方式传递回。如果你知道数组中的数据类型，你可以简单地将变体作为数组访问。如果你不知道包含在变体中的数据类型，可使用 VBA 函数 `VarType` 或 `TypeName` 来处理。这两个函数返回在变体中的数据类型。如果你需要在数组中循环，你可使用 VBA 的 `For` or

`Each` 语句。

以下代码示范了计算由用户输入的两点的距离。在该例中，数据类型已经知道，因为所有坐标均为双精度。三维坐标是双精度的三元素数组，而二维坐标是双精度的二元素数组。

计算两点之间的距离

```
Sub Ch2_CalculateDistance()
```

```
Dim point1 As Variant
```

```
Dim point2 As Variant
```

' 由用户提供点

point1 = ThisDrawing.Utility.GetPoint _

(, vbCrLf & "第一点: ")

point2 = ThisDrawing.Utility.GetPoint _

(point1, vbCrLf & "第二点: ")

' 计算点 point1 和点 point2 之间的距离

Dim x As Double, y As Double, z As Double

Dim dist As Double

x = point1(0) - point2(0)

y = point1(1) - point2(1)

z = point1(2) - point2(2)

dist = Sqr((Sqr((x ^ 2) + (y ^ 2)) ^ 2) + (z ^ 2))

'显示距离的结果

MsgBox "两点之间的距离为: " _

& dist, , "个计算单位"

End Sub

使用其它程序语言

本指南和 AutoCAD ActiveX 及 VBA 参考是为 VBA 编程语言所编写的。所以程序中的例子和示例应用程序都在 VBA 中编写。如果要将这些代码用于其它编辑环境，你必须将其更新到所选择的环境中。

本节内容：

转换 VBA 代码到 VB 中

转换 VBA 代码到 VB 中

更新代码示例以用于 VB 中，你必须首先引用 AutoCAD 类型库。在 VB 中，可从工程菜单中选择引用项弹出引用对话框。从引用对话框中，选择 AutoCAD Release 15 类型库并按确定。

下一步，在代码示例中使用引用活动文档的用户指定变量替换所有引用的 ThisDrawing。要做到这一点，可定义 AutoCAD 应用程序变量(acadApp)和当前文档变量(acadDoc)。然后，设定应用程序变量到当前 AutoCAD 应用程序。

如果 AutoCAD 正在运行，用 GetObject 方法返回 AutoCAD 应用程序对象。如果 AutoCAD 没有运行，则会捕获到一个错误发生(对于本例)，然后清除它。接下来用 CreateObject 方法试图创建 AutoCAD 应用程序对象。如果成功，则会启动 AutoCAD；如果失败，则会弹出一信息框显示出错的内容。以下代码示例使用了 Err 的 Clear 和 Description 属性。如果你的代码环境不支持这些属性，你需要适当修改示例。

从 Visual Basic 中连接到 AutoCAD

```
Sub Ch2_ConnectToAcad()
```

```
Dim acadApp As AcadApplication
```

```
On Error Resume Next
```

```
Set acadApp = GetObject( "AutoCAD.Application")
```

```
If Err Then
```

```
Err.Clear
```

```
Set acadApp = CreateObject("AutoCAD.Application")
```

```
If Err Then
```

```
MsgBox Err.Description
```

```
Exit Sub
```

```
End If
```

```
End If
```

```
MsgBox "现在运行 " + acadApp.Name + _
```

```
" 版本号 " + acadApp.Version
```

```
End Sub
```


下一步，设定文档变量到 AutoCAD 应用程序中的文档对象。文档对象是由应用程序对象的 `ActiveDocument` 属性返回的。

```
Dim acadDoc as AcadDocument
```

```
Set acadDoc = acadApp.ActiveDocument
```

通过以上操作，使 `acadDoc` 变量引用当前 AutoCAD 图形。

注意：当运行多个进程的 AutoCAD，`GetObject` 函数将返回 Windows 运行对象表中 AutoCAD 的第一个实例。请参考 Microsoft

Visual Basic 文档中关于运行对象表(ROT)和 `GetObject` 函数部分以取得更多关于检验 `GetObject` 返回进程的内容。

VBA 相对于 VB 对照代码示例

以下代码示例示范了通过 VBA 和 VB 创建一条直线：

使用 VBA 创建一条线

```
Sub Ch2_AddLineVBA()
```

```
' 本例在模型空间中添加一条直线
```

```
Dim lineObj As AcadLine
```

```
Dim startPoint(0 To 2) As Double
```

```
Dim endPoint(0 To 2) As Double
```

```
' 为直线定义起点和终点
```

```
startPoint(0) = 1
```

```
startPoint(1) = 1
```

```
startPoint(2) = 0
```

```
endPoint(0) = 5
```

```
endPoint(1) = 5
```

```
endPoint(2) = 0
```

```
' 在模型空间中创建该直线
```

```
Set lineObj = ThisDrawing. _
```

```
ModelSpace.AddLine _
```

```
(startPoint, endPoint)
```

```
' 缩放以显示新创建的直线
```

```
ZoomAll
```

```
End Sub
```

使用 VB 创建一条直线

```
Sub Ch2_AddLineVB()
```

```
On Error Resume Next
```

```
' 连接到 AutoCAD 应用程序
```

```
Dim acadApp As AcadApplication
```

```
Set acadApp = GetObject _
```

```
(, "AutoCAD.Application")
```

```
If Err Then
```

```
Err.Clear
```

```
Set acadApp = CreateObject _
```

```
("AutoCAD.Application")
```

```
If Err Then
```

```
MsgBox Err.Description
```

```
Exit Sub
```

```
End If
```

```
End If
```

' 连接到 AutoCAD 图形

Dim acadDoc As AcadDocument

Set acadDoc = acadApp.ActiveDocument

' 确定直线的两个端点

Dim lineObj As AcadLine

Dim startPoint(0 To 2) As Double

Dim endPoint(0 To 2) As Double

startPoint(0) = 1

startPoint(1) = 1

startPoint(2) = 0

endPoint(0) = 5

endPoint(1) = 5

endPoint(2) = 0

' 在模型空间创建一条直线

```
Set lineObj = acadDoc.ModelSpace.AddLine _
```

```
(startPoint, endPoint)
```

```
ZoomAll
```

```
End Sub
```

第三章 控制 **AutoCAD** 环境

本章将详述在 **AutoCAD** 中开发应用程序时的基础知识。它介绍了怎样控制 **AutoCAD** 环境及怎样有效地工作于该环境中。

打开、保存和关闭图形

Documents(文档)集合和 **Document**(文档)对象提供了访问 **AutoCAD** 文件的功能。

创建一个新的图形，或打开现存的图形，应使用 **Documents**(文档)集合中的方法。**Add**(添加)方法创建一个新的文档并将该文档添加到 **Documents**(文档)集合中。**Open**(打开)方法打开一个现存的图形。在 **Documents**(文档)集合中也有一 **Close**(关闭)方法以关闭在 **AutoCAD** 中打开的所有图形。

保存、输入、输出图形，可使用 **Document**(文档)对象的 **Save**, **SaveAs**, **Import** 和 **Export** 方法。

打开图形

本例使用 **Open** 方法打开一个现存的图形。其中 **Visual Basic** 的 **Dir** 函数用于在打开文件前检查该文件是否存在。你可更改文件的名称和路径为你系统中存在的 **AutoCAD** 图形文件。

```
Sub Ch3_OpenDrawing()
```

```
Dim dwgName As String
```

```
dwgName = "c:\Program Files\acad2000\sample\campus.dwg"
```

```
If Dir(dwgName) <> "" Then
```

```
ThisDrawing.Application.Documents.Open dwgName
```

```
Else
```

```
MsgBox "文件 " & dwgName & " 并不存在。"
```

```
End If
```

```
End Sub
```

创建新图形

本例使用 **Add** 方法创建一个基于默认模板的新图形。

```
Sub Ch3_NewDrawing()
```

```
Dim docObj As AcadDocument
```

```
Set docObj = ThisDrawing.Application.Documents.Add
```

```
End Sub
```

本节内容：

保存图形

保存图形

可使用 **Save** 或 **SaveAs** 方法保存图形。

保存活动的图形

以下示例以当前名称保存活动图形然后再以新的名称保存该图形。

```
Sub Ch3_SaveActiveDrawing()
```

```
' 以当前名称保存活动图形
```

```
ThisDrawing.Save
```

```
' 以新的名称保存活动图形
```

```
ThisDrawing.SaveAs "MyDrawing.dwg"
```

```
End Sub
```

当然你会想到去检查活动图形是否存在有未保存的更改。在退出 **AutoCAD** 进程之前或开始一新的图形之前这样做是一种好的习惯。使用 **Saved** 属性可以确定当前图形是否包含未保存的更改。

测试图形是否有未保存的更改内容

本例检查图形是否有未保存的更改并由用户确定是否保存图形(如果取消，则直接跳到结束处)。如果确定的话，使用 **Save** 方法保存当前图形：

```
Sub Ch3_TestIfSaved()
```

```
If Not (ThisDrawing.Saved) Then
```

```
If MsgBox("你是否想保存该图形?", _
```

```
vbYesNo) = vbYes Then
```

```
ThisDrawing.Save
```

```
End If
```

```
End If
```

```
End Sub
```

设定 **AutoCAD** 参数

一共有 9 个属于选项的对象，每一对象描述了选项对话框中的一个选项卡。这些对象提供了访问选项对话框中的所有注册信息项。你可以通过使用在这些对象中的属性来自定义 **AutoCAD** 设置。这些对象为

PreferencesDisplay (显示参数)

PreferencesDrafting (草图参数)

PreferencesFiles (文件参数)

PreferencesOpenSave (打开和保存参数)

PreferencesOutput (输出参数)

PreferencesProfile (配置参数)

PreferencesSelection (选择参数)

PreferencesSystem (系统参数)

PreferencesUser (用户参数)

这些对象可通过 Preferences(参数)对象来访问。至于对 Preferences 对象的访问可使用 Application(应用程序)对象中的 Preferences 属性。示例如下：

访问 Preferences(参数)对象

```
Dim acadPref as AcadPreferences
```

```
Set acadPref = ThisDrawing.Application.Preferences
```

然后你可以使用 Display、Drafting、Files、OpenSave、Output、Profile、Selection、System 和 User 属性。

例如，你可通过 CursorSize 属性控制十字光标的大小。

设定十字光标为全屏

本例设定十字光标为全屏。

```
acadPref.Display.CursorSize = 100
```

你可以通过应用程序使 AutoCAD 界面的某些外观启用或禁用。

显示屏幕菜单和滚动条

本例用 DisplayScreenMenu 和 DisplayScrollBars 属性使屏幕菜单可见，使滚动条不可见。

```
acadPref.Display.DisplayScreenMenu
```

```
= True
```

```
acadPref.Display.DisplayScrollBars = False
```

本节内容：

数据库参数

数据库参数

DatabasePreferences 对象是在 9 个 **Preferences** 对象之外，它包含所有保存于图形中的选项。这个单独对象提供了在没有启动 **AutoCAD** 应用程序时使图形信息有效以便应用程序访问 **AutoCAD** 图形(**ObjectDBX** 应用程序)。

DatabasePreferences 对象可以在 **Document** 对象下找到。

控制应用程序窗口

控制应用程序窗口的能力允许开发者机动地创建有效而智能的应用程序。可能你的代码要在如 **Excel** 这样的程序中执行，就可通过这种方法及时地最小化 **AutoCAD** 窗口。另外，你可能会在执行诸如提示用户输入之类的任务之前确定 **AutoCAD** 窗口的状态。

使用 **Application** 对象中的方法和属性，你可以更改应用程序窗口的位置、大小和可见性。你也可以最小化或最大化应用程序窗口，以及检查其当前状态。

本节内容：

更改应用程序窗口的位置和大小

最小化和最大化 **AutoCAD** 窗口

查找 AutoCAD 窗口的当前状态

使应用程序窗口不可见

更改应用程序窗口的位置和大小

你可以利用 **Application** 对象更改 AutoCAD 应用程序窗口的位置和大小。

调整应用程序窗口的位置

本例使用 **WindowTop**、**WindowLeft**、**Width** 和 **Height** 属性调整 AutoCAD 应用程序窗口相对于屏幕左上角的位置及调整窗口大小为 400 像素宽×400 像素高。

```
Sub Ch3_PositionApplicationWindow()
```

```
ThisDrawing.Application.WindowTop = 0
```

```
ThisDrawing.Application.WindowLeft = 0
```

```
ThisDrawing.Application.width = 400
```

```
ThisDrawing.Application.height = 400
```

```
End Sub
```

最小化和最大化 AutoCAD 窗口

AutoCAD 窗口可通过使用 **WindowState** 属性进行最小化和最大化。以下例子示范了最大化和最小化应用程序窗口。

最大化应用程序窗口

```
ThisDrawing.Application.WindowState
```

```
= acMax
```

最小化应用程序窗口

```
ThisDrawing.Application.WindowState
```

```
= acMin
```

查找 AutoCAD 窗口的当前状态

你可以通过使用 `WindowState` 属性查找 AutoCAD 窗口的当前状态。

查找应用程序的当前状态

本例查询应用程序窗口状态并在消息框里显示其状态。

```
Sub Ch3_CurrentWindowState()
```

```
Dim CurrWindowState As Integer
```

```
Dim msg As String
```

```
CurrWindowState = ThisDrawing.Application.WindowState
```

```
msg = Choose(CurrWindowState, "正常", _
```

```
"最小化", "最大化")
```

```
MsgBox "应用程序窗口为" + msg
```

```
End Sub
```

使应用程序窗口不可见

你也可以使应用程序窗口在用户的桌面上不可见。

使应用程序窗口不可见

以下代码使用 **Visible**(可见性)属性使 **AutoCAD** 应用程序不可见。

```
ThisDrawing.Application.Visible
```

```
= False
```

控制图形窗口

和 **AutoCAD** 应用程序窗口一样，你可最小化、最大化、调整位置、调整大小和检查任何文档窗口的状态。

同时你也可通过使用 **Views**(视图)、**viewports**(视口)和 **zooming**(缩放)方法更改图形在窗口中的显示方式。

AutoCAD ActiveX 提供多种方式来显示图形中的视图，当你要跟踪所做更改的全部效果时，你可控制图形显示以快速移动到图形中的不同区域。你可进行缩放以更改显示倍率或平移以调整图形区域的视图位置，你可保存视图，然后在你需要打印或指向指定的细节时恢复视图，或者通过将屏幕拆分成多个平铺视口以显示多个视图。

本节内容：

更改文档窗口的位置和大小

最小化和最大化文档窗口

查找文档窗口的当前状态

使用缩放

使用命名视图

使用平铺视口

更新文档窗口的几何图形

更改文档窗口的位置和大小

使用 **Document** 对象更改文档窗口的位置和大小。

调整文档窗口的位置

本例使用 **Width** 和 **Height** 属性设定活动文档窗口为 400 像素宽×400 像素高。

```
ThisDrawing.Width = 400
```

```
ThisDrawing.Height = 400
```

最小化和最大化文档窗口

文档窗口可通过使用 **WindowState** 属性进行最小化或最大化。

最大化活动文档窗口

```
ThisDrawing.WindowState
```

```
= acMax
```

最小化活动文档窗口

```
ThisDrawing.WindowState
```

```
= acMin
```

查找文档窗口的当前状态

你可通过使用 **WindowState** 查找文档窗口的当前状态。

查找活动文档窗口的当前状态

```
Sub Ch3_CurrentWindowState()
```

```
Dim CurrWindowState As Integer
```

```
Dim msg As String
```

```
CurrWindowState = ThisDrawing.WindowState
```

```
msg = Choose(CurrWindowState, "正常", _
```

```
"最小化", "最大化")
```

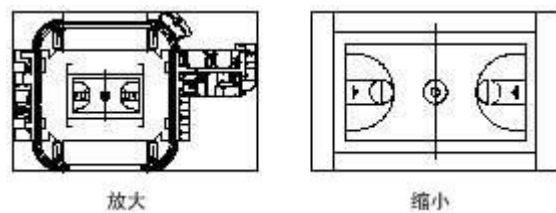
```
MsgBox "文档窗口当前状态为 " + msg
```

```
End Sub
```

使用缩放

视图是图形中特定的放大倍率、位置和方位，更改视图最常用的途径为使用 AutoCAD 的 Zoom(缩放)项，它可增加或减小显示于图形区域中图像的大小。

扩大图像以更接近地查看详细资料称为放大，而收缩图像以看到图形的更大面积称为缩小。



缩放不会更改图形的绝对尺寸，它只更改在图像区域中视图的大小。AutoCAD 提供多种途径更改视图，包括指定显示窗口、缩放到指定比例和显示整个图形。

本节内容：

定义缩放窗口

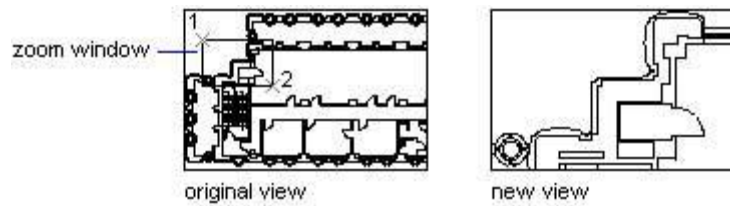
调整视图比例

中心缩放

显示图形界限和范围

定义缩放窗口

你可通过指定对象点快速缩放到某一区域。



如果通过对角点指定的区域不能与视口完全匹配的话，所选定的部分将会显示在中心。你指定的缩放窗口的边缘不必与新视图刚好相同。

通过指定外框而放大到某一区域，可用 **ZoomWindow** 或 **ZoomPickWindow** 方法。**ZoomWindow** 方法允许你通过程序定义缩放窗口的两个点。而 **ZoomPickWindow** 需要用户拾取两个点。拾取的两个点将形成缩放窗口。

通过两个点形成的窗口缩放活动图形

```
Sub Ch3_ZoomWindow()
```

```
' ZoomWindow 方法
```

```
MsgBox "通过以下两点执行 ZoomWindow 方法:" & vbCrLf
```

```
& _
```

```
"1.3, 7.8, 0" & vbCrLf & _
```

```
"13.7, -2.6, 0", , "ZoomWindow"
```

```
Dim point1(0 To 2) As Double
```

```
Dim point2(0 To 2) As Double
```

```
point1(0) = 1.3: point1(1) = 7.8: point1(2) = 0
```



```
point2(0) = 13.7: point2(1) = -2.6: point2(2) = 0
```

```
ThisDrawing.Application.ZoomWindow point1, point2
```

```
' ZoomPickWindow 方法
```

```
MsgBox "执行 ZoomPickWindow 方法", , "ZoomPickWindow"
```

```
ThisDrawing.Application.ZoomPickWindow
```

```
End Sub
```

调整视图比例

如果你需要通过精确的比例增加或减小图像的倍率，你可用以下三种途径指定缩放比例。

- 相对于图形界限
- 相对于当前视图
- 相对于图纸空间单位

当相对于图形界限调整视图比例时，提供简单的比例值“1”将尽可能大地显示图形区域，而中心将会是先前视图的中心。放大或缩小可输入大于或小于的数字。正如下例，你可输入 2 以显示图像的 2 倍尺寸大小，或输入 0.5 以显示一半尺寸大小的图像。

当相对于当前视图调整视图比例，输入 2 为两倍当前视图大小，或 0.5 显示一半当前视图大小。而输入 1 时将不会产生效果。

当相对于图纸空间单位调整视图比例时，你提供的比例值或增加或减小视图相对于当前图纸空间的比例。它用于打印前调整视口比例。

创建比例缩放视图

通过比例调整视图，使用的是 **ZoomScaled** 方法。该方法有两个参数输入：缩放比例和比例的类型。缩放比例只是个数字。该数字是依赖于你选择的比例类型而为 **AutoCAD** 所接受。

比例的类型是确定比例值是相对于图形界限、当前视图或图纸空间单位而创建。相对于图形界限调整比例，使用常数 **acZoomScaledAbsolute**。相对于当前视图调整比例，使用常数 **acZoomScaledRelative**。相对于图纸空间单位调整比例，使用常数 **acZoomScaledRelativePSpace**。

使用指定比例缩放活动图形

```
Sub Ch3_ZoomScaled()
```

```
MsgBox "使用以下参数执行 ZoomScaled:"
```

```
& vbCrLf & _
```

```
"比例类型: acZoomScaledRelative" & vbCrLf
```

```
& _
```

```
"比例因子: 2", "ZoomScaled"
```

```
Dim scalefactor As Double
```

```
Dim scaletype As Integer
```

```
scalefactor = 2
```

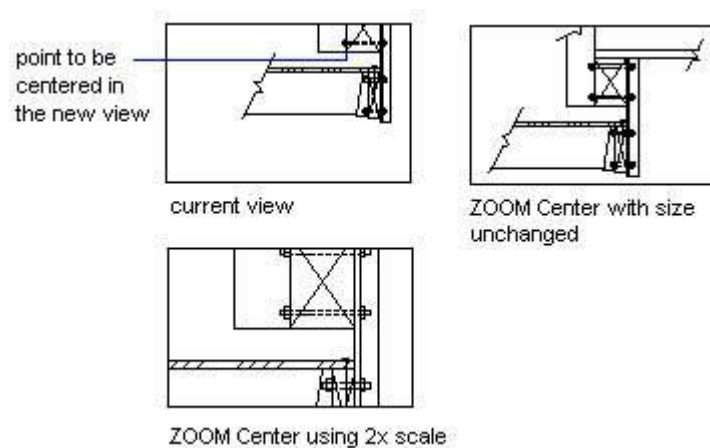
```
scaletype = acZoomScaledRelative
```

`ThisDrawing.Application.ZoomScaled scalefactor, scaletype`

End Sub

中心缩放

你可移动图形中的指定点到图形区域的中心, **ZoomCenter** 方法可有效地用于调整对象大小并将对象移动到视口的中心。以下例子演示了使用 **ZoomCenter** 以同样的大小及两倍大小显示视图的效果。



对于 **ZoomCenter**, 你可通过输入相对于当前视图的倍率来指定比例大小。

缩放活动图形到指定的中心

Sub Ch3_ZoomCenter()

MsgBox "使用以下参数执行 ZoomCenter:"

& vbCrLf & _

"圆心 3, 3, 0" & vbCrLf & _

"缩放倍率: 10", , "ZoomCenter"

```
Dim Center(0 To 2) As Double
```

```
Dim magnification As Double
```

```
Center(0) = 3: Center(1) = 3: Center(2) = 0
```

```
magnification = 10
```

```
ThisDrawing.Application.ZoomCenter Center, magnification
```

```
End Sub
```

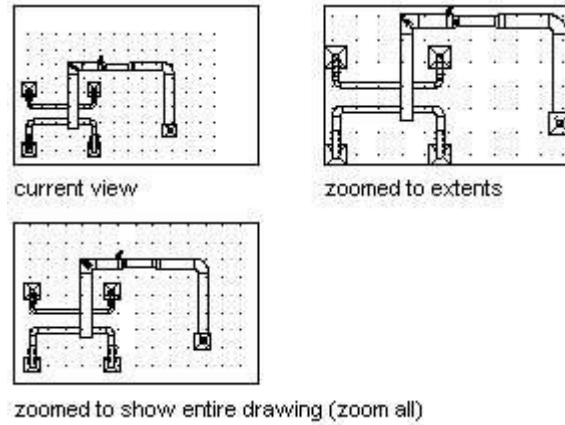
显示图形界限和范围

基于图形的边界或图形中对象的范围显示视图，使用的是 **ZoomAll**、**ZoomExtents** 或 **ZoomPrevious** 方法。

ZoomAll 显示整个图形，如果对象范围超出了界限，**ZoomAll** 显示对象的范围，如果对象都在界限内，**ZoomAll** 显示的为界限。

ZoomExtents 是基于活动视口的范围计算缩放，而不是当前视图。通常活动视口为完全可见，所以结果是明显而直接的。然而，当工作于图纸空间视口期间的模型空间中使用 **Zoom** 方法，如果缩放超出了图纸空间的边界时，缩放后的有些区域可能不可见。

ZoomExtents 更改视图到当前图形所包围的整个范围。有些情况下(对于 **ZoomAll** 和 **ZoomExtents**)，图形可能会重新生成。图形重新生成对于冻结的或关闭的图层不起作用。如果图形中没有对象，**ZoomExtents** 将显示图形的界限。



在三维视图中，ZoomAll 和 ZoomExtents 效果相同。无限构造线(即构造直线)和射线对该两项也不造成影响。

ZoomPrevious 缩放当前视口到其先前的范围。

缩放活动的图形到全图和图形范围

```
Sub Ch3_ZoomAll()
```

```
' ZoomAll
```

```
MsgBox "执行 ZoomAll",
```

```
, "ZoomAll"
```

```
ThisDrawing.Application.ZoomAll
```

```
' ZoomExtents
```

```
MsgBox "执行 ZoomExtents",
```

```
, "ZoomExtents"
```

```
ThisDrawing.Application.ZoomExtents
```

```
End Sub
```

使用命名视图

你可命名并保存视图留待以后使用。当你再不需要该视图时，你也可将其删除。

本节内容：

创建和命名视图

删除视图

创建和命名视图

视图是在建立时命名的。要创建新的视图，使用 **Add** 方法以添加一新的视图到 **Views**(视图)集合。

视图的名称最长可以 255 个字符，可包含字母、数字，还有些特殊字符如美元标志(\$)、连字号(-)和下划线(_)。

当你保存图形时，视图位置和视图的比例也将同时保存下来。

添加新的视图对象

```
Sub Ch3_AddView()
```

```
' 添加一命名视图到视图集合中
```

```
Dim viewObj As AcadView
```

```
Set viewObj = ThisDrawing.Views.Add("View1")
```

```
End Sub
```

删除视图

要删除命名的视图，使用 **Delete**(删除)方法。**View** 对象的 **Delete** 方法在 **View** 对象上，而不是在其上层对象上。以下例子从指定的 **View** 对象(**viewObj**)中和通过指定视图的名称(**View1**)从 **Views** 集合中删除视图。

从 **View** 对象中删除视图

`viewObj.Delete`

从 **Views** 集合中删除视图

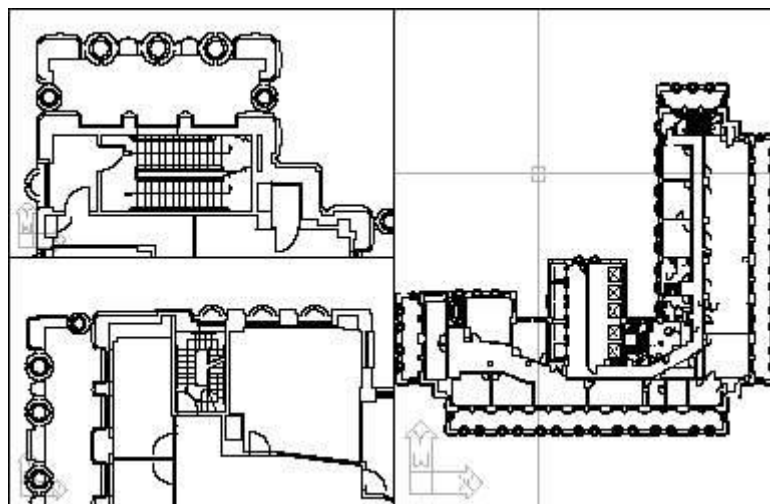
`ThisDrawing.Views("View1").Delete`

使用平铺视口

AutoCAD 通常是使用一填满整个图形区域的单一视口开始新的图形。你可拆分图形区域并同时显示多个视口。例如，如果你使一个全图和一个详细视图都可见，你就可以在整个图形中看到你在详细视图中更改的效果。在每一个视口中，你可做到以下内容：

- 缩放、设定捕捉、栅格和 **UCS** 图标模式、恢复在单独视口中的命名视图
- 执行命令期间可从一个视口进入另一个视口中
- 命名视口的配置以便重新使用它

下例显示了带三个平铺视口的图形。十字光标所在的视口为当前视口。视口完全填充了图形区域而且不会重叠。



在你制图时，在一个视口中所作的更改将会立即反映到其它的视口。

平铺视口与在图纸空间中排列的视口有所不同。图纸空间视口，也称为浮动视口，是用于建立图形的最后布局。它们可以重叠，并可同时进行出图打印。

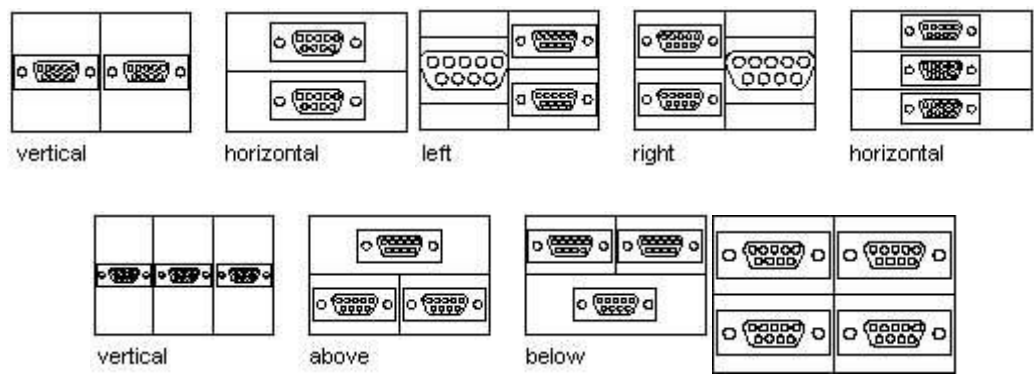
本节内容：

显示多平铺视口

使其它平铺视口成为当前视口

显示多平铺视口

你可用多种配置显示视口。你所显示的视口是依赖于你需要看到的视图数量和大小。以下例图显示了默认的视口配置：



要拆分活动的视口，使用 **Split** 方法。该方法有一个参数，就是拆分视口的配置类型。要指定窗口的配置，使用与上面显示的默认配置相符的常量：`acViewport2Horizontal`、`acViewport2Vertical`、`acViewport3Left`、`acViewport3Right`、`acViewport3Horizontal`、`acViewport3Vertical`、`acViewport3Above`、`acViewport3Below` 或 `acViewport4`。

以下例子创建新的视口，然后将视口拆分成两个水平窗口。

将视口拆分成两个水平窗口

Sub `SplitViewport()`

' 创建新的视口


```
Dim vportObj As AcadViewport
```

```
Set vportObj = ThisDrawing.Viewports.Add("TEST_VIEWPORT")
```

```
' 拆分 vportObj 为 2 个水平窗口
```

```
vportObj.Split acViewport2Horizontal
```

```
' 现在设定 vportObj 为活动视口
```

```
ThisDrawing.ActiveViewport = vportObj
```

```
End Sub
```

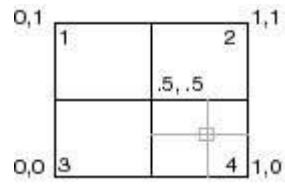
使其它平铺视口成为当前视口

你可在当前视口中输入点或选择对象。当视口为当前状态，箭头光标将更改为十字光标，同时边框为高亮显示。

要让视口成为当前视口，使用 **ActiveViewport** 属性。

你可在现存的视口中循环以查找特定的视口。要做到这一点，首先应使用 **Name** 属性确定你想得到视口的视口配置名称。另外，如果视口配置被拆分，每一在配置上的单独视口可通过 **LowerLeftCorner** 和 **UpperRightCorner** 属性来确定。

LowerLeftCorner 和 **UpperRightCorner** 属性描述了显示视口的图形布置。这些视口定义如下(使用四等分作为例子)：



Viewport 1-LowerLeftCorner

= (0, .5), UpperRightCorner = (.5, 1)

Viewport 2-LowerLeftCorner

= (.5, .5), UpperRightCorner = (1, 1)

Viewport 3-LowerLeftCorner

= (0, 0), UpperRightCorner = (.5, .5)

Viewport 4-LowerLeftCorner

= (.5, 0), UpperRightCorner = (1, .5)

拆分视口，然后在窗口中循环

本例将视口拆分为四个窗口。它可在图形中的所有视口之间循环并显示视口的名称及每个视口的左下角和右上角。

```
Sub Ch3_IteratingViewportWindows()
```

```
' 创建新的视口并使活动
```

```
Dim vportObj As AcadViewport
```

```
Set vportObj = ThisDrawing.Viewports.Add("TEST_VIEWPORT")
```

```
ThisDrawing.ActiveViewport = vportObj
```

' 拆分窗口为 4 个窗口

```
vportObj.Split acViewport4
```

' 在视口之间循环，高亮每个视口并

' 显示每个视口的右上角和左下角

```
Dim vport As AcadViewport
```

```
Dim LLCorner As Variant
```

```
Dim URCorner As Variant
```

```
For Each vport In ThisDrawing.Viewports
```

```
ThisDrawing.ActiveViewport
```

```
= vport
```

```
LLCorner = vport.LowerLeftCorner
```

```
URCorner = vport.UpperRightCorner
```

```
MsgBox "视口: " & vport.Name & "
```

```
现为活动状态。" & _
```

```
vbCrLf & "左下角: " & _
```

```
LLCorner(0) & ", " & LLCorner(1) &  
vbCrLf & _
```

```
"右上角: " & _
```

```
URCorner(0) & ", " & URCorner(1)
```

```
Next vport
```

```
End Sub
```

更新文档窗口的几何图形

你在 **AutoCAD ActiveX** 自动操作中执行的许多动作大多都是修改了 **AutoCAD** 图形的显示。不是所有的动作都会更新图形的显示。这样设计可以让你在图形更改过程中不必每个动作都去等待显示的更新。取而代之的是，你可将你的动作捆扎在一起然后在你完成更改后调用一个命令更新显示。

更新显示的方法为 **Update** 和 **Regen**。

Update 方法只更新单个对象的显示。而 **Regen** 方法重新生成整个图形并重新计算所有对象的屏幕坐标和视图分辨率。它也重新索引图形数据库以优化图形的显示和对象选择的性能。

更新单个对象的显示

本例创建圆，然后将圆的颜色改为红色。接着使用 **Update** 方法更新圆以使圆在 **AutoCAD** 中可见。

```
Sub Ch3_UpdateDisplay()
```

```
Dim circleObj
```

```
As AcadCircle
```

```
Dim center(0 To 2) As Double
```

```
Dim radius As Double
```

```
center(0) = 1: center(1) = 1: center(2) = 0
```

```
radius = 1
```

```
' 创建圆然后将其颜色改为红色
```

```
Set circleObj = ThisDrawing.ModelSpace.AddCircle(center,  
radius)
```

```
circleObj.Color = acRed
```

```
' 更新圆
```

```
circleObj.Update
```

```
End Sub
```

重置活动对象

更改大多数活动对象，如活动的图层和活动的线型，将立即显示出来。然而，还有几个活动的对象必须重置以使更改显示出来。这些对象为活动的字型、活动的 UCS 和活动的视口。如果这些对象的任何一个做过更改，该对象必须重置，并且必须调用 **Regen** 方法以使更改显示出来。

要重置这些对象，只要使用更新了的对象，设定其 **ActiveTextStyle**、**ActiveUCS** 或 **ActiveViewport** 属性。

重新设定活动视口

以下例子在活动视口中更改栅格显示，然后重置视口为活动视口以显示更改。

```
Sub Ch3_ResetActiveViewport()  
  
    ' 切换活动视口的栅格设定  
  
    ThisDrawing.ActiveViewport.GridOn = _  
  
    Not (ThisDrawing.ActiveViewport.GridOn)  
  
    ' 重新设定活动视口  
  
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport  
  
End Sub
```

设定和返回系统变量

Document 对象提供 **SetVariable** 和 **GetVariable** 方法以设定和返回 **AutoCAD** 系统变量。

设定 **MAXSORT** 系统变量

本例使用 **SetVariable** 方法分配整数给 **MAXSORT** 系统变量，以设定最大数目的符号名称或通过命令列表排序的图块名称。

```
ThisDrawing.SetVariable  
  
"MAXSORT", 100
```

精确制图

用 **AutoCAD** 你可在没有乏味计算的基础上创建具有精确几何体的图形。通常你可在不懂得坐标的情况下指定精确的点。在不离开图形屏幕的状态下，你可在图形上执行并显示多种类型的状态信息。

在此，AutoCAD ActiveX 自动操作不能提供以下 AutoCAD 性能的相应方法：

- 设定等轴捕捉和栅格
- 设定对象捕捉
- 指定对象的等距测量或分段等分对象

本节内容：

调整捕捉和栅格对齐

使用正交模式

绘制构造线

计算点和值

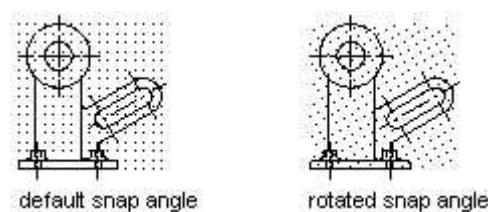
计算面积

调整捕捉和栅格对齐

你可将栅格作为可见的辅助线使用，也可打开捕捉模式以约束光标运动。通过设定间隔，你可调整捕捉和栅格对齐。你可旋转该对齐方式，或你可将其设定为用于等轴图形。

更改捕捉角度和基点

如果你需要通过指定的对齐方式和角度制图，你可旋转捕捉角度。当捕捉或正交模式打开时，该旋转角度重新对齐十字光标并强迫光标到新的对齐方式上。在以下例子中，捕捉角度调整到与锚支架匹配的角度。通过这样的调整，你可使用栅格在 30 度角度状态下绘制图形。



捕捉角度旋转的中心点为捕捉的基点。如果你需要对齐阴影图案，你可更改通常被设为 0,0 的基点。

要旋转捕捉角度，使用 `SnapRotationAngle` 属性。要更改捕捉角度旋转的基点，使用 `SnapBasePoint` 属性。

注意：这两个属性都需要调用 `Update` 方法以更新 AutoCAD 的显示。

更改捕捉基点和旋转角度

本例更改捕捉基点为**(1,1)**，更改捕捉旋转角度为 **30 度**。栅格打开以使所做的更改可见。

```
Sub Ch3_ChangeSnapBasePoint()
```

```
' 打开活动视口的栅格
```

```
ThisDrawing.ActiveViewport.GridOn = True
```

```
' 更改捕捉基点为 1, 1
```

```
Dim newBasePoint(0 To 1) As Double
```

```
newBasePoint(0) = 1: newBasePoint(1) = 1
```

```
ThisDrawing.ActiveViewport.SnapBasePoint = newBasePoint
```

```
' 更改捕捉旋转角度为 30 度(0.575 弧度)
```

```
Dim rotationAngle As Double
```

```
rotationAngle = 0.575
```

```
ThisDrawing.ActiveViewport.SnapRotationAngle = rotationAngle
```


' 重置视口

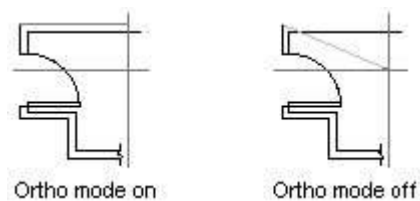
```
ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport
```

End Sub

使用正交模式

你在绘制线或移动对象时，可使用正交模式以约束光标在水平或垂直的轴上(正交对齐是依赖于当前捕捉角度或 UCS 坐标)。正交模式是在你需要指定第二个点时被激活的。使用正交模式不仅可建立垂直或水平方向对齐，而且强迫平行或创建规则的偏移。

通过允许 **AutoCAD** 利用正交抑制可提高制图的速度。例如，你可在开始制图时打开正交模式以创建一系列的平行线。因为所有的线都被强迫在水平或垂直的轴上，你可更快地制图，因为你知道这些线是平行的。



当你移动光标时，定义位移的一个橡皮筋线将跟随在水平或垂直轴上，这要看光标离哪个轴更接近。在透视图图中、或你在命令行输入坐标或指定对象捕捉时，**AutoCAD** 会忽略正交模式。

要打开或关闭正交模式，使用 **OrthoOn** 属性，该属性需要输入逻辑值。设定 **TURE** 时打开正交模式，**FALS E** 时关闭正交模式。

打开活动视口的正交模式

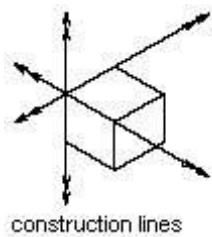
```
ThisDrawing.ActiveViewport.OrthoOn
```

```
= True
```

绘制构造线

你可创建单向或双向无限延伸的构造线。单向延伸的构造线称为射线。而双向延伸的构造线称为构造直线。这些构造线可用于创建其它对象的参考。例如，你可使用构造线以查找三角形的中心、准备多视图中的同一构件，或创建用于对象捕捉的临时交点。

构造线不会更改图形的最大区域，也就是说，这些无限的尺寸不会影响缩放或视口。你可移动、旋转和复制构造线，就象你移动、旋转和复制其它对象一样。你可将构造线创建于构造线图层并在打印前冻结或关闭。



本节内容：

创建构造直线

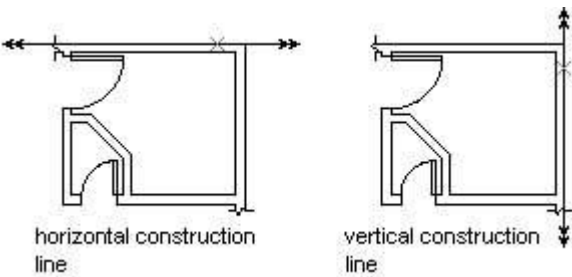
查询构造直线

创建射线

查询射线

创建构造直线

构造直线可置于三维空间的任何位置并双向无限延伸。要创建构造直线，使用 **AddXLine** 方法。该方法通过两个点的方法指定线，你输入或选择两个点以定义方位。第一个点为基点，它做为构造线的中点。



添加构造直线

以下示例代码使用(5, 0, 0)和(1, 1, 0)两点创建构造直线。

```
Sub Ch3_AddXLine()
```

```
Dim xlineObj
```

```
As AcadXline
```

```
Dim basePoint(0 To 2) As Double
```

```
Dim directionVec(0 To 2) As Double
```

```
' 定义构造直线
```

```
basePoint(0) = 2#: basePoint(1) = 2#: basePoint(2) =  
0#
```

```
directionVec(0) = 1#: directionVec(1) = 1#: directionVec(2)  
= 0#
```

```
' 在模型空间创建构造直线
```

```
Set xlineObj = ThisDrawing.ModelSpace.AddXLine _
```

```
(basePoint, directionVec)
```

```
ThisDrawing.Application.ZoomAll
```

```
End Sub
```

查询构造直线

当构造直线创建后，你可使用 **BasePoint** 属性查询构造直线的第一个点。用于创建构造直线的第二个点没有保存在对象中。而是使用 **DirectionVector** 属性以获取构造直线的方位矢量。

查询构造直线

本例查找上节所创建的构造直线的基点和方位矢量。

```
Dim BPoint As
```

```
Variant
```

```
Dim Vector As Variant
```

```
BPoint = xlineObj.basePoint
```

```
Vector = xlineObj.DirectionVector
```

创建射线

射线是在三维空间中开始于指定点并无限延伸的线。射线不是和构造直线一样双向延伸，而只是向单向延伸。正因为这样，射线可帮助减少由于众多的构造直线而产生的混乱。

和构造直线一样，射线在显示图形范围的命令中被忽略。

查询射线

当射线创建后，可使用 **BasePoint** 属性查询射线的第一个点。用于创建射线的第二个点没有被对象保存。

然而可使用 **DirectionVector** 属性获取射线的方位矢量。

添加、查询和编辑射线对象

以下例程使用(5, 0, 0)和(1, 1, 0)两点创建射线对象。然后查询当前基点和方位矢量，并且在消息框中显示结果。随后更改方位矢量，接着又查询和显示基点和新的方位矢量。

```
Sub Ch3_EditRay()
```

```
Dim rayObj As
```

```
AcadRay
```

```
Dim basePoint(0 To 2) As Double
```

```
Dim secondPoint(0 To 2) As Double
```

```
' 定义射线
```

```
basePoint(0) = 3#: basePoint(1) = 3#: basePoint(2) =  
0#
```

```
secondPoint(0) = 4#: secondPoint(1) = 4#: secondPoint(2)  
= 0#
```

```
' 在模型空间创建射线对象
```

```
Set rayObj = ThisDrawing.ModelSpace.AddRay _
```

```
(basePoint, secondPoint)
```

```
ThisDrawing.Application.ZoomAll
```

```
' 查找射线的当前状态
```

```
MsgBox "射线的基点为: " & _
```

```
rayObj.basePoint(0) & ", " & _
```

```
rayObj.basePoint(1) & ", " & _
```

```
rayObj.basePoint(2) & vbCrLf & _
```

```
"射线的方位矢量为: " & _
```

```
rayObj.DirectionVector(0) & ", " &
```

```
_
```

```
rayObj.DirectionVector(1) & ", " &
```

```
_
```

```
rayObj.DirectionVector(2), "编辑射线"
```

```
' 更改射线的方位矢量
```

```
Dim newVector(0 To 2) As Double
```

```
newVector(0) = -1
```

```
newVector(1) = 1
```

```
newVector(2) = 0
```

```
rayObj.DirectionVector = newVector
```

```
ThisDrawing.Regen False
```

```
MsgBox "射线的基点为: " & _
```

```
rayObj.basePoint(0) & ", " & _
```

```
rayObj.basePoint(1) & ", " & _
```

```
rayObj.basePoint(2) & vbCrLf & _
```

```
"射线的方位矢量为: " & _
```

```
rayObj.DirectionVector(0) & ", " &
```

```
_
```

```
rayObj.DirectionVector(1) & ", " &
```

```
_
```

```
rayObj.DirectionVector(2), , "编辑射线"
```

```
End Sub
```

计算点和值

通过使用 **Utility** 对象提供的方法，可快速解决数学问题或在图形中定位点。通过使用 **Utility** 对象中的方法，你可处理以下多方面问题：

- 用 **AngleFromXAxis** 方法查找直线相对于 X 轴的角度
- 用 **AngleToReal** 方法将字符串型的角度转换为实数(双精度)值

- 用 `AngleToString` 方法将角度从实数(双精度)值转换为字符串
- 用 `DistanceToReal` 方法将距离从字符串转换为实数(双精度)值
- 用 `CreateTypedArray` 方法创建包含整数、浮点数、双精度等的数组和变体
- 用 `PolarPoint` 方法查找从给定点在指定角度及距离上的点
- 用 `TranslateCoordinates` 方法将点从一个坐标系统转换到另一个坐标系统
- 用 `GetDistance` 方法查找用户输入的两点间的距离

用 `GetDistance` 方法查找两点间的距离

本例用 `GetDistance` 方法获取点坐标，并用 `MsgBox` 函数显示计算后的距离。

```
Sub Ch3_GetDistanceBetweenTwoPoints()
```

```
Dim returnDist
```

```
As Double
```

```
' 返回用户输入的值，提示用户输入。
```

```
returnDist = ThisDrawing.Utility.GetDistance _
```

```
(, "拾取两点: ")
```

```
MsgBox "两点间的距离为: " & returnDist
```

```
End Sub
```

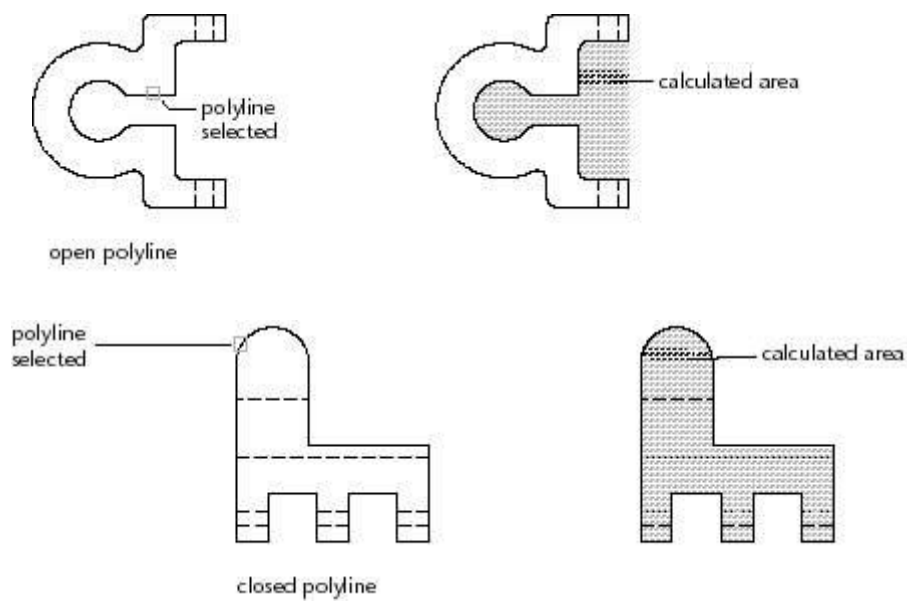
计算面积

你可通过用 `Area` 属性找到圆弧、圆、椭圆、细多段线、多段线、面域、平面闭合样条曲线的面积。

如果你需要计算多个对象的联合面积，你可用添加进行累计或在一系列面域上使用 **Boolean**(布尔)方法以获取描述想要得到的面积的单个面域。通过这个单个的面域你可用 **Area** 属性获取它的面积。

计算的面积依照查询对象类型的不同而不同。

- 闭合多段线和多边形：如果为宽多段线，面积定义为宽度的中心上
- 非闭合对象如非闭合的样条曲线和非闭合的多段线：将其看成有一直线连接起点和终点来计算面积
- 面域：面积与面域中对象的联合面积相等



本节内容：

计算定义的面积

计算定义的面积

你可通过用户指定的三维或三维点的定义测量任意闭合面域。所有点必须在同一平面上。

通过用户指定的点来获取面积

1. 循环使用 **GetPoint** 方法获取用户的点。
2. 通过用户所提供的点创建细多段线，可使用 **AddLightweightPolyline** 方法创建多段线。
3. 用 **Area** 属性获取新创建多段线的面积。
4. 用 **Erase** 方法删除多段线。

由用户输入的点计算面积

本例提示用户输入 5 个点，然后由这些点创建多段线。接着将多段线闭合，并且在消息框中显示多段线的面积。

```
Sub Ch3_CalculateDefinedArea()
```

```
Dim p1 As Variant
```

```
Dim p2 As Variant
```

```
Dim p3 As Variant
```

```
Dim p4 As Variant
```

```
Dim p5 As Variant
```

```
' 从用户处取得点
```

```
p1 = ThisDrawing.Utility.GetPoint(, vbCrLf & "第一个点:  
")
```

```
p2 = ThisDrawing.Utility.GetPoint(p1, vbCrLf & "第二个点:
```

)

p3 = ThisDrawing.Utility.GetPoint(p2, vbCrLf & "第三个点:

)

p4 = ThisDrawing.Utility.GetPoint(p3, vbCrLf & "第四个点:

)

p5 = ThisDrawing.Utility.GetPoint(p4, vbCrLf & "第五个点:

)

' 由这些点创建二维多段线

Dim polyObj As AcadLWPolyline

Dim vertices(0 To 9) As Double

vertices(0) = p1(0): vertices(1) = p1(1)

vertices(2) = p2(0): vertices(3) = p2(1)

vertices(4) = p3(0): vertices(5) = p3(1)

vertices(6) = p4(0): vertices(7) = p4(1)

vertices(8) = p5(0): vertices(9) = p5(1)

Set polyObj = ThisDrawing.ModelSpace.AddLightWeightPolyline

—

(vertices)

```
polyObj.Closed = True
```

```
ThisDrawing.Application.ZoomAll
```

' 显示多段线的面积

```
MsgBox "通过定义的点形成的面积为 " & _
```

```
polyObj.Area, , "计算定义的面积"
```

```
End Sub
```

提示用户输入

作为 **Document** 对象下层的 **Utility** 对象，定义了用户输入方法。用户输入方法在 **AutoCAD** 命令行中显示提示并要求输入各种类型的值。这种用户输入的形式对交互地输入屏幕坐标、选择图元和短字符或数字是非常有用的。如果你的应用程序需要输入很多的项目或值，使用对话框要比用单独的提示更适用。

每一用户输入方法都在 **AutoCAD** 命令行中显示提示并返回指定要求输入类型的值。例如，**GetString** 返回字符串，**GetPoint** 返回变体(包含双精度三元素数组)，而 **GetInteger** 返回整数值。你可用 **InitializeUserInput** 方法预先控制用户的输入。该方法让你控制如 **NULL** 输入(按回车)、输入 **0** 值或负值，还有输入阿拉伯文本值(字母)。

要强迫另起一行显示提示，可在提示字符串的起始处使用回车/新行常量(**vbCrLf**)。

本节内容：

GetString 方法

GetPoint 方法

GetKeyword 方法

控制用户输入

GetString 方法

GetString 方法在 AutoCAD 命令行中提示用户输入字符串。该方法接受两个参数。第一个参数控制在输入字符串中是否允许空格。如果设为 0，则不允许空格(空格为确定输入)；如果设为 1，字符串可包含空格(确定输入时必须按回车)。第二个参数为提示内容。

在 AutoCAD 命令行中获得用户输入的字符串值

以下示例显示输入你的名字的提示，并要求用户输入后按回车确定(在输入字符串中允许空格)。该字符串值保存在 retVal 变量中，并使用消息框显示出来。

```
Sub Ch3_GetStringFromUser()
```

```
Dim retVal As
```

```
String
```

```
retVal = ThisDrawing.Utility.GetString _
```

```
(1, vbCrLf & "输入你的名字: ")
```

```
MsgBox "输入的名字为: " & retVal
```

```
End Sub
```

GetString 方法不支持预先调用 InitializeUserInput 方法。

GetPoint 方法

GetPoint 方法在命令行中提示用户指定一个点。该方法接受两个参数，可选用的参考点和提示字符串。如果提供了参考点，**AutoCAD** 将从该点绘制一橡皮筋线。要控制用户输入，该方法可预先调用 **InitializeUserInput** 方法。

通过用户先把获取点

以下示例提示用户输入两个点，然后用这些点作为起点和终点绘制直线。

```
Sub Ch3_GetPointsFromUser()
```

```
Dim startPnt
```

```
As Variant
```

```
Dim endPnt As Variant
```

```
Dim prompt1 As String
```

```
Dim prompt2 As String
```

```
prompt1 = vbCrLf & "输入直线起点: "
```

```
prompt2 = vbCrLf & "输入直线终点: "
```

```
' 在没有提示参考点的情况下获取第一个点
```

```
startPnt = ThisDrawing.Utility.GetPoint(, prompt1)
```

```
' 利用上边起点作为参考点获取第二个点
```

```
endPnt = ThisDrawing.Utility.GetPoint(startPnt, prompt2)
```

```
' 使用输入的两个点创建直线
```

```
ThisDrawing.ModelSpace.AddLine startPnt, endPnt
```

```
ThisDrawing.Application.ZoomAll
```

```
End Sub
```

GetKeyword 方法

GetKeyword 方法在 **AutoCAD** 命令行提示用户输入关键字。该方法只接受一个参数，就是提示字符串。关键字和输入的参数在一个称为 **InitializeUserInput** 方法中定义。

在 **AutoCAD** 命令行中从用户获取关键字

以下示例通过设定 **InitializeUserInput** 的第一个数为 1 而不允许输入 **NULL**(即不允许按回车)，强迫用户输入一关键字。第二个参数确定有效关键字的列表。

```
Sub Ch3_KeyWord()
```

```
Dim keyWord As
```

```
String
```

```
ThisDrawing.Utility.InitializeUserInput 1, "Line
```

```
Circle Arc"
```

```
keyWord = ThisDrawing.Utility.GetKeyword _
```

```
(vbCrLf & "输入选项[直线(L)/圆(C)/圆弧(A)]: ")
```

```
MsgBox keyWord, , "GetKeyword 示例"
```

```
End Sub
```

更加友好的关键字提示就是在用户按回车时(即输入 NULL)提供默认值。注意二次修改的示例如下：

```
Sub Ch3_KeyWord2()
```

```
Dim keyWord As
```

```
String
```

```
ThisDrawing.Utility.InitializeUserInput 0, "Line
```

```
Circle Arc"
```

```
keyWord = ThisDrawing.Utility.GetKeyword _
```

```
(vbCrLf & "输入选项[直线(L)/圆(C)/圆弧(A)]: ")
```

```
If keyWord = "" Then keyWord = "Arc"
```

```
MsgBox keyWord, , "GetKeyword 示例"
```

```
End Sub
```

控制用户输入

你可使用 **InitializeUserInput** 方法定义关键字或约束输入到用户输入方法的类型。其使用和参数值与 AutoL

ISP 的 **initget** 函数大致相同。**InitializeUserInput** 可用于以下方法：**GetAngle**、**GetCorner**、**GetDistance**、

GetInteger、GetKeyword、GetOrientation、GetPoint 和 GetReal。InitializeUserInput 不能用于 GetString 方法。当用户输入方法没有返回字符串值时，使用 GetInput 方法以返回字符串值(关键字或字母输入)。

InitializeUserInput 方法接受两个参数，第一个参数为位编码整数值，它确定用户输入方法的输入选择。第二个参数为字符串，它定义了有效的关键字。

在 AutoCAD 命令行中从用户获取整数值或关键字

以下示例提示用户输入非零正整数或关键字：

```
Sub Ch3_UserInput()
```

```
' InitializeUserInput
```

```
(6)的第一个参数约束输入非零正整数。
```

```
' 第二个参数为有效的关键字列表。
```

```
ThisDrawing.Utility.InitializeUserInput 6, "Big
```

```
Small Regular"
```

```
' 设定提示字符串
```

```
Dim promptStr As String
```

```
promptStr = vbCrLf & "输入大小或[大(B)/小(S)/正常(R)]<正常>:"
```

```
' 在 GetInteger 提示下输入关键字将会出错；
```

```
' 在检查到有出错信息后还允许程序继续运行，
```

```
' 你必须设定出错处理以恢复该错误。
```

On Error Resume Next

' 获取用户输入值

Dim returnInteger As Integer

returnInteger = ThisDrawing.Utility.GetInteger(promptStr)

' 检查错误。如果错误描述与以下提示匹配，

' 则使用 **GetInput** 以取得返回的字符串

' 否则，使用 **returnInteger** 值。

If Err.Description = "User input is a keyword"

Then

Dim returnString

As String

returnString = ThisDrawing.Utility.GetInput()

Err.Clear

Else

If returnInteger

= 0 Then '如果用户按了回车

```

returnString

= "Regular" '设定默认值

Else '否则,

returnString = returnInteger '使用输入的值

End If

End If

' Display the result

MsgBox returnString, , "InitializeUserInput 示例"

End Sub

```

访问 **AutoCAD** 命令行

你可使用 **SendCommand** 方法直接发送命令到 **AutoCAD** 命令行。**SendCommand** 方法发送单一的字符串到命令行。字符串必须包含所执行命令次序相同的命令列表变量。字符串中的空格或相当于回车的 **ASCII** 码与在键盘中按回车是相同的。与 **AutoLISP** 环境不同的是，没有变量的情况下调用 **SendCommand** 方法是无效的。

发送命令到 **AutoCAD** 命令行

以下示例用圆心(2, 2, 0)和半径为 4 创建圆。然后缩放显示图形的所有几何体。注意到字符串的最后有一空格，它描述了最后的回车以开始执行命令。

```

Sub Ch3_SendACommandToAutoCAD()

ThisDrawing.SendCommand "_Circle

2,2,0 4 "

```

```
ThisDrawing.SendCommand "_zoom a "
```

```
End Sub
```

工作于无打开文档状态

AutoCAD 通常在启动时新建或打开现在文档。然而，有可能在当前进程中关闭所有文档。

如果你在 AutoCAD 用户界面中关闭所有文档，你将会注意到应用程序窗口起了些变化。菜单减小到简单的文件、视图、窗口和帮助菜单，而且这些菜单也少了一些项。你也会注意到命令行不见了。

同样，在无文档打开时 **ActiveX** 界面只允许以下动作：

- 你可打开文档
- 你可创建新的文档
- 你可输入文档
- 你可退出 AutoCAD

这些动作在 **Documents** 集合中全都有效。这些 **Documents** 集合中的方法和属性，再加上 **Application** 对象的方法和属性的有限集，只能在没有打开文档时的界面下有效。如果你执行其它的一些试图访问用户选项，动作将造成错误。

使用 **Documents** 集合的 **Count** 属性确定 AutoCAD 是否为零文档状态。如果 **Documents.Count** = 0，则 AutoCAD 为零文档状态，而 **Documents.Count** > 0，则至少有一图形打开。

一样值得注意的是当 AutoCAD 为零文档状态时，VBA 的 **ThisDrawing** 对象是没有定义的。提醒一下，**ThisDrawing** 通常指的是活动的图形，而在零文档状态时是没有打开的图形。试图执行使用了 **ThisDrawing** 的宏将造成运行时错误。要解决该错误，可在没有文档打开时使用 VBA 的 **GetObject** 函数以获得与 AutoCAD 的连接。

输入其它文件格式

你可通过用指定格式打开其它应用程序的图形或图像来使用它们。AutoCAD 处理 DXF、SAT、BMP 和 Post Script 文件的一些转换格式。对于所有版本，你可通过使用 Import 方法输入文件，该方法需要输入三个值：输入文件的名称、文件置于图形中的插入点和输入图形所使用的比例因子。

输出到其它文件格式

如果你需要在其它应用程序中使用 AutoCAD 图形，可通过使用 Export 方法将其转换为指定格式。该方法可输出 AutoCAD 图形到 WMF、SAT、EPS、DXF、DWF 或 BMP 格式。Export 方法需要输入三个值：所要创建的新文件名称、新文件的扩展名和输入对象的选择集(需要时提供)。

当输出为 WMF、SAT 和 BMP 格式，必须提供非空选择集。该选择集指定图形中输出的对象，如果没有指定选择集，将不作输出并捕获到无效变量出错结果。

当输出到 EPS 和 DXF 格式，不必指定选择集。整个图形将自动输出到相应格式。

输出图形到 DXF 文件格式，然后再将其输入

本例在当前图形中创建圆，然后输出该图形到名为 DXFExprt.DXF 的文件。再接着打开新的图形并输入该文件。

```
Sub Ch3_ImportingAndExporting()
```

```
' 创建圆以便整个过程可见
```

```
Dim circleObj As AcadCircle
```

```
Dim centerPt(0 To 2) As Double
```

```
Dim radius As Double
```

```
centerPt(0) = 2: centerPt(1) = 2: centerPt(2) = 0
```

```
radius = 1
```

```
Set circleObj = ThisDrawing.ModelSpace.AddCircle _
```

```
(centerPt, radius)
```

```
ThisDrawing.Application.ZoomAll
```

```
' 创建一空的选择集
```

```
Dim sset As AcadSelectionSet
```

```
Set sset = ThisDrawing.SelectionSets.Add("NEWSSET")
```

```
' 输出当前图形到 DXF 文件
```

```
Dim exportFile As String
```

```
exportFile = "C:"
```

```
ThisDrawing.Export exportFile, "DXF", sset
```

```
' 打开新的图形
```

```
ThisDrawing.Application.Documents.Add "acad.dwt"
```

```
' 定义输入
```

```
Dim importFile As String
```

```
Dim insertPoint(0 To 2) As Double
```

```
Dim scalefactor As Double
```

```
importFile = "C:.dxf"
```

```
insertPoint(0) = 0: insertPoint(1) = 0: insertPoint(2)  
= 0
```

```
scalefactor = 2#
```

```
' 输入文件
```

```
ThisDrawing.Import importFile, insertPoint, scalefactor
```

```
ThisDrawing.Application.ZoomAll
```

```
End Sub
```

第四章 创建和编辑 **AutoCAD** 图元

你可创建各类对象，包括从简单的线和圆到样条曲线、椭圆和关联阴影区域等。一般情况下，使用 **Add** 方法以添加对象到模型空间。也可在图纸空间或图块中创建对象。

对象创建后，可更改对象的图层、颜色和线型。也可添加文本以注释图形。

创建对象

虽然在 AutoCAD 中有许多种不同的方法可创建相同的图形对象，但在 ActiveX 自动操作中对每一对象只提供一种创建方法。例如，在 AutoCAD 中有四种不同的方法可创建圆：(1)通过指定圆心和半径；(2)通过定义直径的两个点；(3)通过定义圆周的三个点；或(4)通过两个切点和一个半径。然而，在 ActiveX 自动操作中只提供一种创建方法来创建圆，该方法就是使用圆心和半径。

注意：不管使用 CreateObject 或用 New 关键字 Dim 而创建对象的 VB 和 VBA 方法只用于创建 AutoCAD 应用程序对象。所有其它 AutoCAD 对象必须使用 AutoCAD 界面提供的 Add 或 Add<对象名称>方法来创建。

确定容器对象

图形对象是创建于 ModelSpace 集合、PaperSpace 集合或 Block 对象中。

ModelSpace 集合是通过 ModelSpace 属性返回，PaperSpace 集合是通过 PaperSpace 属性返回。

你可直接引用这些对象，或通过用户定义变量引用。直接引用对象，必须包含调用层次的对象。例如，以下语句添加直线到模型空间中：

```
Set lineObj = ThisDrawing.ModelSpace.AddLine(startPoint,endPoint)
```

通过用户定义变量引用对象，必须将变量定义为 AcadModelSpace 或 AcadPaperSpace 类型，然后设定变量活动文档的适当属性。以下例子定义两量并设定其与当前模型空间和图纸空间相等，分别为：

```
Dim moSpace As AcadModelSpace
```

```
Dim paSpace As AcadPaperSpace
```

```
Set moSpace = ThisDrawing.ModelSpace
```

```
Set paSpace = ThisDrawing.PaperSpace
```

以下语句使用用户定义变量添加直线到模型空间：

```
Set lineObj = moSpace.AddLine(startPoint,endPoint)
```


创建直线

直线 AutoCAD 中最基本的对象。你创建多种不同的线-单一的直线和带有圆弧或不带圆弧的复线。一般情况下，你通过指定坐标点来绘制线。默认的线型为 **CONTINUOUS**，一种不间断的直线，还有许多使用点划线的线型可供选用。

创建直线，可使用以下方法中的一种：

AddLine

通过两点创建线。

AddLightweightPolyline

通过顶点的列表创建二维细多段线。

AddMLine

创建复线。

AddPolyline

创建二维或三维多段线。

标准直线和复线创建于 **WCS** 的 **XY** 平面上。多段线和细多段线是创建于对象坐标系统(**OCS**)。关于转换 **OC**
S 坐标的更多信息，请查看"转换坐标"。

创建多段线对象

本例使用 **AddLightweightPolyline** 方法通过二维坐标(2,4)、(4,2)和(6,4)创建简单的两段多段线。

```
Sub Ch4_AddLightWeightPolyline()
```

```
Dim plineObj As AcadLWPolyline
```

```
Dim points(0 To 5) As Double
```

```
' 为二维多段线顶点赋值
```

```
points(0) = 2: points(1) = 4
```

```
points(2) = 4: points(3) = 2
```

```
points(4) = 6: points(5) = 4
```

```
' 在模型空间中创建一细多段线对象
```

```
Set plineObj = ThisDrawing.ModelSpace. _
```

```
AddLightWeightPolyline(points)
```

```
ThisDrawing.Application.ZoomAll
```

```
End Sub
```

创建曲线对象

你可在 AutoCAD 中创建多种曲线对象，包括样条曲线、圆、圆弧和椭圆。所有曲线创建于当前 WCS 的 XY 平面上。

创建曲线，使用的是以下方法：

AddArc

给定圆心、半径、起始角度和终止角度创建圆弧。

AddCircle

给定圆心和半径创建圆。

AddEllipse

给定中心点、主轴上的点和半径比率创建椭圆。

AddSpline

创建二次或三次 NURBS(曲线曲面的非均匀有理 B 样条)曲线。

创建 Spline(样条曲线)对象

本例使用 (0, 0, 0)、(5, 5, 0)和(10, 0, 0)三个点在模型空间中创建样条曲线。该样条曲线的起始和终止切向为(0.5, 0.5, 0.0)。

Sub Ch4_CreateSpline()

' 本例在模型空间中创建样条曲线对象。

' 声明所需的变量

Dim splineObj As AcadSpline

Dim noOfPoints As Integer

Dim startTan(0 To 2) As Double

Dim endTan(0 To 2) As Double

```
Dim fitPoints(0 To 8) As Double
```

```
' 为变量赋值
```

```
noOfPoints = 3
```

```
startTan(0) = 0.5: startTan(1) = 0.5: startTan(2)  
= 0
```

```
endTan(0) = 0.5: endTan(1) = 0.5: endTan(2) = 0
```

```
fitPoints(0) = 1: fitPoints(1) = 1: fitPoints(2)  
= 0
```

```
fitPoints(3) = 5: fitPoints(4) = 5: fitPoints(5)  
= 0
```

```
fitPoints(6) = 10: fitPoints(7) = 0: fitPoints(8)  
= 0
```

```
' 创建样条曲线
```

```
Set splineObj = ThisDrawing.ModelSpace.AddSpline
```

```
—
```

```
(fitPoints, startTan, endTan)
```

ZoomAll

End Sub

关于样条曲线的更多信息，请参考 [AutoCAD ActiveX](#) 和 [VBA](#)

参考中的样条曲线和 [AddSpline](#) 方法文档。

创建点对象

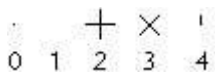
Point 对象非常有用，例如，将其作为节点或参考点可捕捉到该或从该点处偏移对象。你可设定点的样式和相对于屏幕或绝对单位的大小。

本节内容：

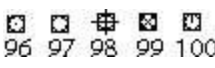
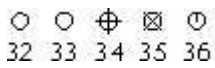
控制点的外观

控制点的外观

PDMODE 和 **PDSIZE** 系统变量控制着 **Point** 对象的外观。**PDMODE** 的值 0、2、3 和 4 指定绘制点的形状。值 1 为无显示。



在先前值加上 32、64 或 96 则在点的周围另外增加其它的形状：



当 **PDMODE** 值为 **0** 和 **1** 除外，**PDSIZE** 控制点形状的大小。**PDSIZE** 设定为 **0** 则按 **5%**的屏幕高度生成点。
PDSIZE 为正值时指定点形状的绝对大小。负值时被理解为视口大小的百分比。当图形重新生成时所有点的大小将重新计算。

在你更改了 **PDMODE** 和 **PDSIZE** 后，现存点的外观将会在下次图形重新生成时更改过来。

设定 **PDMODE** 和 **PDSIZE**，使用 **SetVariable** 方法。

创建 **Point**(点)对象并更改其外观

以下例程按坐标(5, 5, 0)在模型空间创建 **Point** 对象。

```
Sub Ch4_CreatePoint()
```

```
Dim pointObj As AcadPoint
```

```
Dim location(0 To 2) As Double
```

```
' 定义点的位置
```

```
location(0) = 5#: location(1) = 5#: location(2) =
```

```
0#
```

```
' 创建点
```

```
Set pointObj = ThisDrawing.ModelSpace.AddPoint(location)
```

```
ThisDrawing.SetVariable "PDMODE", 34
```

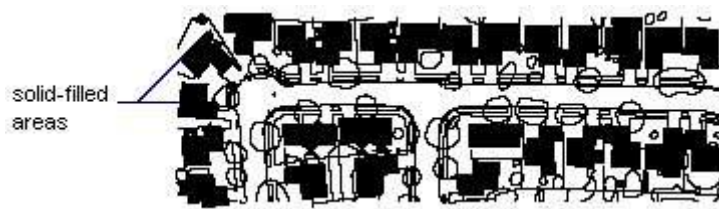
```
ThisDrawing.SetVariable "PDSIZE", 1
```

```
ZoomAll
```

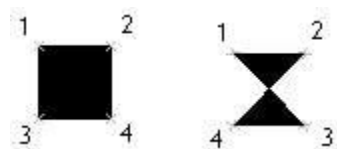
```
End Sub
```

创建实体填充区域

你可创建填满颜色的三角形和四边形。为了快速产生结果，可以 **FILLMODE** 系统变量为关闭时创建这些区域，然后将 **FILLMODE** 打开以填充完成的区域。



当你创建四边形实体填充区域，第三点和第四点的次序决定它的形状。请比较以下的图例：



前两个点定义了多边形的一边。第三个点定义了相对于第二点对角点。如果第四个点与第三个点相同，则创建填充的三角形。

要创建实体填充区域，使用 **AddSolid** 方法。

创建实体填充对象

以下例程使用坐标(0, 0, 0)、(5, 0, 0)、(5, 8, 0)和(0, 8, 0)在模型空间中创建四边形实体。

```
Sub Ch4_CreateSolid()
```

```
Dim solidObj As AcadSolid
```

```
Dim point1(0 To 2) As Double
```

```
Dim point2(0 To 2) As Double
```

```
Dim point3(0 To 2) As Double
```

```
Dim point4(0 To 2) As Double
```

```
' 为实体赋值
```

```
point1(0) = 0#: point1(1) = 0#: point1(2) = 0#
```

```
point2(0) = 5#: point2(1) = 0#: point2(2) = 0#
```

```
point3(0) = 5#: point3(1) = 8#: point3(2) = 0#
```

```
point4(0) = 0#: point4(1) = 8#: point4(2) = 0#
```

```
' 在模型空间中创建实体对象
```

```
Set solidObj = ThisDrawing.ModelSpace.AddSolid _
```

```
(point1, point2, point3, point4)
```


ZoomAll

End Sub

创建面域

面域是由称为回路的闭合形状所创建的附于二维空间的区域。回路是曲线或顺序连接曲线，它定义了非交叉边界在平面上的区域。回路可由直线、细多段线、圆、圆弧、椭圆、椭圆弧、样条曲线、三维面、轨迹和实体结合而成。生成回路和对象必须闭合或与其它对象共享终点的闭合区域生成。它们同时也必须共面(在同一平面上)。

如果开放的曲线内部有交点，它们是不能形成面域。

如三维多段线和网面之类的对象通过分解可转换为面域。你不能从具有交点的开放对象形成的闭合区域来生成面域：例如，相交的圆弧或自我相交的曲线。

生成面域的回路必须定义为对象的数组。

你可对面域应用阴影和着色，你也可分析其面积和惯性力矩等属性。你可创建形状，然后选择对象以创建面域。

创建面域，使用的是 **AddRegion** 方法。该方法将通过由输入的曲线数组构成的每一闭合回路创建面域。

AutoCAD 转换闭合二维和平面三维多段线为单独的面域，然后转换构成闭合平面回路的多段线、直线和曲线。

如果有两个以上的曲线共享一个终点，结果面域可能不可预知。由于这样，当使用 **AddRegion** 方法时尽量创建单独的面域。使用变体以保存最新创建面域的数组。

计算所创建面域对象的总数，使用 Visual Basic 的 **UBound** 和 **LBound** 函数

。例如，以下语句：

UBound(objRegions) - LBound(objRegions)

+ 1

这里 **objRegions** 是包含从 **AddRegion** 返回的变体。该语句将计算所创建面域的总数。

创建简单的面域

以下代码示例通过单一的圆创建面域。

```
Sub Ch4_CreateRegion()
```

```
' 定义数组以保存面域的边界。
```

```
Dim curves(0 To 0) As AcadCircle
```

```
' 创建形成面域边界的圆。
```

```
Dim center(0 To 2) As Double
```

```
Dim radius As Double
```

```
center(0) = 2
```

```
center(1) = 2
```

```
center(2) = 0
```

```
radius = 5#
```

```
Set curves(0) = ThisDrawing.ModelSpace.AddCircle _
```

```
(center, radius)
```

```
' 创建面域
```

```
Dim regionObj As Variant
```

```
regionObj = ThisDrawing.ModelSpace.AddRegion( curves)
```

```
ZoomAll
```

```
End Sub
```

本节内容：

创建复合面域

创建复合面域

你可通过差集、并集或查找面域或三维实体的交集来创建复合面域。然后你可挤出或旋转复合面域以创建复杂的实体。创建复合面域，使用 **Boolean** 方法。

当你从另外的面域减去当前面域，是从第一个面域中调用 **Boolean** 方法。该面域就是你想被减去的面域。

例如，计算地面需要多少地毯，是从地面的外边界中调用 **Boolean** 方法，并用未铺地毯的区域(如柱子和柜台)作为 **Boolean** 参数列表中的对象。

创建复合面域

```
Sub Ch4_CreateCompositeRegions()
```

```
' 创建两个圆，一个代表房间，
```

' 另外一个为房间中心的柱子

```
Dim RoomObjects(0 To 1) As AcadCircle
```

```
Dim center(0 To 2) As Double
```

```
Dim radius As Double
```

```
center(0) = 4
```

```
center(1) = 4
```

```
center(2) = 0
```

```
radius = 2#
```

```
Set RoomObjects(0) = ThisDrawing.ModelSpace. _
```

```
AddCircle(center, radius)
```

```
radius = 1#
```

```
Set RoomObjects(1) = ThisDrawing.ModelSpace. _
```

```
AddCircle(center, radius)
```

' 用两个圆创建面域

```
Dim regions As Variant
```

```
regions = ThisDrawing.ModelSpace.AddRegion(RoomObjects)
```

```
' 将面域复制到面域变体中以方便使用
```

```
Dim RoundRoomObj As AcadRegion
```

```
Dim PillarObj As AcadRegion
```

```
If regions(0).Area > regions(1).Area Then
```

```
' 第一个面域为房间
```

```
Set RoundRoomObj = regions(0)
```

```
Set PillarObj = regions(1)
```

```
Else
```

```
' 第一个面域为柱子
```

```
Set PillarObj = regions(0)
```

```
Set RoundRoomObj = regions(1)
```

```
End If
```

' 将房间颜色设为红色，柱子颜色设为青色

```
RoundRoomObj.Color = acRed
```

```
PillarObj.Color = acCyan
```

```
ZoomAll
```

' 从地板空间中减去柱子的空间

' 以得到代表总的地毯面积的面域。

```
RoundRoomObj.Boolean acSubtraction, PillarObj
```

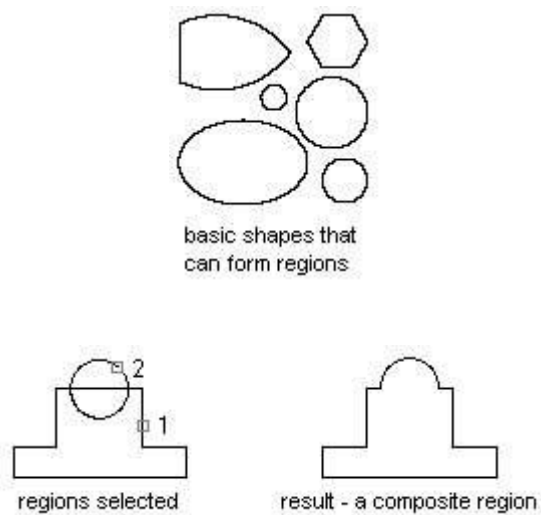
' 使用 **Area** 属性以确定总的地毯面积

```
MsgBox "地毯面积为: " & RoundRoomObj.Area
```

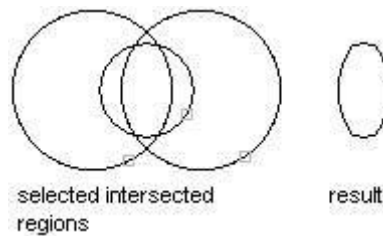
```
End Sub
```

用 **Area** 属性可找到结果面域的面积。

要结合面域，调用 **Boolean** 方法并输入常量 **acUnion** 以代替 **acSubtraction**。要查找两个面域的交集，使用常量 **acIntersection**。你可通过任何次序结合面域以取得其并集或交集。



以下例图显示三个面域的交集：



创建阴影

阴影为用图案填充图形中的指定区域。

当创建阴影时，最先操作不是指定填充的区域。首先你必须创建 **Hatch** 对象。做完这一步后，你可指定外部的回路，也就是阴影的最外边界。然后你可继续指定存在于阴影中的其它内容回路。

本节内容：

创建 **Hatch**(阴影)对象

关联阴影

分配阴影图案类型和名称

定义阴影边界

创建 Hatch(阴影)对象

当创建 Hatch 对象时，你要指定阴影图案类型、阴影图案名称和关联性。当 Hatch 对象创建后，你再也不能更改阴影的关联。

创建 Hatch 对象，使用 **AddHatch** 方法。

关联阴影

你可创建关联或非关联阴影。关联阴影是与阴影的边界相链接，当边界修改后阴影也会随之更新。非关联阴影是与它们的边界相独立。

关联性只能在阴影创建时设定。当阴影创建后，你可将其分解为非关联，但你不能再次将其关联进来。

生成关联阴影，可设定 **AddHatch** 方法中的 **Associativity** 参数为 **TRUE**。生成非关联阴影，可设定 **AddHatch** 方法中的 **Associativity** 参数为 **FALSE**。

分配阴影图案类型和名称

AutoCAD 提供实心填充和多于五十种的工业标准阴影图案。阴影图案可高亮图形中特殊的特征或区域。例如，图案可帮助区分三维对象的构成或代表制造对象的材料。

你可使用由 **AutoCAD** 提供的图案或外部图案库。关于 **AutoCAD** 提供的阴影图案的列表，请参见 **AutoCAD** 命令参考的附录 **E"标准库"**。

指定唯一的图案，你必须在创建阴影对象时输入图案类型和图案名称。图案类型指定在何处找到图案名称。当输入图案，使用以下某一个常量：

acHatchPatternTypePredefined

从定义于 **acad.pat** 文件中的图案选择图案名称。

acHatchPatternTypeUserDefined

使用当前线型定义线图案。

acHatchPatternTypeCustomDefined

从其它的 PAT 文件(除 acad.pat)选择图案名称。

当输入图案名称时，使用对图案类型指定的文件有效的名称。

定义阴影边界

当阴影对象创建后，可添加阴影边界。边界可由任何直线、圆弧、圆、二维多段线、椭圆、样条曲线和面域组成。




添加的第一个边界必须为外部的边界，它定义了阴影填充的最大外界限。添加外部边界，使用 AppendOuterLoop 方法。

当定义了外部边界后，你可继续添加内部边界。添加内部边界用 AppendInnerLoop 方法。

内容边界定义了阴影内的孤岛。Hatch 对象对于这些孤岛的处理方法依靠 HatchStyle 属性的设定值。

HatchStyle 属性可设定为以下情形：

阴影样式定义

阴影样式	条件	描述
	正常	指标准样式或正常。该选项由外向内填充阴影。如果 AutoCAD 遇到内部边界，它会关闭阴影直到再遇到另一个边界。该项为 HatchStyle 属性的默认设定。
	外部	只填充最外部的区域。该样式也是从区域边界向内填充阴影，但当它遇到内部边界时关闭阴影而不再打开。
	忽略	忽略内部结构。该选项填充所有内部对象。

当你完成定义阴影时，在显示前它必须求值。使用 **Evaluate** 方法以达到该要求。

创建阴影对象

本例在模型空间中创建关联阴影。当阴影创建后，更改关联了阴影的圆的大小。阴影将会自动更改以与当前圆的大小匹配。

```
Sub Ch4_CreateHatch()
```

```
Dim hatchObj As AcadHatch
```

```
Dim patternName As String
```

```
Dim PatternType As Long
```

```
Dim bAssociativity As Boolean
```

```
' 定义阴影
```

```
patternName = "ANSI31"
```

```
PatternType = 0
```

```
bAssociativity = True
```

```
' 创建关联阴影对象
```

```
Set hatchObj = ThisDrawing.ModelSpace.AddHatch _
```

(PatternType, patternName, bAssociativity)

' 为阴影创建外部边界(一个圆)。

Dim outerLoop(0 To 0) As AcadEntity

Dim center(0 To 2) As Double

Dim radius As Double

center(0) = 3: center(1) = 3: center(2) = 0

radius = 1

Set outerLoop(0) = ThisDrawing.ModelSpace. _

AddCircle(center, radius)

' 附加外部边界到阴影对象，

' 并显示阴影

hatchObj.AppendOuterLoop (outerLoop)

hatchObj.Evaluate

ThisDrawing.Regen True

End Sub

编辑对象

修改现存对象，使用关联于对象的方法和属性。如果你图形对象的可见属性，要使用 **Update** 方法以刷新屏幕上的对象。

本节描述了怎样编辑二维对象。

工作于命名的对象

不仅是被 **AutoCAD** 使用的图形对象，还有多个类型的非图形对象保存在图形文件中。这些对象具有与其相关联的描述名称。例如，图块、图层、组合和标注样式。在大部分情况下，当你创建对象时命名对象，然后再进行重命名。名称保存在符号表中。当你指定了命名的对象，你引用该名称并关联到符号表中的对象数据。

本节内容：

清理命名的对象

重命名对象

清理命名的对象

你可在编辑进程中的任何时候从图形中清理无用的、没有被引用的命名对象。清理将减小图形的大小。你不能清理被其它对象引用的对象。例如，字体文件可能被文本样式所引用。图层中可能还存在着对象。

清理图形，使用 **PurgeAll** 方法。

清理图形

ThisDrawing.PurgeAll

重命名对象

当图形越来越复杂时，你可重命名对象以保持名称的可读性或避免在你将其插入到其它图形时产生的名称冲突。

除了 **AutoCAD** 的默认名称(如图层 0 或 CONTINUOUS 线型)外，你可重命名任何命名的对象。

名称最长可以是 255 个字符。不仅可以是字母和数字，名称也可以包含空格(虽然 **AutoCAD** 会删除名称前后的空格)和任何不在 **Microsoft**

Windows 或 **AutoCAD** 为其他的用途而使用的特殊字符。不能使用折特殊字符包括小于和大于符号(<>)、斜杠和反斜杠(/ \)、双引号(")、分号(;)、问号(?)、逗号(,)、星号(*)、竖杠(|)、等于号(=)和单引号(')。
你也不能使用创建于双字节字符集字体的特殊字符。

重命名对象，使用相应对象的 **Name** 属性。

重命名图层

本例创建名为"NewLayer"的图层，然后将图层重命名为"MyLayer"。

```
Sub Ch4_RenamingLayer()
```

```
' 创建图层
```

```
Dim layerObj As AcadLayer
```

```
Set layerObj = ThisDrawing.Layers.Add("NewLayer")
```

```
' 更改图层名称
```

```
layerObj.Name = "MyLayer"
```

End Sub

选择对象

选择集是作为独立单元进行处理的指定 **AutoCAD** 对象的组合。选择集可由单个对象组成，或它可以更加复杂的组合：例如，在确定图层的确定颜色的对象集。

定义选择集有两步过程。第一，你必须创建新的选择集并将其添加到 **SelectionSets** 集合。创建后，你就可往选择集上加入你想处理的对象。

本节内容：

创建选择集

添加对象到选择集

过滤选择集

从选择集上移去对象

创建选择集

创建命名的选择，使用 **Add** 方法。该方法只需要单独一个参数，就是选择集的名称。

创建一个空的选择集

本例创建一个新的选择集。

```
Sub Ch4_CreateSelectionSet()
```

```
Dim selectionSet1 As AcadSelectionSet
```

```
Set selectionSet1 = ThisDrawing.SelectionSets. _
```

Add("NewSelectionSet")

End Sub

添加对象到选择集

你可通过使用以下的方法添加对象到活动的选择集。

AddItem

添加一个或多个对象到指定的选择集。

Select

选择对象并将其放入活动的选择集中。你可选择所有对象、矩形窗选区域或矩形框选区域的对象、多边形窗选区域或多边形框选区域的对象、栅选的对象、最近创建的对象、最近选择集的对象。

SelectAtPoint

选择通过给定点的对象并将其放入活动的选择集中。

SelectByPolygon

通过栅选的对象并将其放入活动的选择集中。

SelectOnScreen

提示用户从屏幕中拾取对象并将其添加到活动的选择中。

添加对象到选择集中

本例提示用户选择对象，然后添加这些对象到选择集中。随后将选择集中对象的颜色改为蓝色。

```
Sub Ch4_AddToASelectionSet()

' 创建新的选择集

Dim sset As AcadSelectionSet

Set sset = ThisDrawing.SelectionSets.Add("SS1")

' 提示用户选择对象并将它们添加到选择集中。

' 要完成选择，按回车。

sset.SelectOnScreen

' 在选择集中循环并将每一对象的颜色改为蓝色。

Dim entry As AcadEntity

For Each entry In sset

    entry.Color = acBlue

    entry.Update

Next entry

End Sub
```


过滤选择集

你可用过滤器列表通过属性限制选择集，如通过颜色或对象类型。例如，你可以只复制电路板图形中的红色对象或确定图层中的对象。

注意： 过滤器只识别直接分配给对象颜色或线型，而非继承于图层的颜色或线型。

使用过滤器机制，必须提供用于分类过滤器类型和过滤器数据。过滤器类型是指定使用哪种过滤的代码。A

utoCAD

ActiveX 自动操作使用 DXF 组码来指定过滤器类型。最常用的几个过滤器类型在以下列出。完整的列表，可参见 AutoCAD

DXF 参考。

常用过滤器的 DXF 代码

DXF 代码 过滤器类型	
0	对象类型(字符串)如"Line"、"Circle"、"Arc"等。
2	对象名称(字符串)命名对象的表(给定)名称。
8	图层名称(字符串)如"Layer 0"。
60	对象的可见性(整数)使用 0 = 可见、1 = 不可见。
62	颜色号(整数)由 0 到 256 的数字索引值。0 代表随块。256 代表随层。负值代表图层关闭。
67	模型/图纸空间指示(整数)使用 0 或忽略=模型空间、1 = 图纸空间。

以下例子示范了不同的过滤器。

只添加文本对象到选择集：

FilterType =

0

FilterData = "TEXT"

sset.SelectOnScreen FilterType, FilterData

只添加直线对象到选择集:

FilterType =

0

FilterData = "LINE"

sset.SelectOnScreen FilterType, FilterData

只添加在图层 **FLOOR9** 上的对象到选择集:

FilterType =

8

FilterData = "FLOOR9"

sset.SelectOnScreen FilterType, FilterData

只添加颜色为红色的对象到选择集:

Filter Type

= 62

Filter Data = 5

sset.SelectOnScreen FilterType, FilterData

从选择集上移去对象

当你创建选择集后，你可从选择集中有选择地移去个别对象，或所有对象。例如，你可选择密集组合对象的整个组合并移去在组合中的指定对象，只剩下所要的集。

使用以下方法以从选择集中移去项目：

RemoveItems

RemoveItems 方法从选择集中移去一个或多个项目。所移去的项目仍然存在，只是它们不在选择集中。

Clear

Clear 将清空选择集。该选择集仍然存在，只是其中并无包含项目。先前在选择集中的项目仍然存在，只是它们不在选择集中。

Erase

Erase 方法删除选择集中的所有项目。该选择集仍然存在，只是其中并无包含项目。先前在选择集中的项目也将不复存在。

Delete

Delete 方法删除选择集和在选择集中的项目。在调用 **Delete** 后，不管是选择集还是先前在选择集中的项目将不再存在。

复制对象

你可在当前图形中复制单个或多个对象。偏移为从选定对象或通过指定点的指定距离创建新的对象。镜像为通过指定的镜像线创建对象的镜像图像。阵列创建按矩形或圆形方式排列的多重对象副本集。

注意：当在集合中循环的同时不可以执行以下的方法：当该方法试图执行读写操作时(因为循环只是以只读方式打开工作区)。在完成循环后你才能调用这些方法。

本节内容：

复制单个对象到同一位置

复制多个对象或复制到不同文档

偏移对象

镜像对象

阵列对象

复制单个对象到同一位置

复制单个对象，使用对象提供的 **Copy** 方法。该方法创建由原始对象所复制的新对象。新的对象与原始对象在同一位置，并由方法返回。

复制多个对象或复制到不同文档

复制多个对象，使用 **CopyObjects** 方法或创建对象数组以使用 **Copy** 方法(复制对象到选择集，在选择集里循环并保存对象到数组中)。在数组中循环，逐一复制每一对象，再将新创建的对象收集到第二个数组中。

复制多个对象

本例创建两个圆对象并使用 **CopyObjects** 方法生成圆的拷贝。

```
Sub Ch4_CopyCircleObjects()
```

```
Dim ACADApp As AcadApplication
```

```
Dim DOC1 As AcadDocument
```

```
Dim circleObj1 As AcadCircle
```

```
Dim circleObj2 As AcadCircle
```

```
Dim circleObj1Copy As AcadCircle
```

```
Dim circleObj2Copy As AcadCircle
```

```
Dim centerPoint(0 To 2) As Double
```

```
Dim radius1 As Double
```

```
Dim radius2 As Double
```

```
Dim radius1Copy As Double
```

```
Dim radius2Copy As Double
```

```
Dim objCollection(0 To 1) As Object
```

```
Dim retObjects As Variant
```

```
' 定义圆对象
```

```
centerPoint(0) = 0: centerPoint(1) = 0: centerPoint(2)
```

```
= 0
```

```
radius1 = 5#: radius2 = 7#
```

```
radius1Copy = 1#: radius2Copy = 2#
```

```
' 由系统获得应用程序对象
```

```
Set ACADApp = GetObject(, "AutoCAD.Application")
```

```
' 创建新图形
```

```
Set DOC1 = ACADApp.Documents.Add
```

```
' 添加两个圆到图形中
```

```
Set circleObj1 = DOC1.ModelSpace.AddCircle _
```

```
(centerPoint, radius1)
```

```
Set circleObj2 = DOC1.ModelSpace.AddCircle _
```

```
(centerPoint, radius2)
```

```
ZoomAll
```

```
' 将对象置于与 CopyObjects 相兼容的形式以便复制
```

```
Set objCollection(0) = circleObj1
```

```
Set objCollection(1) = circleObj2
```

```
' 复制对象并返回新对象(拷贝)的集合
```

```
retObjects = DOC1.CopyObjects(objCollection)
```

' 取得新创建对象并应用新属性到拷贝上

```
Set circleObj1Copy = retObjects(0)
```

```
Set circleObj2Copy = retObjects(1)
```

```
circleObj1Copy.radius = radius1Copy
```

```
circleObj1Copy.Color = acRed
```

```
circleObj2Copy.radius = radius2Copy
```

```
circleObj2Copy.Color = acRed
```

```
ZoomAll
```

```
End Sub
```

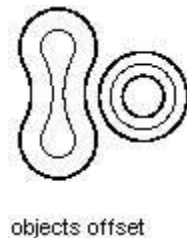
偏移对象

偏移对象是以与原始对象指定偏移距离创建新对象。你可偏移圆弧、圆、椭圆、直线、细多段线、多段线、样条曲线和构造直线。

偏移对象，使用对象提供的 **Offset** 方法。

在该方法中唯一的输入为偏移对象的距离。如果该距离为负值，这将被 AutoCAD 解释为偏移生成"小点"的曲线(也就是说，对于圆弧，它可能偏移生成的圆弧半径比起始曲线的半径小)。如果"小点"没有意义(象直线等)，则 AutoCAD 将向小的 X、Y、Z

WCS 坐标方向偏移。如果偏移距离无效，则返回错误。



对于很多对象，该操作的结果形成单一的新曲线(它的类型可能与原始曲线不同)。例如，偏移椭圆结果将会是样条曲线，这是因为结果不能与椭圆的方程式匹配。有时偏移结果可能形成多个曲线。由于这样，该方法返回新的对象或作为变体对象数组。

偏移多段线

本例创建细多段线然后偏移该多段线。再将新偏移形成的多段线的颜色改为红色。

```
Sub Ch4_OffsetPolyline()
```

```
' 创建多段线
```

```
Dim plineObj As AcadLWPolyline
```

```
Dim points(0 To 11) As Double
```

```
points(0) = 1: points(1) = 1
```

```
points(2) = 1: points(3) = 2
```

```
points(4) = 2: points(5) = 2
```



```
points(6) = 3: points(7) = 2
```

```
points(8) = 4: points(9) = 4
```

```
points(10) = 4: points(11) = 1
```

```
Set plineObj = ThisDrawing.ModelSpace. _
```

```
AddLightWeightPolyline(points)
```

```
plineObj.Closed = True
```

```
ZoomAll
```

```
' 偏移多段线
```

```
Dim offsetObj As Variant
```

```
offsetObj = plineObj.Offset(0.25)
```

```
offsetObj(0).Color = acRed
```

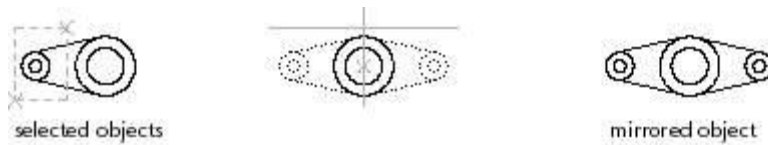
```
ZoomAll
```

```
End Sub
```

```
镜像对象
```

镜像是围绕一个轴或镜像轴线创建对象的镜像图像拷贝。你可镜像所在图形对象。

镜像对象，使用对象提供的 **Mirror** 方法。不象在 AutoCAD 中的镜像命令，该方法旋转反射图像到图形中并保留原始对象。(要移去原始对象，使用 **Erase** 方法)。



要操纵 **Text**(文本)对象的映象属性，使用 **MIRRTEXT** 系统变量。**MIRRTEXT** 的默认设定值为 **On** (1)，文本不被镜像。使用 **GetVariable** 和 **SetVariable** 方法可查询和设定 **MIRRTEXT** 系统变量值。



你可在图纸空间中镜像 **Viewport**(视口)对象，尽管这样做不会影响模型空间视图或模型空间的对象。

通过一个轴镜像多段线

本例创建细多段线然后通过一个轴镜像该多段线。再将新创建多段线的颜色改为红色。

```
Sub Ch4_MirrorPolyline()
```

```
' 创建多段线
```

```
Dim plineObj As AcadLWPolyline
```

```
Dim points(0 To 11) As Double
```

```
points(0) = 1: points(1) = 1
```

```
points(2) = 1: points(3) = 2
```

```
points(4) = 2: points(5) = 2
```

```
points(6) = 3: points(7) = 2
```

```
points(8) = 4: points(9) = 4
```

```
points(10) = 4: points(11) = 1
```

```
Set plineObj = ThisDrawing.ModelSpace. _
```

```
AddLightWeightPolyline(points)
```

```
plineObj.Closed = True
```

```
ZoomAll
```

```
' 定义镜像轴
```

```
Dim point1(0 To 2) As Double
```

```
Dim point2(0 To 2) As Double
```

```
point1(0) = 0: point1(1) = 4.25: point1(2) = 0
```

```
point2(0) = 4: point2(1) = 4.25: point2(2) = 0
```

' 镜像多段线

```
Dim mirrorObj As AcadLWPolyline
```

```
Set mirrorObj = plineObj.Mirror(point1, point2)
```

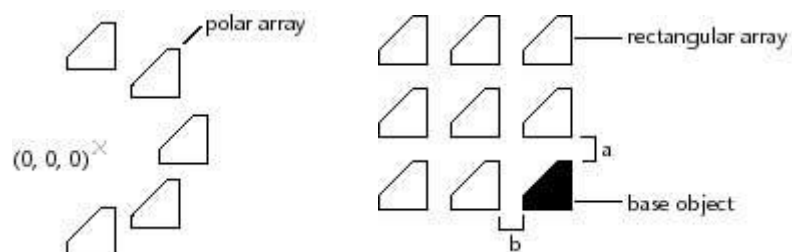
```
mirrorObj.Color = acRed
```

```
ZoomAll
```

```
End Sub
```

阵列对象

你可复制对象到圆周或矩形阵列。对于圆周阵列，你可控制对象拷贝的数量和阵列所填充的角度。对于矩形阵列，你可控制行和列的数量以及它们相互之间的距离。



创建圆周阵列

你可阵列所有图形对象。创建圆周阵列，使用对象提供的 **ArrayPolar** 方法。该方法需要你提供创建对象的数量、填充的角度和阵列的圆心。对象的数量必须大于 **1** 的正整数。填充的角度必须为弧度。正值为反时针方向。负值为顺时针方向。如果角度等于 **0** 时则出错。圆心为包含三位双精度数的变体数组。这些双精度数代表指定圆周阵列的中心点的三维 **WCS** 坐标。

AutoCAD 从阵列的圆心到原始对象的参考点确定距离。所用的参考点依赖于对象的类型。**AutoCAD** 使用圆或圆弧的圆心、图块或形的插入点、文本的起点以及直线或轨迹线的一个端点作为参考点。

在 AutoCAD 列阵命令的复制选项期间这方法不支持旋转。

创建圆周阵列

本例创建圆，然后进行圆的圆周阵列。所创建四个圆围绕基点(4,4,0)做 180 度填充。

```
Sub Ch4_ArrayingACircle()
```

```
' 创建圆
```

```
Dim circleObj As AcadCircle
```

```
Dim center(0 To 2) As Double
```

```
Dim radius As Double
```

```
center(0) = 2#: center(1) = 2#: center(2) = 0#
```

```
radius = 1
```

```
Set circleObj = ThisDrawing.ModelSpace. _
```

```
AddCircle(center, radius)
```

```
ZoomAll
```

```
' 定义圆周阵列
```

```
Dim noOfObjects As Integer
```

```
Dim angleToFill As Double
```

```
Dim basePnt(0 To 2) As Double
```

```
noOfObjects = 4
```

```
angleToFill = 3.14 ' 180 度
```

```
basePnt(0) = 4#: basePnt(1) = 4#: basePnt(2) = 0#
```

' 以下例子将围绕点(3,3,0)通过旋转和复制创建对象的四个拷贝。

```
Dim retObj As Variant
```

```
retObj = circleObj.ArrayPolar _
```

```
(noOfObjects, angleToFill, basePnt)
```

```
ZoomAll
```

```
End Sub
```

创建矩形阵列

创建二维或三维矩形阵列，使用对象所提供的 **ArrayRectangular** 方法。该方法需要你提供行数、列数、行间距离和列间距离。当你创建三维阵列时，你也必须指定层数及层间距离。

矩形阵列是通过相应次数地复制选择集中的对象而构成的。如果 你只定义一行，则你必须指定多于一列，反之亦然。

阵列是向上及向右生成的，原始对象被假定为左下角。如果行间距离为负值，则行向下添加。如果列间距离为负值，则列向左添加。

AutoCAD 通过以当前捕捉旋转角度定义的基线生成矩形阵列。该角度默认为 0 度，所以矩形阵列的行和列是与图形的 X 和 Y 轴正交的。你可通过设定捕捉旋转角度为非零值来更改该角度以创建旋转的阵列。要做到这一点，使用 **SnapRotationAngle** 属性。

创建矩形阵列

本例创建圆，然后执行圆的矩形阵列创建五行和五列的圆。

```
Sub Ch4_ArrayRectangularExample()
```

```
' 创建圆
```

```
Dim circleObj As AcadCircle
```

```
Dim center(0 To 2) As Double
```

```
Dim radius As Double
```

```
center(0) = 2#: center(1) = 2#: center(2) = 0#
```

```
radius = 0.5
```

```
Set circleObj = ThisDrawing.ModelSpace. _
```

```
AddCircle(center, radius)
```

```
ZoomAll
```

' 定义矩形阵列

Dim numberOfRows As Long

Dim numberOfColumns As Long

Dim numberOfLevels As Long

Dim distanceBwtnRows As Double

Dim distanceBwtnColumns As Double

Dim distanceBwtnLevels As Double

numberOfRows = 5

numberOfColumns = 5

numberOfLevels = 2

distanceBwtnRows = 1

distanceBwtnColumns = 1

distanceBwtnLevels = 1

' 创建对象阵列


```
Dim retObj As Variant
```

```
retObj = circleObj.ArrayRectangular _
```

```
(numberOfRows, numberOfColumns, numberOfLevels, _
```

```
distanceBwtnRows, distanceBwtnColumns, distanceBwtnLevels)
```

```
ZoomAll
```

```
End Sub
```

移动对象

你可在不改变对象方向或大小的情况下沿一个矢量移动对象。你也可围绕一个基点旋转对象。

本节内容：

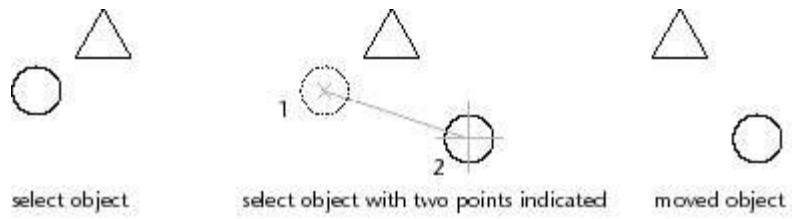
沿一个矢量移动对象

旋转对象

沿一个矢量移动对象

你可沿指定的矢量移动所有图形对象和属性参照对象。

移动对象，使用对象提供的 **Move** 方法。该方法需要输入两个坐标。这些坐标定义了位移矢量指示了对象移动多长距离及向哪个方向移动。



沿一个矢量移动圆

本例创建圆，然后沿 X 轴移动圆两个单位距离。

```
Sub Ch4_MoveCircle()
```

```
' 创建圆
```

```
Dim circleObj As AcadCircle
```

```
Dim center(0 To 2) As Double
```

```
Dim radius As Double
```

```
center(0) = 2#: center(1) = 2#: center(2) = 0#
```

```
radius = 0.5
```

```
Set circleObj = ThisDrawing.ModelSpace. _
```

```
AddCircle(center, radius)
```

```
ZoomAll
```

```
' 定义产生移动矢量的点。
```

' 移动矢量将使圆沿着 X 轴方向移动两个单位。

```
Dim point1(0 To 2) As Double
```

```
Dim point2(0 To 2) As Double
```

```
point1(0) = 0: point1(1) = 0: point1(2) = 0
```

```
point2(0) = 2: point2(1) = 0: point2(2) = 0
```

' 移动圆

```
circleObj.Move point1, point2
```

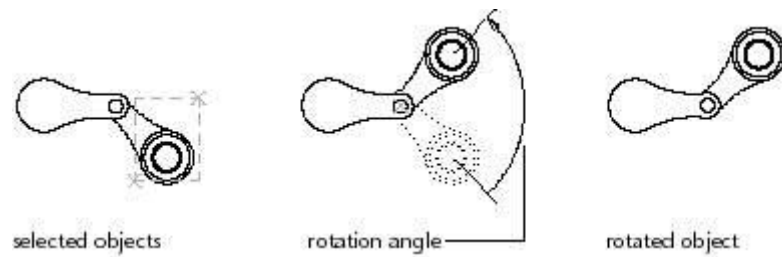
```
circleObj.Update
```

```
End Sub
```

旋转对象

你可旋转所有图形对象和属性参照对象。

旋转对象，使用对象提供的 **Rotate** 方法。该方法需要输入基点和旋转角度。基点是三位双精度数的变体数组。这些双精度数代表三维 **WCS** 坐标，指定了定义旋转轴的点。旋转角度为弧度。该角度确定了对象相对于当前位置围绕基点的旋转距离。



围绕基点旋转多段线

本例创建一闭合细多段线，然后围绕基点(4, 4.25, 0)按 45 度旋转多段线。

```
Sub Ch4_RotatePolyline()
```

```
' 创建多段线
```

```
Dim plineObj As AcadLWPolyline
```

```
Dim points(0 To 11) As Double
```

```
points(0) = 1: points(1) = 2
```

```
points(2) = 1: points(3) = 3
```

```
points(4) = 2: points(5) = 3
```

```
points(6) = 3: points(7) = 3
```

```
points(8) = 4: points(9) = 4
```

```
points(10) = 4: points(11) = 2
```

```
Set plineObj = ThisDrawing.ModelSpace. _
```

```
AddLightWeightPolyline(points)
```

```
plineObj.Closed = True
```

```
ZoomAll
```

```
' 定义旋转角度为 45 度及基点为(4, 4.25, 0)
```

```
Dim basePoint(0 To 2) As Double
```

```
Dim rotationAngle As Double
```

```
basePoint(0) = 4: basePoint(1) = 4.25: basePoint(2)  
= 0
```

```
rotationAngle = 0.7853981 ' 45 度
```

```
' 旋转多段线
```

```
plineObj.Rotate basePoint, rotationAngle
```

```
plineObj.Update
```

```
End Sub
```

删除对象

你可通过使用 **Delete** 方法删除单独的对象。

注意： 在 **ActiveX** 自动操作中的 **Collection** 对象也有 **Delete** 方法，那是因为这些对象被定义在类型库中。

然而，**Collection** 对象，就如 **ModelSpace** 集合、**Layers** 集合和 **Dictionaries** 集合是永远都不能删除的。如果你试图删除这些集合的话就会产生出错。

删除对象的示范代码

本例创建一细多段线，然后删除它。

```
Sub Ch4_DeletePolyline()
```

```
    ' 创建多段线
```

```
    Dim lwpolyObj As AcadLWPolyline
```

```
    Dim vertices(0 To 5) As Double
```

```
    vertices(0) = 2: vertices(1) = 4
```

```
    vertices(2) = 4: vertices(3) = 2
```

```
    vertices(4) = 6: vertices(5) = 4
```

```
    Set lwpolyObj = ThisDrawing.ModelSpace. _
```

```
    AddLightWeightPolyline(vertices)
```

```
    ZoomAll
```

```
' 删除多段线
```

```
lwpolyObj.Delete
```

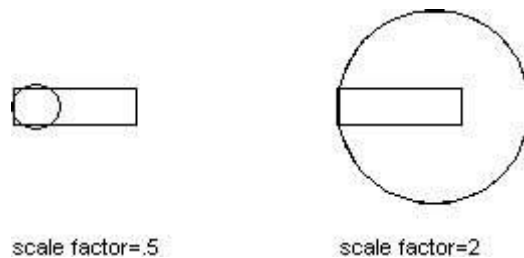
```
ThisDrawing.Regen acActiveViewport
```

```
End Sub
```

比例缩放对象

你可通过指定基点和长度(作为基于当前图形单位的比例因子)来比例缩放对象。你可比例缩放所有图形对象和属性参照对象。

比例缩放对象，使用对象提供的 **ScaleEntity** 方法。该方法在 **X**、**Y** 和 **Z** 方向等比例缩放对象。它接受输入比例缩放的基点和比例因子。基点为三位双精度数的变体数组。这些双精度数代表三维 **WCS** 坐标，指明了比例缩放的起始点。比例因子为缩放对象的比例。对象的尺寸是与比例因子乘积缩放。比例因子大于 **1** 时放大对象。比例因子在 **0** 和 **1** 之间时缩小对象。



比例缩放多段线

本例创建一闭合细多段线，然后用 **0.5** 的比例缩放多段线。

```
Sub Ch4_ScalePolyline()
```

```
' 创建多段线
```

```
Dim plineObj As AcadLWPolyline
```

```
Dim points(0 To 11) As Double
```

```
points(0) = 1: points(1) = 2
```

```
points(2) = 1: points(3) = 3
```

```
points(4) = 2: points(5) = 3
```

```
points(6) = 3: points(7) = 3
```

```
points(8) = 4: points(9) = 4
```

```
points(10) = 4: points(11) = 2
```

```
Set plineObj = ThisDrawing.ModelSpace. _
```

```
AddLightWeightPolyline(points)
```

```
plineObj.Closed = True
```

```
ZoomAll
```

```
' 定义比例
```

```
Dim basePoint(0 To 2) As Double
```

```
Dim scalefactor As Double
```



```
basePoint(0) = 4: basePoint(1) = 4.25: basePoint(2)  
= 0
```

```
scalefactor = 0.5
```

```
' 比例缩放多段线
```

```
plineObj.ScaleEntity basePoint, scalefactor
```

```
plineObj.Update
```

```
End Sub
```

转换对象

你可使用 **TransformBy** 方法给定一个 **4×4** 转换矩阵移动、比例缩放或旋转对象。

下表示范了转换矩阵结构，这里 **R** 为旋转，**T** 为转换：

转换矩阵结构

```
R00 R01 R02 T0
```

```
R10 R11 R12 T1
```

```
R20 R21 R22 T2
```

```
0 0 0 1
```

转换对象，首先要初始化转换矩阵。下例显示了一个分配给变量 **tMatrix** 的转换矩阵，它将围绕点(0, 0)按 90 度旋转图元：

```
tMatrix(0,0)
```

```
= 0.0
```

```
tMatrix(0,1) = -1.0
```

```
tMatrix(0,2) = 0.0
```

```
tMatrix(0,3) = 0.0
```

```
tMatrix(1,0) = 1.0
```

```
tMatrix(1,1) = 0.0
```

```
tMatrix(1,2) = 0.0
```

```
tMatrix(1,3) = 0.0
```

```
tMatrix(2,0) = 0.0
```

```
tMatrix(2,1) = 0.0
```

```
tMatrix(2,2) = 1.0
```

```
tMatrix(2,3) = 0.0
```

```
tMatrix(3,0) = 0.0
```

```
tMatrix(3,1) = 0.0
```

```
tMatrix(3,2) = 0.0
```

```
tMatrix(3,3) = 1.0
```

当完成了转换矩阵后，使用 **TransformBy** 方法应用矩阵到对象上。以下代码行示范了应用矩阵(**tMatrix**)到对象(**anObj**):

```
anObj.TransformBy
```

```
tMatrix
```

使用转换矩阵旋转直线

本例创建一直线并使用转换矩阵将其旋转 **90** 度。

```
Sub Ch4_TransformBy()
```

```
' 创建一直线
```

```
Dim lineObj As AcadLine
```

```
Dim startPt(0 To 2) As Double
```

```
Dim endPt(0 To 2) As Double
```

```
startPt(0) = 2
```

```
startPt(1) = 1
```

```
startPt(2) = 0
```

```
endPt(0) = 5
```

```
endPt(1) = 1
```

```
endPt(2) = 0
```

```
Set lineObj = ThisDrawing.ModelSpace. _
```

```
AddLine(startPt, endPt)
```

```
ZoomAll
```

```
' 初始化 transMat 转换矩阵的变量，
```

```
'以使其围绕点(0,0,0)并按 90 度旋转对象。
```

```
Dim transMat(0 To 3, 0 To 3) As Double
```

```
transMat(0, 0) = 0#: transMat(0, 1) = -1#
```

```
transMat(0, 2) = 0#: transMat(0, 3) = 0#
```

```
transMat(1, 0) = 1#: transMat(1, 1) = 0#
```

```
transMat(1, 2) = 0#: transMat(1, 3) = 0#
```

```
transMat(2, 0) = 0#: transMat(2, 1) = 0#
```

```
transMat(2, 2) = 1#: transMat(2, 3) = 0#
```

```
transMat(3, 0) = 0#: transMat(3, 1) = 0#
```

```
transMat(3, 2) = 0#: transMat(3, 3) = 1#
```

'使用定义了的转换矩阵转换直线

```
lineObj.TransformBy transMat
```

```
lineObj.Update
```

```
End Sub
```

以下为转换矩阵的更多例子：

旋转矩阵：围绕点(0,0,0)按 90 度

```
0.0 -1.0 0.0
```

```
0.0
```

```
1.0 0.0 0.0 0.0
```

```
0.0 0.0 1.0 0.0
```

```
0.0 0.0 0.0 1.0
```

旋转矩阵：围绕点(5, 5, 0)按 45 度

0.707107 -0.707107

0.0 5.0

0.707107 0.707107 0.0 -2.071068

0.0 0.0 1.0 0.0

0.0 0.0 0.0 1.0

转换矩阵：通过(10,10,0)移动图元

1.0 0.0 0.0 10.0

0.0 1.0 0.0 10.0

0.0 0.0 1.0 0.0

0.0 0.0 0.0 1.0

比例矩阵：按点(0, 0, 0)比例为 10,10 缩放

10.0 0.0 0.0

0.0

0.0 10.0 0.0 0.0

0.0 0.0 10.0 0.0

0.0 0.0 0.0 1.0

比例矩阵：按点(2, 2, 0)比例 10,10 缩放

```
10.0 0.0 0.0
```

```
-18.0
```

```
0.0 10.0 0.0 -18.0
```

```
0.0 0.0 10.0 0.0
```

```
0.0 0.0 0.0 1.0
```

延伸和修剪对象

可以改变圆弧的角度，也可以改变非闭合的直线、圆弧、非闭合多段线、椭圆弧和非闭合样条曲线的长度。结果与延伸和修剪相似。

你可通过编辑对象属性来延伸或修剪对象。例如，要加长一条直线，只需更改 **StartPoint**(起点)或 **EndPoint**(终点)属性。要更改圆弧的角度，只需更改圆弧的 **StartAngle** 或 **EndAngle** 属性。当你改变了对对象的一个或多个属性，使用 **Update** 方法使你的更改在图形中可见。

加长直线

本例创建一直线，然后更改直线的终点使线更长。

```
Sub Ch4_LengthenLine()
```

```
' 定义并创建直线
```

```
Dim lineObj As AcadLine
```

```
Dim startPoint(0 To 2) As Double
```

```
Dim endPoint(0 To 2) As Double
```

```
startPoint(0) = 0
```

```
startPoint(1) = 0
```

```
startPoint(2) = 0
```

```
endPoint(0) = 1
```

```
endPoint(1) = 1
```

```
endPoint(2) = 1
```

```
Set lineObj = ThisDrawing.ModelSpace. _
```

```
AddLine(startPoint, endPoint)
```

```
lineObj.Update
```

' 通过更改终点到 4,4,4 以加长直线

```
endPoint(0) = 4
```

```
endPoint(1) = 4
```

```
endPoint(2) = 4
```

```
lineObj.endPoint = endPoint
```

```
lineObj.Update
```


End Sub

分解对象

分解对象把单个的对象转换成它们下一个层次的组成对象，但有时看不出对象有什么变化。例如，分解多段线、矩形、圆环和多边形将把它们转换成多个简单的直线和圆弧。另外，分解对象将把块参照或关联标注替换成组成块或标注的简单对象。

你可分解三维多边形，细多段线、多段线、多边形网面、面域和图块参照。分解对象，使用 **Explode** 方法。

一个被分解的对象看起来与原有对象没有任何不同，但其颜色、线型和线宽可能改变。

分解多段线时，**AutoCAD** 将清除关联的宽度信息。生成的直线和圆弧将遵循多段线的点划线设置。如果分解包含多段线的块，则需要单独分解多段线。

用不相等的 **X**、**Y** 和 **Z** 比例因子插入的块可能会产生不可预料的结果。不能分解外部参照和它们依赖的块。如果分解具有属性的块，属性将被清除，但创建它们的属性定义仍会被保留。属性值和任何通过 **ATTEDIT** 命令所进行的修改将会丢失。

分解多段线

本例创建一细多段线对象。然后将其分解为分离的对象。随后在结果的对象中循环并显示包含每一对象名称和其在分解对象列表中的序号的消息框。

```
Sub Ch4_ExplodePolyline()
```

```
Dim plineObj As AcadLWPolyline
```

```
Dim points(0 To 11) As Double
```

```
' 定义二维多段线点
```

```
points(0) = 1: points(1) = 1
```

```
points(2) = 1: points(3) = 2
```

```
points(4) = 2: points(5) = 2
```

```
points(6) = 3: points(7) = 2
```

```
points(8) = 4: points(9) = 4
```

```
points(10) = 4: points(11) = 1
```

```
' 创建细多段线对象
```

```
Set plineObj = ThisDrawing.ModelSpace. _
```

```
AddLightWeightPolyline(points)
```

```
' 设定多段线上的一个线段为凸出量
```

```
' 以改变多段线中对象的类型
```

```
plineObj.SetBulge 3, -0.5
```

```
plineObj.Update
```

' 分解多段线

Dim explodedObjects As Variant

explodedObjects = plineObj.Explode

' 在分解的对象中循环并显示每一对象类型的消息框

Dim I As Integer

For I = 0 To UBound(explodedObjects)

explodedObjects(I).Color = acRed

explodedObjects(I).Update

MsgBox "分解的对象 " & I & ": " & _

explodedObjects(I).ObjectName

explodedObjects(I).Color = acByLayer

explodedObjects(I).Update

Next

End Sub

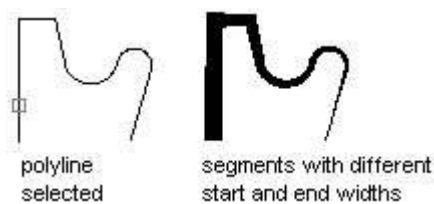
编辑多段线

二维和三维多段线、矩形、多边形和三维多边形网面均为多段线的变形，可用同一种途径进行编辑。

AutoCAD 可以识别拟合多段线和样条拟合多段线。样条拟合多段线使用曲线拟合，类似于 **B-样条曲线**。样条曲线拟合多段线有两种类型：二次和三次。这些多段线都受 **SPLINETYPE** 系统变量控制。拟合多段线使用标准曲线进行曲线拟合，它利用在任何给定顶点设置的切线方向。

你可通过闭合或打开多段线、以及移动、添加或删除单个顶点来编辑多段线。你可将多段线上任意两个顶点之间的线段拉直，你可切换其线型以在整条多段线上的所有顶点前后生成连续线型。你可设定整条多段线为统一的宽度或控制每段的宽度。

你可合并与多段线的端点重合的直线、圆弧或其它多段线到开放的多段线上。如果线穿过多段线的端点形成 **T** 字形，则该对象不能合并。如果两直线与多段线相遇成 **Y** 字形，则 **AutoCAD** 选择其中一条线合并到多段线中。合并同样造成隐含的非曲线化，因为 **AutoCAD** 会放弃原始多段线及合并的多段线的样条曲线信息。当合并完成后，你可拟合出新的样条曲线。



编辑多段线，使用 **LightweightPolyline** 或 **Polyline** 对象的属性和方法。使用以下属性打开或闭合多段线、更改多段线顶点坐标或添加顶点：

Closed

打开或闭合多段线。

Coordinates

指定多段线上每一顶点的坐标。

AddVertex

添加顶点到细多段线上。

使用以下方法更新多段线的凸出量及宽度：

SetBulge

给定序号设定多段线的凸出量。

SetWidth

给定序号设定多段线起点及终点的宽度。

编辑多段线

本例创建一细多段线。然后添加凸出量到多段线的第三段，添加一顶点到多段线上，更改最后一段的宽度，最后闭合多段线。

```
Sub Ch4_EditPolyline()
```

```
Dim plineObj As AcadLWPolyline
```

```
Dim points(0 To 9) As Double
```

```
' 定义二维多段线点
```

```
points(0) = 1: points(1) = 1
```

```
points(2) = 1: points(3) = 2
```

```
points(4) = 2: points(5) = 2
```

```
points(6) = 3: points(7) = 2
```

```
points(8) = 4: points(9) = 4
```

```
' 创建一细多段线对象
```

```
Set plineObj = ThisDrawing.ModelSpace. _
```

```
AddLightWeightPolyline(points)
```

```
' 增加第三段的凸出量
```

```
plineObj.SetBulge 3, -0.5
```

```
' 定义新的顶点
```

```
Dim newVertex(0 To 1) As Double
```

```
newVertex(0) = 4: newVertex(1) = 1
```

```
' 添加顶点到多段线上
```

```
plineObj.AddVertex 5, newVertex
```

' 设定新线段的宽度

```
plineObj.SetWidth 4, 0.1, 0.5
```

' 闭合多段线

```
plineObj.Closed = True
```

```
plineObj.Update
```

```
End Sub
```

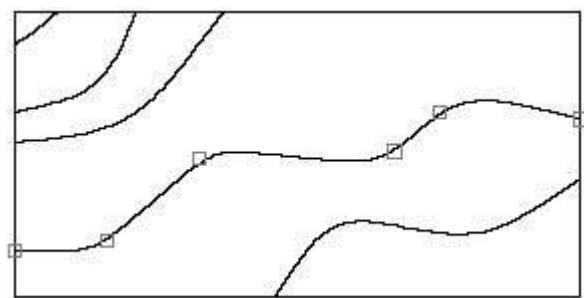
编辑样条曲线

可以删除样条曲线的拟合点，也可以为提高精度而增加拟合点，或者移动拟合点修改样条曲线的形状。还可以打开或闭合样条曲线，编辑样条曲线的起始和末端的切线。样条曲线的方向是可反转的。也可以改变样条曲线的允差。允差表示样条曲线拟合所指定的拟合点集时的拟合精度。允差越小样条曲线与拟合点集的接近程度越高。

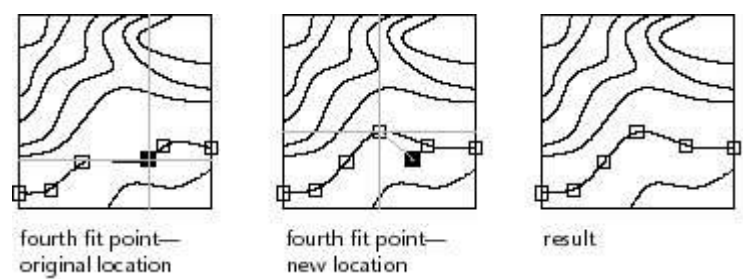
可以向一段样条曲线中增加控制点的数目或改变指定的控制点的权值来细化样条曲线。增加控制点的权值将把样条曲线进一步拉向该点。也可以通过改变它的阶来细化样条曲线。样条曲线的阶是样条多项式的次数加一。例如，三次样条曲线的阶为

4。样条曲线的阶越高，控制点越多。

下面的图例中：



已创建一条样条曲线表示地形轮廓。要移动第四个拟合点以提高精确度。使用 **SetFitPoint** 方法为第四个拟合点输入新的坐标。



使用以下可编辑属性更改样条曲线：

Closed

打开或闭合样条曲线。

ControlPoints

指定样条曲线的控制点。

EndTangent

指定作为样条曲线方向矢量的终切向。

FitPoints

指定样条曲线的所有拟合点。

FitTolerance

用新的容许值重新拟合样条曲线到现有的点。

Knots

指定样条曲线的节矢量。

StartTangent

指定样条曲线的起始切向。

另外，你可使用以下方法编辑样条曲线：

AddFitPoint

按给定的序号添加单个拟合点到样条曲线。

DeleteFitPoint

按给定的序号删除样条曲线的拟合点。

ElevateOrder

提升样条曲线的阶数到给定的阶数。

GetFitPoint

获得给定序号的样条曲线拟合点。(只能获得一个拟合点。如果查询样条曲线的所有拟合点，使用 **FitPoints** 属性。

Reverse

反转样条曲线的方向。

SetControlPoint

设定给定序号的样条曲线的控制点。

SetFitPoint

设定给定序号的样条曲线拟合点)只能设定一个拟合点。如果要更改样条曲线的所有拟合点，使用 **FitPoints** 属性。

SetWeight

设定给定序号的控制点重量。

使用以下只读属性查询样条曲线：

Degree

获取样条曲线多项式表示的角度。

Area

获取样条曲线附着的面积。

IsPeriodic

指定给定样条曲线是否为循环。

IsPlanar

指定给定样条曲线是否在一平面上。

IsRational

指定给定样条曲线是否为有理数。

NumberOfControlPoints

获取样条曲线的控制点数目。

NumberOfFitPoints

获取样条曲线的拟合点数目。

更改样条曲线上的控制点

本例创建一样条曲线，然后更改样条曲线的第一个控制点。

```
Sub Ch4_ChangeSplineControlPoint()
```

```
' 创建样条曲线
```

```
Dim splineObj As AcadSpline
```

```
Dim noOfPoints As Integer
```

```
Dim startTan(0 To 2) As Double
```

```
Dim endTan(0 To 2) As Double
```

```
Dim fitPoints(0 To 8) As Double
```

```
noOfPoints = 3
```

```
startTan(0) = 0.5: startTan(1) = 0.5: startTan(2) =
```

```
0
```

```
endTan(0) = 0.5: endTan(1) = 0.5: endTan(2) = 0
```

```
fitPoints(0) = 1: fitPoints(1) = 1: fitPoints(2) = 0
```

```
fitPoints(3) = 5: fitPoints(4) = 5: fitPoints(5) = 0
```

```
fitPoints(6) = 10: fitPoints(7) = 0: fitPoints(8) =  
0
```

```
Set splineObj = ThisDrawing.ModelSpace. _
```

```
AddSpline(fitPoints, startTan, endTan)
```

```
splineObj.Update
```

```
' 更改第一个控制点的坐标。
```

```
Dim controlPoint(0 To 2) As Double
```

```
controlPoint(0) = 0
```

```
controlPoint(1) = 3
```

```
controlPoint(2) = 0
```

```
splineObj.SetControlPoint 0, controlPoint
```

```
splineObj.Update
```

End Sub

编辑阴影

你可以编辑阴影边界和阴影图案。如果编辑关联阴影的边界，只要编辑结果位于有效边界，则图案将会被更新。即使关联阴影所处的图层已被关闭，它们也会进行更新。可以修改阴影图案或为现有的图案阴影选择新图案，但关联性只能在阴影创建时设定。你可通过使用 **AssociativeHatch** 属性查看阴影对象是否关联。(参考 **AddHatch** 方法以得到更多关于创建阴影的信息)。

你必须使用 **Evaluate** 方法重新求值阴影以看到对阴影的编辑。

本节内容：

编辑阴影边界

编辑阴影图案

编辑阴影边界

你可附加或插入回路到阴影边界中。关联阴影随边界的变化而更新。非关联阴影将不被更新。

编辑阴影边界，使用以下的方法：

AppendInnerLoop

附加内部回路到阴影中。

AppendOuterLoop

附加外部回路到阴影中。

InsertLoopAt

插入回路到阴影的给定索引。

附加内部回路到阴影中

本例创建一关联阴影。然后创建一个圆再把圆做为内部回路附加到阴影中。

```
Sub Ch4_AppendInnerLoopToHatch()
```

```
Dim hatchObj As AcadHatch
```

```
Dim patternName As String
```

```
Dim PatternType As Long
```

```
Dim bAssociativity As Boolean
```

```
' 定义并创建阴影
```

```
patternName = "ANSI31"
```

```
PatternType = 0
```

```
bAssociativity = True
```

```
Set hatchObj = ThisDrawing.ModelSpace. _
```

```
AddHatch(PatternType, patternName, bAssociativity)
```

' 创建阴影的外部回路。

```
Dim outerLoop(0 To 1) As AcadEntity
```

```
Dim center(0 To 2) As Double
```

```
Dim radius As Double
```

```
Dim startAngle As Double
```

```
Dim endAngle As Double
```

```
center(0) = 5: center(1) = 3: center(2) = 0
```

```
radius = 3
```

```
startAngle = 0
```

```
endAngle = 3.141592
```

```
Set outerLoop(0) = ThisDrawing.ModelSpace. _
```

```
AddArc(center, radius, startAngle, endAngle)
```

```
Set outerLoop(1) = ThisDrawing.ModelSpace. _
```

```
AddLine(outerLoop(0).startPoint, outerLoop(0).endPoint)
```

' 附加外部回路到阴影对象上

hatchObj.AppendOuterLoop (outerLoop)

' 创建作为阴影内部回路的圆

Dim innerLoop(0) As AcadEntity

center(0) = 5: center(1) = 4.5: center(2) = 0

radius = 1

Set innerLoop(0) = ThisDrawing.ModelSpace. _

AddCircle(center, radius)

' 将圆作为内部回路附加到阴影

hatchObj.AppendInnerLoop (innerLoop)

' 重新求值并显示阴影

hatchObj.Evaluate

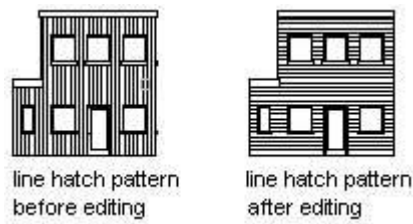
ThisDrawing.Regen True

End Sub

编辑阴影图案

你可以修改现有阴影图案的角度或间距，用实体填充或 **AutoCAD** 提供的预先定义的图案替换它。在边界阴影圣诞框中的图案选项显示这些图案的列表。为了减小文件的大小，图形中定义的阴影将作为单一的图形对象。

在下面的样例中，将修改填充图案的角度。



使用以下属性和方法编辑阴影图案：

PatternAngle

指定阴影图案的角度。

PatternDouble

指定在用户定义阴影的双向性。

PatternName

指定阴影图案名称(并没有改变图案类型)。

PatternScale

指定阴影图案比例。

PatternSpace

指定用户定义阴影图案的间距。

SetPattern

设定阴影图案名称和图案类型。

更改阴影图案的间距

本例创建一个阴影。然后加 2 到阴影的当前图案间距。

```
Sub Ch4_ChangeHatchPatternSpace()
```

```
Dim hatchObj As AcadHatch
```

```
Dim patternName As String
```

```
Dim PatternType As Long
```

```
Dim bAssociativity As Boolean
```

```
' 定义阴影
```

```
patternName = "ANSI31"
```

```
PatternType = 0
```

```
bAssociativity = True
```

```
' 创建关联阴影对象
```

```
Set hatchObj = ThisDrawing.ModelSpace. _
```

```
AddHatch(PatternType, patternName, bAssociativity)
```

```
' 创建阴影的外部回路
```

```
Dim outerLoop(0 To 0) As AcadEntity
```

```
Dim center(0 To 2) As Double
```

```
Dim radius As Double
```

```
center(0) = 5
```

```
center(1) = 3
```

```
center(2) = 0
```

```
radius = 3
```

```
Set outerLoop(0) = ThisDrawing.ModelSpace. _
```

```
AddCircle(center, radius)
```

```
hatchObj.AppendOuterLoop (outerLoop)
```

```
hatchObj.Evaluate
```

```
' 通过当前间距加 2 更改阴影图案的间距
```

```
hatchObj.patternSpace = hatchObj.patternSpace + 2
```

```
hatchObj.Evaluate
```

```
ThisDrawing.Regen True
```

```
End Sub
```

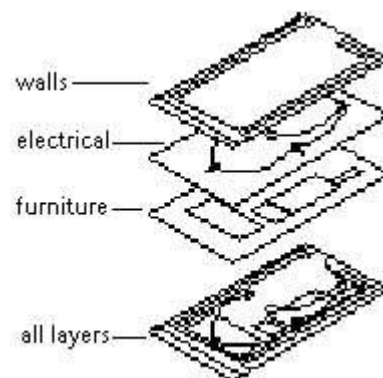
使用图层、颜色和线型

图层就象是透明的覆盖图，运用它可以很好地组织不同类型的图形信息。您创建的对象都具有的特性包括图层、颜色和线型等。颜色有助于区分图形中相似的元素，线型则可以轻易地区分不同的绘图元素（例如中心线或隐藏线）。组织图层和图层上的对象使得处理图形中的信息更加容易。

使用图层

任何图形对象都是绘制在图层上的。该图层可能是默认图层，或者是自己创建和命名的图层。每个图层都有与其相关联的颜色、线型、线宽和打印样式。例如，可以创建一个用于绘制中心线的图层，并为该图层指定中心线需具备的特性（如颜色、线型和线宽）。在绘制中心线时切换到中心线图层开始绘图，而无需在每次绘制中心线时去设置线型、线宽和颜色。同时，如果不想显示或打印某个特定的图层，可以关闭该图层或关闭该图层的打印。使用图层是

AutoCAD 代替纸和笔创建图形的主要优点之一。



在布局（图纸空间）中，或当你在浮动视口中，可以分别为每个视口指定图层的可见性。相同的图形界限、坐标系和缩放比例将应用到图形中的所有图层。如果要重复使用一个特定的图层方案，就可以用该图层及其已指定的关联线型、线宽、颜色和打印样式建立样板图形。

本节内容：

图层和线型分类

创建和命名图层

使图层成为当前图层

控制图层的可见性

打开和关闭图层

冻结和解冻图层

锁定和解锁图层

指定图层颜色

指定图层线型

删除图层

图层和线型分类

所有图层和线型都是在其上层的 **Collection** 对象中。图层是在 **Layers** 集合中，线型是在 **Linetypes** 集合中。

你可在集合中循环以查找图形中的所有图层和线型。

在 **Layers** 中循环

以下代码在 **Layers** 集合中循环并收集图形中所有图层的名称。这些名称将显示在消息框中。

```
Sub Ch4_IteratingLayers()

    Dim layerNames
    As String

    Dim entry As
    AcadLayer

    layerNames =
    ""

    For Each entry
    In ThisDrawing.Layers

        layerNames =
        layerNames + entry.Name + vbCrLf

    Next

    MsgBox "图形中的图层有：

    " + _

    vbCrLf + layerNames
```

End Sub

创建和命名图层

可以为在设计概念上相关的一组对象（例如墙或标注）创建和命名图层，并为这些图层指定通用特性。当组织你的图层方案时，请慎重选择图层名称。

开始绘制一个新图形时，AutoCAD 将创建一个名为 0 的特定图层。默认时，图层 0 将被指定编号为 7 的颜色（白色或黑色，由背景色决定）、CONTINUOUS（连续）线型、“缺省”线宽（“缺省”的缺省设置是 .01 英寸或 .25 毫米）以及“普通”打印样式。图层 0 不能被删除或重命名。

可以创建新图层并为其指定颜色、线型、线宽和打印样式特性。每一单独的图层均为 Layers 集合的一部分。使用 Add 方法创建新的图层并将其添加到 Layers 集合中。

创建图层时可指定图层的名称。在图层创建后更改图层的名称，使用 Name 属性。图层名称可包含至多 31 个字符，可以包含字母、数字和特殊字符美元符号(\$)、连字号(-)和下划线(_)，但不能包含空格。

指定对象到新图层中

以下代码创建一个圆和一个新的图层。新的图层指定为红色。将圆指定到该图层上，此时圆的颜色将因此而改变。

Sub Ch4_NewLayer()

' 创建圆

```
Dim circleObj As AcadCircle
```

```
Dim center(0 To 2) As Double
```

```
Dim radius As Double
```

```
center(0) = 2: center(1) =
```

```
2: center(2) = 0
```

```
radius = 1
```

```
Set circleObj = ThisDrawing.ModelSpace.
```

```
—
```

```
AddCircle(center, radius)
```

' 指定圆的颜色为“ByLayer”以使圆

' 可以随着图层的颜色变化而变化。

```
circleObj.Color = acByLayer
```

' 创建名为“ABC”的新图层

```
Dim layerObj As AcadLayer
```

```
Set layerObj = ThisDrawing.Layers.Add("ABC")
```


' 指定“ABC”图层的颜色为红色

```
layerObj.Color = acRed
```

' 指定圆到“ABC”图层上

```
circleObj.Layer = "ABC"
```

```
circleObj.Update
```

```
End Sub
```

使图层成为当前图层

绘图操作总是在当前图层上进行的。将某个图层设置为当前图层后，可以从中创建新对象。如果想让其他图层成为当前图层，则后面创建的对象都将在新的当前图层上面，并使用它的颜色、线型、线宽和打印样式（此时所有对象特性保留“随层”缺省值）。不能将被冻结的图层或依赖外部参照的图层设置为当前图层。

设置图层为当前图层，使用 **ActiveLayer** 属性。该属性是在当前图形中设定。例如：

```
Dim newlayer As AcadLayer
```

```
Set newlayer = ThisDrawing.Layers.Add("LAYER1")
```

```
ThisDrawing.ActiveLayer = newlayer
```

控制图层的可见性

AutoCAD 不显示和打印绘制在不可见图层上的对象。在图形中，被冻结或关闭的图层是不可见的。如果在某一个或一组图层上详细绘图时需要一个无遮挡的视图，则可以关闭图层或冻结图层。如果不想打印某些细节（例如构造线或参照线），也可以冻结、关闭图层，或者关闭可见图层的打印。

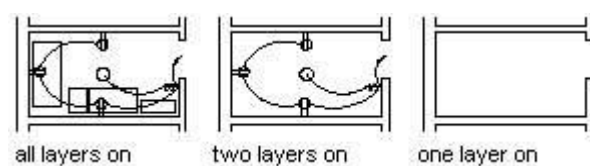
使用哪种控制图层可见性的方法由绘图方式和图形复杂程度决定。可以冻结长时间不需要显示的图层。

只有那些打开的和解冻的图层才能打印输出，从而能够控制图层是否打印。对于大多数打印机，可指定不同的画笔给图形中的每一颜色号。对于单笔打印机，**AutoCAD** 中止笔的更改。

打开和关闭图层

关闭的图层与图形一起重生成，但不能被显示或打印。如果要频繁地将图层从可见切换到不可见，可以关闭图层而不用冻结。关闭图层而不冻结，可以避免每次解冻图层时重生成图形。打开已关闭的图层时，**AutoCAD**

将重画该图层上的对象。



要打开和关闭图层，使用 **LayerOn** 属性。如果输入值为 **TRUE** 到该属性中，该图层将打开，如果输入值为 **FALSE** 到属性，则图层将关闭。

关闭图层

本例创建一新图层，添加一个圆到该图层中，然后关闭图层以使用圆不可见。

```
Sub Ch4_LayerInvisible()
```

```
' 创建圆
```

```
Dim circleObj As AcadCircle
```

```
Dim center(0 To 2) As Double
```

```
Dim radius As Double
```

```
center(0) = 2: center(1) =
```

```
2: center(2) = 0
```

```
radius = 1
```

```
Set circleObj = ThisDrawing.ModelSpace.
```

```
—
```

```
AddCircle(center, radius)
```

```
circleObj.Color = acByLayer
```

```
' 创建名为“ABC”的新图层
```

```
Dim layerObj As AcadLayer
```

```
Set layerObj = ThisDrawing.Layers.Add("ABC")
```

```
layerObj.Color = acRed
```

' 指定图到“ABC”图层中

```
circleObj.Layer = "ABC"
```

```
circleObj.Update
```

' 关闭图层“ABC”

```
layerObj.LayerOn = False
```

```
ThisDrawing.Regen acActiveViewport
```

```
End Sub
```

冻结和解冻图层

冻结图层可以加速显示的变化，提高对象选择的性能，减少复杂图形的重生成时间。**AutoCAD** 不能在被冻结的图层上显示、打印或重生成对象。可以将长期不需要显示的图层冻结。解冻已冻结的图层时，**AutoCAD**

将重生成并显示该图层上的对象。

要冻结或解冻图层，使用 **Freeze** 属性。如果输入值 **TRUE** 到该属性中，则图层冻结。如果输入值 **FALSE**，则图层解冻。

冻结图层

本例创建一名为“ABC”的新图层，然后冻结该图层。

```
Sub Ch4_LayerFreeze()  
  
' 创建名为"ABC"的新图层  
  
Dim layerObj As AcadLayer  
  
Set layerObj = ThisDrawing.Layers.Add("ABC")  
  
  
  
' 冻结图层"ABC"  
  
layerObj.Freeze = True  
  
End Sub
```

锁定和解锁图层

如果要编辑与特殊图层相关联的对象，同时又想查看但不编辑其他图层对象，那么可以锁定图层。锁定图层上的对象不能被编辑或选择，然而，如果该图层处于打开状态并被解冻，上面的对象仍是可见的。可以使被锁定的图层成为当前图层并在其中创建新对象。可以冻结和关闭锁定的图层和更改与其相关联的颜色和线型。

要锁定或解锁图层，使用 **Lock** 属性。如果输入值 **TRUE** 到该属性，图层将被锁定。如果输入值 **FALSE**，则图层被解锁。

锁定图层

本例创建一名为"ABC"的新图层，然后锁定该图层。

```
Sub Ch4_LayerLock()
```

```
' 创建名为"ABC"的新图层
```

```
Dim layerObj As AcadLayer
```

```
Set layerObj = ThisDrawing.Layers.Add("ABC")
```

```
' 锁定图层"ABC"
```

```
layerObj.Lock = True
```

```
End Sub
```

指定图层颜色

你可为图层指定颜色。指定颜色时，你可输入颜色名称或颜色的 **AutoCAD** 颜色索引号(ACI)。标准颜色名称只对颜色 1 到 7 有效。

默认下，**AutoCAD** 指定编号为 7 的颜色（白色或黑色，由背景色决定）给新创建的图层。你可给对象指定不同于图层颜色的颜色。

要指定图层的颜色，使用 **Color** 属性。

颜色可在 0~256 的范围内作为数字索引值进行设定和读取。对于标准的 7 种颜色和 **BYBLOCK** 及 **BYLAYER** 名称提供了常量。

如果使用 **acByBlock**，**AutoCAD** 用默认颜色绘制新对象（白色或黑色，由背景色决定）直到它们被组合到图块中。当图块插入图形时，在图块中的对象继承当前 **Color** 属性的设置。

如果使用 **acByLayer**，新对象采用其对应图层上的颜色。

指定图层线型

当定义图层时，线型提供另外的方法传达可视信息。线型是点、线和空白等的重复图案，你可用其来区分线与线之间的不同意图。

线型的名称和定义描述了详细线点的次序、线与空白之间的相对长度以及包含着文字和形状的特征。

指定图层的线型，使用 **Linetype** 属性。该属性利用线型的名称作为输入。

删除图层

删除图层，使用 **Delete** 方法。

在绘图期间随时都能删除图层。但不能删除当前图层、图层 **0**、依赖外部参照的图层或包含对象的图层。

注意 被块定义参照的图层和名为 **DEFPOINTS** 的特殊图层也不能被删除，即使它们不包含可见对象。

使用颜色

可以给图层和图形中单独的对象指定颜色。每一颜色均由名称或 **AutoCAD** 颜色索引(ACI)号 **1** 至 **255** 的整数来识别。任何数目的对象和图层可拥有相同的颜色号。可以指定每一颜色号到笔式绘图仪的不同画笔或使用颜色号以识别图形中的某些对象，即使不能看到屏幕上的颜色。

本节内容：

指定颜色

设定当前颜色

指定颜色

指定颜色时，可输入颜色名称或其 **ACI** 号。**ACI** 提供了 255 个颜色号。标准颜色名称只对颜色 1 至 7 有效。

颜色 1 至 7

颜色号	颜色名称
1	Red(红色)
2	Yellow(黄色)
3	Green(绿色)
4	Cyan(青色)
5	Blue(蓝色)
6	Magenta(紫色)
7	Black/White(黑/白)

颜色 8 至 255 必须通过号码或在对话框中选择颜色来指定。默认的颜色(7)

为白色或黑色，由背景色决定。

设定当前颜色

可以给图层或创建的对象指定颜色。可以定义当前颜色作为当前图层的颜色，或指定其他不同的颜色。

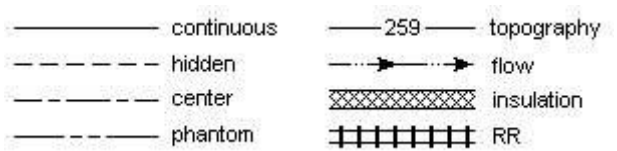
如果选择了 **BYLAYER**(随层)，新建对象将采用对象所绘制图层的颜色。如果选择了 **BYBLOCK**(随块)，新建对象将用默认的颜色绘制直到他们组合到图块中。在图块中的对象将继承当前的颜色设置。

设定对象或图层的当前颜色，使用 **Color** 属性。

使用线型

线型是点、横线和空格按一定规律重复出现形成的图案。复杂线型是符号与点、横线、空格组合的图案。

线型名及其定义描述了一定的点划序列、横线和空格的相对长度，以及任何包含文字或形的特性。用户可以创建自定义线型。



要使用线型，必须首先将其加载到图形中。在将线型加载到图形中之前，线型定义必须已存在于 LIN 库文件中。加载线型为图形中，使用 **Load** 方法。

加载线型到 AutoCAD

本例尝试从 acad.lin 文件中加载线型“CENTER”。如果线型存在，或文件不存在，则显示一消息。

```
Sub Ch4_LoadLinetype()
```

```
On Error GoTo
```

```
ERRORHANDLER
```

```
Dim linetypeName
```

```
As String
```

```
linetypeName
```

```
= "CENTER"
```

'从 acad.lin 文件中加载

"CENTER" 线型

ThisDrawing.Linetypes.Load

linetypeName, "acad.lin"

Exit Sub

ERRORHANDLER:

MsgBox Err.Description

End Sub

注意 不要将 AutoCAD 内部使用的线型与某些绘图仪提供的硬件线型混淆。这两种类型的虚线产生的效果相似。不要同时使用这两种类型，否则，可能会产生不可预料的后果。

本节内容：

使线型成为当前线型

重命名线型

删除线型

更改线型说明

指定线型比例

使线型成为当前线型

要使用一种线型绘制新对象，必须选择一种线型并将其设置为当前线型。所有新创建对象都使用当前线型绘制。

注意 依赖外部参照的线型不能被设置为当前线型。

如果选择 **BYLAYER**，新对象采用当前线型的线型属性。如果选择 **BYBLOCK**，新对象将使用他们组合到图块中的线型进行绘制。图块中的对象将继承当前线型属性。

使线型成为当前线型，使用 **ActiveLinetype** 属性。该属性是在当前图形中设定。例如

```
ThisDrawing.ActiveLinetype
```

```
= ThisDrawing. _
```

```
Linetypes.Item("CONTINUOUS")
```

重命名线型

为了使线型的名称更好地表明其作用，可以重命名线型。在绘图期间随时都可以重命名线型。

重命名线型时，重命名的只是图形中的线型定义。**LIN** 库文件中的线型名称不会被更新以反映新名称。

重命名线型，使用 **Name** 属性。

注意：不能重命名“随层”、“随块”和 **CONTINUOUS**（连续）这几种线型和依赖外部参照的线型。

删除线型

在绘图期间随时可以删除线型。但是，有一些线型是不能被删除的，包括“随层”、“随块”、“CONTINUOUS（连续）”、当前线型和依赖外部参照的线型，还有图形中的对象使用的线型。同样，块定义参照的线型即使不包含可见对象也不能被删除。

注意 删除一个线型时，只是从当前图形中将其删除，而不是从 LIN 库文件中删除。

删除线型，使用 **Delete** 方法。

更改线型说明

线型可以有与其相关联的说明。说明提供了线型的 **ASCII** 表示。可以使用 **Description** 属性指定或更改线型说明。

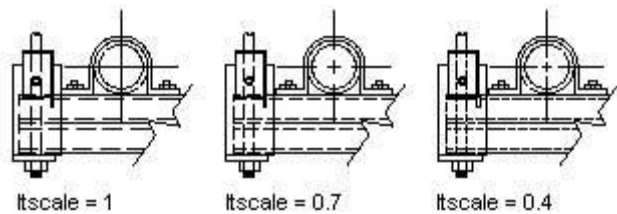
线型说明可以有 47 个字符。说明可以是注释或一系列的下划线、点、连字号和空格以显示线型图案的简单描述。例如：

```
ThisDrawing.ActiveLinetype.Description  
= "外墙"
```

指定线型比例

可以为所创建的对象设置全局线型缩放比例。该值越小，每个绘图单位中画出的重复图案越多。在缺省情况下，AutoCAD

的全局线型缩放比例为 1.0，该比例等于一个绘图单位。可更改所有图形对象、属性参照和组合的线型比例。



更改线型比例，使用 LinetypeScale 属性。

系统变量 CELTSCALE 为新创建的对象设置线型缩放比例。LTSCALE

全局修改现有对象和新对象的线型缩放比例。用 AutoCAD ActiveX 自动操作修改系统变量的值，可使用 SetVariable 方法。

修改圆的线型比例

```
Sub Ch4_ChangeLinetypeScale()
```

```
Dim center(0
```

```
To 2) As Double
```

```
Dim radius As
```

```
Double
```

```
Dim circleObj
```

```
As AcadCircle
```

```
' 在模型空间中创建一个圆对象
```

```
center(0) =  
2  
  
center(1) =  
2  
  
center(2) =  
0  
  
radius = 4  
  
Set circleObj  
= ThisDrawing.ModelSpace. _  
  
AddCircle(center,  
radius)  
  
' 设定圆的线型比例为 0.5  
  
circleObj.LinetypeScale  
= 0.5  
  
circleObj.Update  
  
End Sub
```

分配图层、颜色和线型给对象

当定义了图层、颜色和线型后，可在图形中将其指定给对象。可通过指定对象给图层以组合图形中的关联构件。可控制图层的可见性、颜色和线型并指定图层中的对象是否可编辑。可将对象从一个图层中移动到另外的图层中并修改图层的名称。

图形中图层的数量和每一图层中对象的数量事实上是无限制的。可指定名称给每一图层和选择任意关联的图层以显示。

可从最初绘制于使用不同颜色和线型的不同图层中的对象定义图块。可保护图块中对象的图层、颜色和线型信息。然后，每次在插入图块时，每一对象将用其初始颜色和线型的初始图层绘制出来。

本节内容：

更改对象的图层

更改对象的颜色

更改对象的线型

更改对象的图层

当创建对象并指定其图层、颜色和线型属性后，可能想去更改对象的图层。如果你意外地将对象创建于错误的图层中或决定更改随后的图层结构时，更改对象的图层是很有用的。

更改对象的图层，使用对象提供的 **Layer** 属性。该图层属性需要图层的名称作为输入。

移动对象到不同的图层

本例在当前图层中创建一个圆，然后创建一个名为“ABC”的新图层。随后将圆移动到新的图层中。

Sub Ch4_MoveObjectNewLayer()

' 创建一个圆

```
Dim circleObj
```

```
As AcadCircle
```

```
Dim center(0
```

```
To 2) As Double
```

```
Dim radius As
```

```
Double
```

```
center(0) =
```

```
2: center(1) = 2: center(2) = 0
```

```
radius = 1
```

```
Set circleObj
```

```
= ThisDrawing.ModelSpace. _
```

```
AddCircle(center,
```

```
radius)
```

```
' 创建一个名为 "ABC"的新图层
```

```
Dim layerObj
```

```
As AcadLayer
```

```
Set layerObj
```

```
= ThisDrawing.Layers.Add("ABC")
```

```
' 指定圆到 "ABC"
```

```
图层
```



```
circleObj.Layer
```

```
= "ABC"
```

```
circleObj.Update
```

```
End Sub
```

更改对象的颜色

可指定颜色到图形中的单独对象。每一颜色是通过 **ACI** 号(从 1 到 255 的整数)来识别。标准颜色常量只对颜色 1 至 7 有效。这些颜色常量为

acRed、**acYellow**、**acGreen**、**acCyan**、**acBlue**、**acMagenta** 和 **acWhite**

设定对象的颜色将覆盖对象所在的图层中的颜色设置。

如果想保留对象在指定的图层中又不想保持图层的颜色，可更改对象的颜色。更改对象的颜色，使用对象提供的 **Color** 属性。

更改圆的颜色

本例创建一个圆，然后将圆的颜色改为红色。

```
Sub Ch4_ColorCircle()
```

```
' 创建一个圆
```

```
Dim circleObj As AcadCircle
```

```
Dim center(0 To 2) As Double
```

```
Dim radius As Double
```

```
center(0) = 2: center(1) = 2: center(2)  
= 0
```

```
radius = 1
```

```
Set circleObj = ThisDrawing.ModelSpace.
```

```
—
```

```
AddCircle(center, radius)
```

```
' 将圆的颜色改为红色
```

```
circleObj.Color = acRed
```

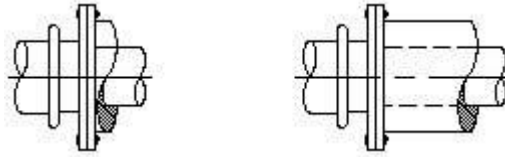
```
circleObj.Update
```

```
End Sub
```

更改对象的线型

可使用当前线型创建对象。如果当前线型设置为 **BYLAYER**，所创建对象使用的线型为当前图层的线型。线型的默认设定为 **BYLAYER**。

可将线型关联到所有 **AutoCAD** 对象，但对于文本、点、视口、构造直线、射线、三维多段线和图块线型是不会显示出。如果线条太短而不能保留至少一个段序列，**AutoCAD** 高尔夫球在两端点之间绘制一连续线。



默认情况下，对象继承其所创建时所在图层的线型。更改对象的线型，使用对象提供的 **Linetype** 属性。Linetype 属性要求指定对象的线型名称作为输入。

更改圆的线型

本例创建一个圆。然后尝试从 **acad.lin** 文件中加载线型“CENTER”。如果线型存在，或文件不存在，则显示一消息。最后将圆的线型设定为“CENTER”。

```
Sub Ch4_ChangeCircleLinetype()
```

```
On Error Resume
```

```
Next
```

```
' 创建一个圆
```

```
Dim circleObj
```

```
As AcadCircle
```

```
Dim center(0
```

```
To 2) As Double
```

```
Dim radius As
```

```
Double
```

```
center(0) =
```

```
2: center(1) = 2: center(2) = 0
```

```
radius = 1
```

```
Set circleObj
```

```
= ThisDrawing.ModelSpace. _
```

```
AddCircle(center,  
radius)
```

```
Dim linetypeName
```

```
As String
```

```
linetypeName
```

```
= "CENTER"
```

```
' 从 acad.lin 文件中加载线型"CENTER"
```

```
ThisDrawing.Linetypes.Load
```

```
linetypeName, "acad.lin"
```

```
If Err.Description
```

```
<> "" Then MsgBox Err.Description
```

```
' 指定圆的线型为"CENTER"
```

```
circleObj.Linetype
```

```
= "CENTER"
```

```
circleObj.Update
```

End Sub

注意：在给对象指定线型前，线型必须已经加载到当前图形中。加载线型到图形中，使用 **Load** 方法。

添加文本到图形中

图形中的文字表达了重要的信息。可以在标题块中使用文字，还可以用文字标记图形的各个部分、提供说明或进行注释。

AutoCAD 提供了多种创建文字的方法。对简短的输入项使用单行文字，对带有内部格式的较长的输入项使用多行文字。虽然所有输入的文字都使用当前文字样式建立缺省字体和格式设置，但也可自定义文字外观。

本节内容：

处理文字样式

使用单行文字

使用多行文字

使用双字节字符、控制代码和特殊字符

替换字体

拼写检查

处理文字样式

AutoCAD 图形中的所有文字都有与之相关联的文字样式。当输入文字时，**AutoCAD**

使用当前的文字样式，该样式设置字体、字号、角度、方向和其他文字特性。文字样式可控制下表列出的各项属性：

文字样式设置

设置	缺省	说明
样式名	STANDARD	最长为 255 个字符
字体名	txt.shx	与字体相关联的文件（字符样式）
大字体	非	用于非 ASCII 字符集的特殊形定义文件。如 Kanji
高度	0	字符高度
宽度比例	1	延展或压缩字符
倾斜角度	0	倾斜字符
反向	否	反向文字
倒置	否	倒置文字
垂直	否	垂直或水平文字

可使用或修改默认样式或创建和加载新的样式。当创建是样式后，可修改其属性或在不再需要时将其删除。

本节内容：

创建和修改文字样式

指定字体

使用 TrueType 字体

使用双字节和大字体

设定文字高度

设定倾斜角度

设定文字生成标记

创建和修改文字样式

除了缺省的 **STANDARD** 样式外，必须创建所需文字样式。新输入的文字将继承当前文字样式的高度、宽度比例、倾斜角、反向、倒置和垂直对齐等特性。创建文字样式，使用 **Add** 方法以创建一新的 **TextStyle** 对象并将其添加到 **TextStyles** 集合中。**Add** 方法需要要文字样式的名称作为输入。创建以后，可能通过 **AutoCAD**

ActiveX 自动操作更改文字样式的名称。

样式名称可包含字母、数字和特殊字符美元符号(**\$**)、连字号(**-**)和下划线(**_**)。**AutoCAD** 将这些字符转换为大写。如果未输入样式名称，**AutoCAD** 将自动命名样式名为 **Style**
n，数字 **n** 起始为 **1**。每新建一个样式它会增加 **1**。

可通过更改 **TextStyle** 对象的属性修改现存的样式。也可更新现存文字的样式类型以更改文字的样式。使用以下属性以修改 **TextStyle** 对象：

FontFile

指定关联字体(字符样式)的文件。

BigFontFile

指定用于非 **ASCII** 字符集的特殊形定义文件，如中文。

Height

指定字符的高度。

Width

指定字符的扩展和压缩。

ObliqueAngle

指定字符的倾斜度。

TextGenerationFlag

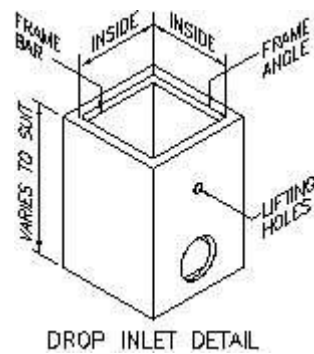
指定反向文字、倒置文字或两者。

如果更改现存样式的字体或方向。所有应用该样式的文字也将更改为使用新的字体或方向。更改文字的高度、宽度因子和倾斜角度不会更改现存文字，但它会更改随后创建的文字对象。

注意：必须调用 **Regen** 或 **Update** 方法以看到对以上属性的更改。

指定字体

字体定义了构成字符集的文字字符的形状。在多个样式中可以使用同一种字体。假如您的公司使用标准字体类型，可修改其他样式设置来创建一个样式集，以不同的方式使用标准字体。下图表明了在不同的样式中使用同一字体，这些样式具有定义字体倾斜的不同倾斜设置。



可使用 **TextStyle** 对象的 **FontFile** 属性来指定文字样式中的字体。通过输入包含 **AutoCAD** 编译的 **SHX** 字体的字体文件，就可指定字体到文字样式中。

设定文字字体

本例获取活动文字样式的当前字体值，然后更改字体的字样为“**PlayBill**”。新的字体使用 **SetFont** 方法进行设定。

```
Sub Ch4_UpdateTextFont()
```

```
Dim typeFace As String
```

```
Dim Bold As Boolean
```

```
Dim Italic As Boolean
```

```
Dim charSet As Long
```

Dim PitchandFamily As Long

' 用 SetFont 方法获取当前设置的默认值

ThisDrawing.ActiveTextStyle.GetFont

typeFace, _

Bold, Italic, charSet, PitchandFamily

' 更改字体的字样

typeFace = "PlayBill"

ThisDrawing.ActiveTextStyle.SetFont

typeFace, _

Bold, Italic, charSet, PitchandFamily

ThisDrawing.Regen acActiveViewport

End Sub

使用 TrueType 字体

图形中的 TrueType 字体是以填充方式显示出来的，在打印时，TEXTFILL

系统变量控制是否填充该字体。TEXTFILL 系统变量的缺省设置为 1，这时打印出填充的字体。当用 PSO

UT

输出图形并在 PostScript 设备上打印时，打印出的字体与设计时的一样。

为了在 AutoCAD 中提高 TrueType 字体的显示速度和性能，Windows

操作系统直接绘制了一些 TrueType 文字。由于受 Windows 的限制，AutoCAD 必须绘制用特定的方法转换的

TrueType 文字。例如，镜像文字、倒置文字、反向文字、倾斜文字，它们宽度比例不为 1 或者位于非屏幕的共面方向。基本规则是：如果

AutoCAD 中的 TrueType 文字与字处理器中的文字一致，则该文字是由 Windows 系统绘制的，否则是由

AutoCAD 绘制。在某些情况下，转换后的文字显得稍粗一些，特别是在低分辨率下。这些差异仅仅出现在字体的显示上，并不影响打印输出。

使用 Unicode 和大字体

现在 AutoCAD 支持 Unicode 字符编码标准。Unicode

字体可包含 65,535 个字符和为多种语言设计的形。Unicode 字体包含的字符比系统中定义的多。因此，要想使用不能直接从键盘上输入的字符，可以输入转义序列

+nnnn，其中 nnnn 表示字符的 Unicode 十六进制值。现在 AutoCAD 的所有 SHX 形字体都是 Unicode 字体。

在 R13 以前的版本中使用的 SHX 字体不支持 +nnnn 序列，但是你仍可以用

128 - 256 范围内的字符生成带读音符号的字符。

对于一些包含几千个非 ASCII 字符的字母表文本文件，例如汉字，AutoCAD

相应地提供一种称作大字体文件的特殊类型的形定义。可以用常规字体和大字体文件来设置样式。

要想获得 AutoCAD 提供的标准字体的样例，请参见命令参考附录 E

标准库。AutoCAD 还提供了替换字体或指定缺省字体的方法（请参见替换字体）。

指定普通字体使用 **FontFile** 属性。指定大字体使用 **BigFontFile** 属性。

更改字体文件

本例更改 **FontFile** 和 **BigFontFile** 属性。你需要替换列于本例中的路径信息以与你的系统中的路径的文件名称相配。

```
Sub Ch4_ChangeFontFiles()
```

```
    ThisDrawing.ActiveTextStyle.BigFontFile
```

```
    = _
```

```
    "C:/Program Files/ACAD2000/Fonts/bigfont.shx"
```

```
    ThisDrawing.ActiveTextStyle.fontFile
```

```
    = _
```

```
    "C:/Program Files/ACAD2000/Fonts/italic.shx"
```

```
End Sub
```

注意：包含有逗号的长文件名不能作为字体文件名称接受。

设定文字高度

文字高度确定在使用的字体中以图形单位计算的字母大小。除 **TrueType** 字体外，该值通常表示大写字母的大小。

对于 **TrueType** 字体，文字的高度值可能并不表示大写字母的高度。指定的高度表示首字母的高度加上重音标识的升调区和其他用于非英语语言中的标志。指定给首字母和升调字符的相对区域部分由字体设计者在设计字体时决定，因此各种字体之间会有所不同。

除了组成用户指定高度的首字母和重音标志区域以外，**TureType** 字体还有字符部分的下划区域，延伸到文字基线以下，如

y、j、p、g 和 q。

指定文字高度使用 **Height** 属性。该属性只接受正数。

更改文字对象的高度

本例创建一行文字，然后更改文字的高度。

```
Sub Ch4_ChangeTextHeight()
```

```
Dim textObj As AcadText
```

```
Dim textString As String
```

```
Dim insertionPoint(0 To 2) As Double
```

```
Dim height As Double
```

```
' 定义文字对象
```

```
textString = "Hello, World."
```

```
insertionPoint(0) = 3
```

```
insertionPoint(1) = 3
```

```
insertionPoint(2) = 0
```

```
height = 0.5
```

' 在模型空间中创建文字对象

```
Set textObj = ThisDrawing.ModelSpace.
```

```
—
```

```
AddText(textString, insertionPoint,  
height)
```

' 更改高度值为 1

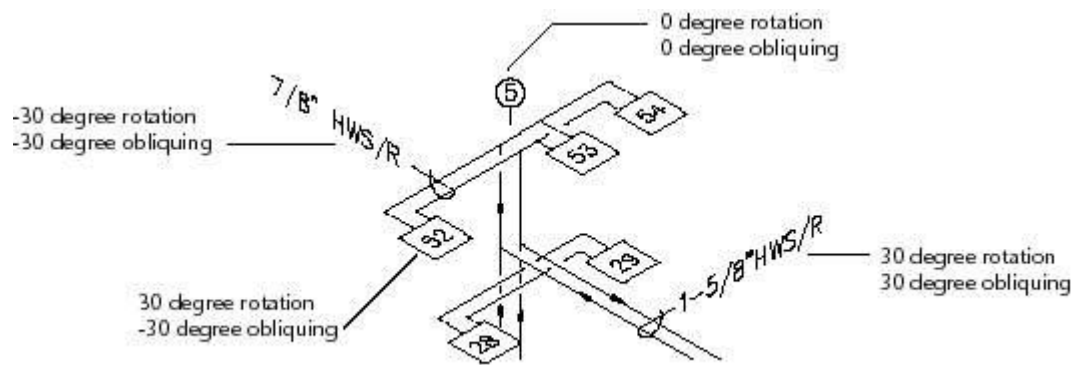
```
textObj.height = 1
```

```
textObj.Update
```

```
End Sub
```

设定倾斜角度

倾斜角决定了文字是向前还是向后倾斜。倾斜角表示的是相对于 90 度角方向的偏移角度。



设定倾斜角度，使用 **ObliqueAngle** 属性。该倾斜角度必须提供弧度。正的角度表示向右倾斜，负值时将会自动加上 2π 以转换为正值。

创建倾斜文字

本例创建一文字对象，然后将文字倾斜 **45 度**。

```
Sub Ch4_ObliqueText()
```

```
Dim textObj As AcadText
```

```
Dim textString As String
```

```
Dim insertionPoint(0 To 2) As Double
```

```
Dim height As Double
```

```
' 定义文字对象
```

```
textString = "Hello, World."
```

```
insertionPoint(0) = 3
```

```
insertionPoint(1) = 3
```

```
insertionPoint(2) = 0
```

```
height = 0.5
```

```
' 在模型空间中创建文字对象
```

```
Set textObj = ThisDrawing.ModelSpace.
```

```
—
```

```
AddText(textString, insertionPoint,  
height)
```

```
' 更改 ObliqueAngle 值为 45 度(0.707 弧度)
```

```
textObj.ObliqueAngle = 0.707
```

```
textObj.Update
```

```
End Sub
```

设定文字生成标记

文字的生成标记指定了文字是否显示为反向或倒置。

设定文字的生成标记，使用 `TextGenerationFlag` 属性。要将文字反向时，将该属性设置为 `acTextFlagBackward`。要将文字倒置时，将该属性设置为 `acTextFlagUpsideDown`。要将同时文字反向和倒置时，将两个常量加在一起，也就是将该属性设置为 `acTextFlagBackward+acTextFlagUpsidedown`。

文字反向显示

本例创建一行文字，然后使用 `TextGenerationFlag` 属性将其设置为反向显示。

```
Sub Ch4_ChangingTextGenerationFlag()
```

```
Dim textObj As AcadText
```

```
Dim textString As String
```

```
Dim insertionPoint(0 To 2) As Double
```

```
Dim height As Double
```

```
' 创建文字对象
```

```
textString = "Hello, World."
```

```
insertionPoint(0) = 3
```

```
insertionPoint(1) = 3
```

```
insertionPoint(2) = 0
```

```
height = 0.5
```

```
Set textObj = ThisDrawing.ModelSpace.
```

```
—
```

```
AddText(textString, insertionPoint,  
height)
```

' 更改 TextGenerationFlag 的值

```
textObj.TextGenerationFlag = acTextFlagBackward
```

```
textObj.Update
```

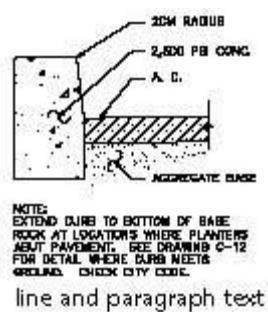
```
End Sub
```

使用单行文字

添加到图形中的文字可以表达各种信息。它可能是复杂的规格说明、标题块信息、标签或图形的一部分。

对于不需要使用多种字体的简短内容，如标签，可使用

TEXT 对象创建单行文字。单行文字对于标签来说更为方便。



本节内容：

创建单行文字

格式化单行文字

对象单行文字

更改单行文字

创建单行文字

当使用单行文字时每一单独行的文字均为分离的对象。创建单行文字对象，使用 **AddText** 方法。该方法需要三个值作为输入：文字字符串、插入点和文字高度。

文字字符串是显示的实际文字。它可接受 **Unicode**、控制代码和特殊字符。插入点是放置文字的位置，它是一个包含代表图形中三维 **WCS** 坐标的三个双精度变体数组。文字高度为代表大写文字高度的正数。高度以当前单位计算。

创建单行文字

本例在模型空间的坐标点(2, 2, 0)创建一单行文字。

```
Sub Ch4_CreateText()
```

```
Dim textObj As AcadText
```

```
Dim textString As String
```

```
Dim insertionPoint(0 To 2) As Double
```

```
Dim height As Double
```

```
' 创建文字对象
```

```
textString = "Hello, World."
```

```
insertionPoint(0) = 2
```

```
insertionPoint(1) = 2
```

```
insertionPoint(2) = 0
```

```
height = 0.5
```

```
Set textObj = ThisDrawing.ModelSpace.
```

```
—
```

```
AddText(textString, insertionPoint,  
height)
```

```
textObj.Update
```

```
End Sub
```

格式化单行文字

文字对象是使用当前文字样式创建的。可通过更改关联到文字对象的文字样式更改文字对象的格式，也可通过编辑文字对象的属性来更改。你不能应用格式到单独的单词或字符。

更改关联到单独文字对象的文字样式，可设定 **StyleName** 属性为新的文字样式。当更改了文字样式后，应使用文字对象的 **Update** 方法以看到图形中更改后的文字。

除了图元的标准可编辑属性(颜色、图层、线型等)外，文字对象的其它可更改属性包括以下这些：

Alignment

指定文字的水平和垂直排列。

InsertionPoint

指定文字的插入点。

ObliqueAngle

指定单独文字对象的倾斜角度。

Rotation

指定用弧度表示的文字的旋转角度。

ScaleFactor

指定文字的比例因子。

TextAlignmentPoint

指定文字的对象点。

TextGenerationFlag

指定文字是否显示为反向、倒置或两者同时。

TextString

指定显示的实际文本字符串。

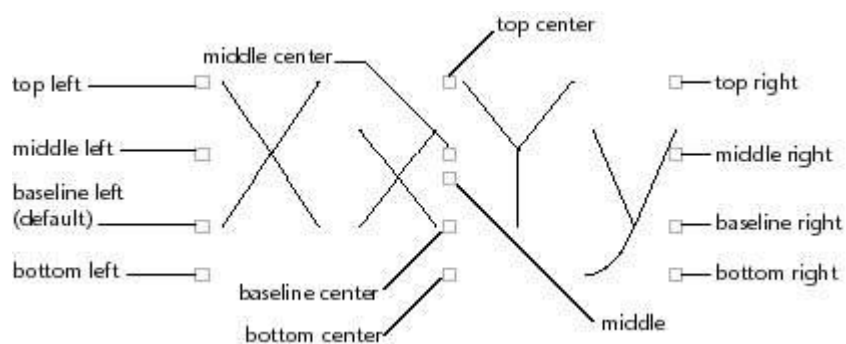
当更改了属性后，使用 **Update** 方法以看到图形中的更改。

注意：要得到完整的方法和属性列表，请查看 **AutoCAD**

ActiveX 和 **VBA** 参考有关 **Text** 对象的文档。

对象单行文字

可根据下图所示的对齐选项之一对齐文字。缺省设置为左对齐。设置水平和垂直对象选项，使用 **Alignmen**
t 属性。



重新对象文字

本例创建一个 **Text** 对象和一个 **Point** 对象。点对象设定为文字的对齐点，并将点改为红色十字叉线以可视。

每次更改了文字的对齐方式后均显示一消息框以暂停宏的执行。这样可让你看到文字对齐更改后的效果。

```
Sub Ch4_TextAlignment()
```

```
Dim textObj As AcadText
```

```
Dim textString As String
```

```
Dim insertionPoint(0 To 2) As Double
```

```
Dim height As Double
```

```
' 定义新的 Text 对象
```

```
textString = "Hello, World."
```

```
insertionPoint(0) = 3
```

```
insertionPoint(1) = 3
```

```
insertionPoint(2) = 0
```

```
height = 0.5
```

```
' 在模型空间中创建 Text 对象
```

```
Set textObj = ThisDrawing.ModelSpace.
```

```
—
```

```
AddText(textString, insertionPoint,  
height)
```

```
' 在文字的对齐点上创建一个点，
```

```
' 这样可更好的看到对齐的处理过程
```

```
Dim pointObj As AcadPoint
```

```
Dim alignmentPoint(0 To 2) As Double
```

```
alignmentPoint(0) = 3
```

```
alignmentPoint(1) = 3
```

```
alignmentPoint(2) = 0
```

```
Set pointObj = ThisDrawing.ModelSpace.
```

```
—
```

```
AddPoint(alignmentPoint)
```

```
pointObj.Color = acRed
```

```
' 设置点样式为十字交叉
```

```
ThisDrawing.SetVariable "PDMODE",
```

```
2
```


' 左对齐文字

```
textObj.Alignment = acAlignmentLeft
```

```
ThisDrawing.Regen acActiveViewport
```

```
MsgBox "文字对象现在为左对齐"
```

' 中心对齐文字

```
textObj.Alignment = acAlignmentCenter
```

' 将文字对齐该点(除了左对齐文字外，其它均为必须)

```
textObj.TextAlignmentPoint = alignmentPoint
```

```
ThisDrawing.Regen acActiveViewport
```

```
MsgBox "文字对象现在为中心对齐"
```

' 右对齐文字

```
textObj.Alignment = acAlignmentRight
```

```
ThisDrawing.Regen acActiveViewport
```

```
MsgBox "文字对象现在为右对齐"
```

End Sub

更改单行文字

和任何其他对象一样，可以移动、旋转、删除和复制单行文字对象。也可以镜像或制作反向文字的副本。

如果在镜像文字时不打算使文字反向，需将

MIRRTEXT 系统变量设置为 **0**。

以下列表描述了一些用于编辑的 **Text** 对象的方法。对于完整的列表，请查看 **AutoCAD**

ActiveX 和 **VBA** 参考的 **Text** 对象文档。

ArrayPolar

创建圆周阵列。

ArrayRectangular

创建矩形阵列。

Copy

复制 **Text** 对象。

Erase

删除 **Text** 对象。

Mirror

镜像 Text 对象。

Move

移动 Text 对象。

Rotate

旋转 Text 对象。

使用多行文字

对于较长、较为复杂的内容，可用 **MTEXT** 创建多行文字。多行文字可布满指定宽度，同时还可以在垂直方向上无限延伸。可以设置多行文字对象中单个字或字符的格式。

多行文字是由任意数目的文字行或段落组成的，布满指定的宽度。与单行文字不同的是，在一个多行文字编辑任务中创建的所有文字行或段落都被当作同一个多行文字对象。可以移动、旋转、删除、复制、镜像、拉伸或比例缩放多行文字对象。

与单行文字相比，多行文字具有更多的编辑选项。例如可以将下划线、字体、颜色和高度的变化应用到段落中的单个字符、词语或词组。

本节内容：

创建多行文字

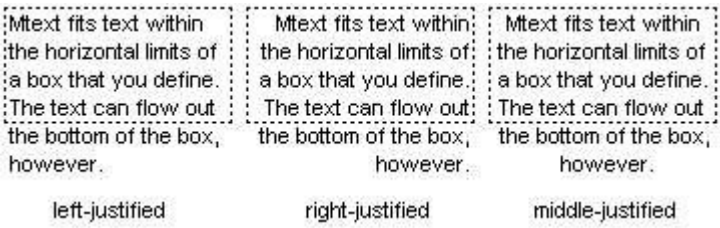
格式化多行文字

创建多行文字

可通过使用 **AddMText** 方法创建多行文字对象(**Mtext** 对象)。该对象需要三个值作为输入：文本字符串、文字放置于图形中的插入点和文字边界框的宽度。

文本字符串为实际显示的文字。可接受 **Unicode**、控制代码和特殊字符。插入点是放置文字的位置，它一个包含代表图形中三维 **WCS** 坐标的三个双精度变体数组。文字宽度是描述文字边界框宽度的正数。宽度以当前单位计算。

当 **Mtext** 对象创建后，可将文字高度、对齐方式、旋转角度和样式应用到 **Mtext** 对象上，或应用字符格式到选定的字符上。对齐方式要考虑文字边界以决定文字要插入的位置。



所创建的 **Mtext** 对象的高度与文字字符串中的字符数多少相关。

创建多行文字

以下代码在模型空间中的坐标点(2,2,0)创建一 **MText** 对象。

```
Sub Ch4_CreateMText()  
  
Dim mtextObj As AcadMText  
  
Dim insertPoint(0 To 2) As Double
```

```
Dim width As Double
```

```
Dim textString As String
```

```
insertPoint(0) = 2
```

```
insertPoint(1) = 2
```

```
insertPoint(2) = 0
```

```
width = 4
```

```
textString = "This is a text string  
for the mtext object."
```

```
' 在模型空间中创建文字对象
```

```
Set mtextObj = ThisDrawing.ModelSpace.
```

```
—
```

```
AddMText(insertPoint, width, textString)
```

```
ZoomAll
```

```
End Sub
```

格式化多行文字

新的文字将自动采用当前文字样式的特征。默认的文字样式为 **STANDARD**。可通过设置单独字符的格式和设置 **MText** 对象的属性来覆盖默认的文字样式。也可使用以下章节的所描述的方法来显示格式化的或特殊的字符。

诸如下划线、堆叠文字或字体等格式化选项或应用于在段落中的单独的单词或字符。诸如样式、对齐方式、宽度和旋转等定向选项将影响整个 **MText** 对象。可通过使用属性和关联到 **MText** 对象的方法来改变这两种格式。

格式化单独的单词或字符

可通过指定与文字格式代码相等效的 **ASCII** 来应用格式到单独的单词或字符。可为文字加下划线、上划线和创建堆叠文字。也可更改颜色、字体和文字高度。可更改文字字符间的间隔或缩小字符的宽度。要应用格式化，使用下表所列出的格式化代码：

多行文字格式化代码

格式化代码	用 途	输入内容...	显示内容...
\O...o	打开或关闭上划线	Autodesk \OAutoCAD\o 2000	Autodesk <u>AutoCAD</u> 2000
\L...l	打开或关闭下划线	Autodesk \OAutoCAD\l 2000	Autodesk <u>AutoCAD</u> 2000
\~	插入不断开空格	Autodesk AutoCAD\~2000	Autodesk AutoCAD 2000
\\	插入反斜杠	Autodesk \\AutoCAD	Autodesk \AutoCAD
\{...}	插入开始或结束大括号	Autodesk \{AutoCAD\} 2000	Autodesk {AutoCAD} 2000
\F 文件名	更改为指定的字体文件	Autodesk \Ftimes;AutoCAD 2000	Autodesk AutoCAD
\H 值;	按图形单位更改文字高度	Autodesk \H2;AutoCAD	Autodesk AutoCAD
\H 值 x; 倍数	更改文字高度为当前文字高度的 倍数	Autodesk AutoCAD \H3x;2000	Autodesk AutoCAD 2000

\S...^...;	堆叠在\u12289、#或^符号后的文字	1.000\S+0.010^-0.000;	
\T 值;	从 0.75 到 4 倍之间调整字符的间隔	\T2;Autodesk	
\Q 角度;	更改倾斜角度	\Q20;Autodesk	
\W 值;	更改宽度因子以产生较宽的文字	\W2;Autodesk	
\A 值;	设置对齐值; 有效值如下: 0(底对齐)、1(中间对齐)、2(顶对齐)	\A1;1\S1/2	
\P	换行	Autodesk \PAutoCAD 2000	

使用大括号可单独格式化括号内的内容。大括号可嵌套八层。

也可在一行中或段落中输入控制代码的 **ASCII** 等效值以显示格式化或特殊字符, 就如公差和标注符号。

以下控制字符可用于创建图中的文字内容。(该字串的 **ASCII** 等效值请查看之下的样例。)

```
{ {.5x; Big text} ; over text;/; under text}
```

大文字 上文字 / 下文字

使用控制字符以格式文字

本例创建一多行文字对象, 文字对象将格式化成上图所示。

Sub Ch4_FormatMText()

```
Dim mtextObj As AcadMText
```

```
Dim insertPoint(0 To 2) As Double
```

```
Dim width As Double
```

```
Dim textString As String
```

```
insertPoint(0) = 2
```

```
insertPoint(1) = 2
```

```
insertPoint(2) = 0
```

```
width = 4
```

```
' 定义控制字符的 ASCII 字符
```

```
Dim OB As Long ' 开始大括号 {
```

```
Dim CB As Long ' 结束大括号 }
```

```
Dim BS As Long ' 反斜杠 \par Dim FS As
```

```
Long ' 斜杠 /
```

```
Dim SC As Long ' 分号 ;
```

```
OB = Asc("{")
```

```
CB = Asc("}")
```

```
BS = Asc("\")
```



```
FS = Asc("/")
```

```
SC = Asc(";")
```

' 指定以下控制字符和文字字符的文本字符串:

```
' {{.5x; Big text}; over text;/; under  
text}
```

```
textString = Chr(OB) + Chr(OB) + Chr(BS)
```

```
+ "H1.5x" _
```

```
+ Chr(SC) + "Big text" +
```

```
Chr(CB) + Chr(BS) + "A2" _
```

```
+ Chr(SC) + "over text" +
```

```
Chr(BS) + "A1" + Chr(SC) _
```

```
+ Chr(FS) + Chr(BS) + "A0"
```

```
+ Chr(SC) + "under text" _
```

```
+ Chr(CB)
```

' 在模型空间中创建文字对象

```
Set mtextObj = ThisDrawing.ModelSpace.
```

```
_
```

```
AddMText(insertPoint, width, textString)
```

ZoomAll

End Sub

格式化多行文字对象

可设置控制样式、文字对齐方式和文字边框的大小及旋转的 **MText** 对象属性。这些设置影响到整个文字边框内，而不是某些单词或字符。

StyleName 属性设定默认的字体和格式化新建文字的特征。当创建文字后，可从现存样式列表中选择要用的样式。

当更改了部分字符中已被局部格式化过 **MText** 对象的样式时，样式将应用到整个对象中，而不保留字符的任何格式化。举个例子，将 **TrueType** 样式更改为使用 **SHX** 字体或其它 **TrueType** 字体的样式将导致整个对象都使用新的字体，任何字符的格式化也随之丢失。

对齐方式控制着文字的排列和基于指定对齐点的流向。文字的对齐是与左边界和右边界相关。而从段落的中间、顶部或底部流出(即开始填充文字)是与上边界和下边界相关。上下边是基于多行文字对象的最上行的最下行。**AutoCAD** 提供 9 种对齐的设置：左上(TL)、中上(TC)、右上(TR)、左中(ML)、正中(MC)、右中(MR)、左下(BL)、中下(BC)和右下(BR)。





使用 AttachmentPoint 属性可更改 MText 的对齐方式。

Rotation 属性控制着文字边界的旋转角度。

使用 Unicode 字符、控制代码和特殊字符

可在文本字符串中使用 Unicode 字符、控制代码和特殊字符以显示符号。(所有非文字字符必须用其 ASCII 等效值输入。)

Unicode 字符描述

Unicode 字符	描述
\U+00B0	角度符号
\U+00B1	正负差符号
\U+2205	直径标注符号

除了使用 Unicode 字符显示特殊字符外，也可通过在文本字符串中包括控制信息来指定特殊字符。使用一对百分比号(%%)以引入每一控制序列。

该控制代码使用于标准 AutoCAD 字体和 PostScript 字体：

%%nnn

在 VB 或 VBA 字符串中，以上例子要按以下输入

```
Dim percent as Long

percent = ASC("%")

TextString = chr(percent) + chr(percent)

+ "nnn"
```

这些控制代码只用于标准 AutoCAD 字体：

控制代码描述

控制代码	描述
%%o	切换上划线模式的开关
%%u	切换下划线模式的开关
%%d	绘制角度符号
%%p	绘制正负差符号
%%c	绘制直径标注符号
%%%	绘制单个百分比号

替换字体

当 AutoCAD 不能找到图形中指定的字体时，可以指定字体来替换其他字体或作为默认字体。

图形中的文字所使用的字体由文字样式决定，对于多行文字，由应用到各个文字的字体格式分别决定。在某些情况下，用户可能要确保图形只使用某一种特定的字体，或者要将所使用的当前字体转换为其他字体。为达到这一目的，可以使用任意文字编辑器来创建字体映射表。

可以使用字体映射表实施一致的字体标准，或者用于脱机打印。例如，如果和其他人共享图形，当遇到用其他字体创建的文字对象时，可用字体映射表指定

AutoCAD 使用的替换字体。同样，使用绘制速度较快的 **SHX** 字体编辑图形，并在最终打印时切换到比较复杂的字体。为此，可以设置一个字体映射表将每个 **SHX** 字体转换为对应的字体。

字体映射表是纯 **ASCII** 文本 (**FMP**) 文件，每一行包含一个字体映射，由字体文件的基本名（不含目录名或路径）、分号

(;) 及其替换字体文件的名称构成。替换字体文件名包含如 **.ttf** 这样的扩展名。

例如，可以在字体映射表中输入下列内容指定用 **times.ttf** TrueType

字体文件替代 **romanc.shx** 字体文件。

```
romanc.shx; times.ttf
```

AutoCAD 在 **Support** 文件夹下的 **acad.fmp** 文件中提供了缺省的字体映射表。可以使用任意

ASCII 文字编辑器编辑该文件。也可以通过使用在 **Preferences** 对象中的 **FontFileMap** 属性来指定不同的字体映射表文件。

本节内容：

指定替换默认字体

指定替换默认字体

如果图形中包含了一个在当前系统中不可用的字体，AutoCAD 自动使用替换字体取代该字体。缺省情况下，AutoCAD 使用 simplex.shx 文件。然而，可以根据需要指定另一个字体。通过在 Preferences 对象上提供的 AltFont File 属性可设置替换字体文件名。

如果文字样式中使用的是大字体，可以用 AltFontFile 属性将它映射为另一种字体。此属性使用配套的缺省字体文件：txt.shx 和 bigfont.shx。

AutoCAD 使用的字体替换规则如下表所示：

字体替换规则

文件扩展名	第一映射次序	第二映射次序	第三映射次序	第四映射次序
TTF	使用 FONTMAP 值	使用文字样式中定义的字体	Windows	替换类似的字体
SHX	使用 FONTMAP 值	使用文字样式中定义的字体	使用 FONTALT	提示输入新字体
PFB	使用 FONTMAP 值	使用 FONTALT	提示输入新字体	

拼写检查

在拼写检查过程中，AutoCAD 要使图形中的字与当前主词典中的字相匹配。新添的字存储在执行拼写检查时的当前自定义词典里。例如，可以添加正确的人名或名称以使 AutoCAD 不再把它们作为拼错的字来标识。

要检查其他语言的拼写时可切换至另一个主词典。

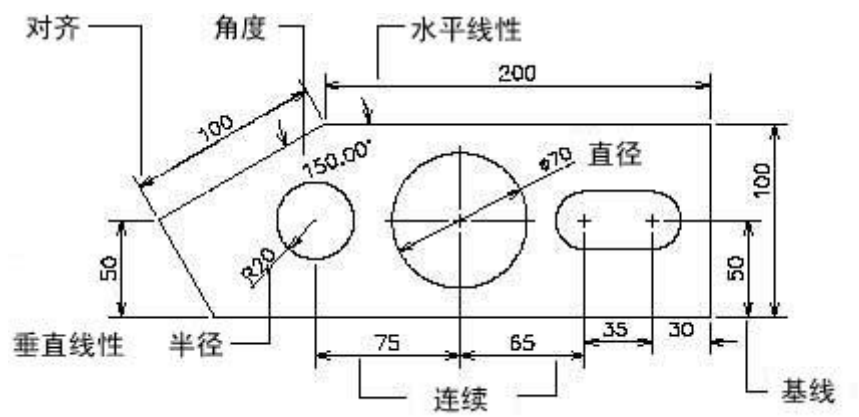
在 AutoCAD ActiveX 自动操作中未提供拼写检查的方法。然而，可使用 Preferences 对象上的 MainDictionary 属性指定不同的主字典，或使用 CustomDictionary 属性指定不同的自定义字典。

第五章 标注与公差

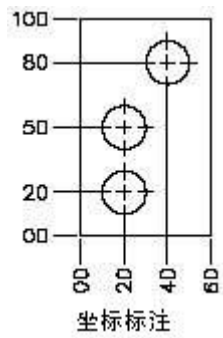
标注就是增加尺寸到图形中。公差指的是标注的允许偏差。使用 **ActiveX** 自动操作，标注可通过标注样式和替代来管理。

标注的概念

标注显示了对象的几何尺寸，对象之间的距离和角度，或一个特征的 X、Y 坐标。**AutoCAD** 提供了三种基本类型的标注：线性、径向和角度。线性标注包括对齐、旋转和坐标标注。每种标注的一个简单的例子在下图列出：



可为直线、复线、圆弧、圆和多义线创建标注，或可创建单独的标注。



AutoCAD 在当前图层中绘制标注。每一标注都关联着标注样式，这个标注样式可以是默认的或自定义的。该样式控制着标注的特性，如颜色、文字样式和线型比例等。它不支持厚度信息。样式族允许对不同类型的标注在基础样式上进行细微的修改。替代允许对指定标注的样式进行修改。

本节内容：

标注的元素

字义标注系统变量

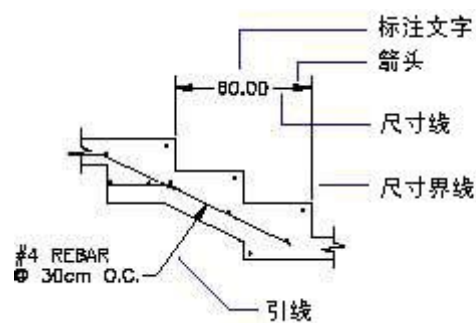
设置标注文字样式

理解引线

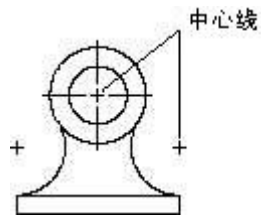
理解关联标注

标注的元素

本节简要介绍标注的各部分元素。



尺寸线是显示尺寸的方向和范围。如果为角度标注，尺寸线为一个圆弧。尺寸界线，也称为投影线，是从所标注的特征延伸到尺寸线上。箭头，也称为终止符，它是添加到每一尺寸线的末端。标注文字为一个文本字符串，通常是显示尺寸的实际距离。文字也可包括前缀、后缀和公差。引线是通过一条实线将一些注解引到引用的特征上。中心标记是在圆或圆弧上一个小的十字交叉。中心线是标记圆或圆弧的中心的虚线。



字义标注系统变量

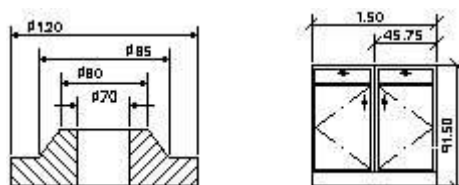
标注的系统变量控制着标注的外观。这些标注系统变量包括：DIMAUNIT、DIMUPT、DIMTOFL、DIMFIT、DIMITI、DIMTOH、DIMJUST 和 DIMITAD。可通过使用 `SetVariable` 方法设置这些变量。例如，以下代码行设置 DIMAUNIT 系统变量(角度标注的单位格式)为弧度(3)：

```
ThisDrawing.SetVariable "DIMAUNIT",
```

```
3
```

设置标注文字样式

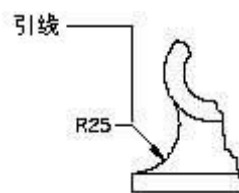
标注文字指的是关联到标注上的任意文字，包括尺寸、公差(允差和形位)、前缀、后缀和通过单行或段落形式的原文注释。可使用默认的由 AutoCAD 所计算出来的尺寸文字，也可提供自己的文字或者连文字都不要。可使用标注文字添加信息，例如特殊制造过程或装配说明。



单行标注文字所使用的现行文字样式可通过 `ActiveTextStyle` 属性指定。段落文字使用的现行文字样式可在文本字符串中进行改动。

理解引线

默认的引线是一根带有指向图形中某一特征的箭头的直的线。在这种情况下下的注解指的是段落文字、图块或特征控制框。这样的引线 with **AutoCAD** 为半径、直径和线性标注所创建的引线是不同的，它的文字不是在尺寸界线之间。



引线是与注解关联着，所以当编辑注解时，引线将因此而更新。可将注解复制到图形的其它地方并将其头一回到引线上，或可创建新的注解。也可以创建不带注解的引线。

理解关联标注

关联标注是将标注中所有的线、箭头、圆弧和文字作为单独的标注对象绘制。**DIMASO** 系统变量控制着关联标注，默认时为开。如果 **DIMASO** 为关时，尺寸线、尺寸界线、箭头、引线和标注文字将作为分离的对象绘制。如果需要将标注进行改变而不需要通过变量进行控制时，可创建非关联标注。一般来说，关联标注更容易维护，因为它们可以作为单一的对象进行修改。

设置和查询系统变量，可使用 **SetVariable** 和 **GetVariable** 方法。

创建标注

可创建线性、径向、角度和坐标标注。

当创建标注时，将使用现行的标注样式。标注创建后，可修改尺寸界线的起点、尺寸文字的位置和尺寸文字的内容和其相对于尺寸线的角度。也可以更改用于该标注的标注样式。

本节内容：

创建线性标注

创建径向标注

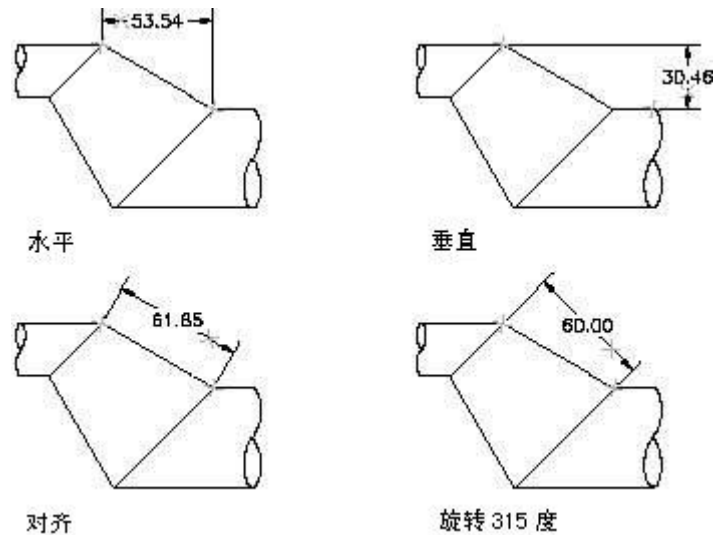
创建角度标注

创建坐标标注

创建线性标注

线性标注可以是对齐或旋转。对齐标注指的是尺寸线与尺寸界线的起点所在位置平等。旋转标注指的是尺寸线尺寸线与尺寸界线起点形成一个角度。

创建线性标注，使用 **AddDimAligned**、**AddDimRotated** 或 **AddDim3PointAligned** 方法。在创建了线性标注后，可修改其文字、文字的角度或尺寸线的角度。在以下的例图中，尺寸界线起点尺寸有具体的标明。结果的尺寸线位置也显示出来：



创建对齐标注，使用 **AddDimAligned** 方法。该方法要求三个坐标作为输入：两个尺寸界线起点和文字位置。

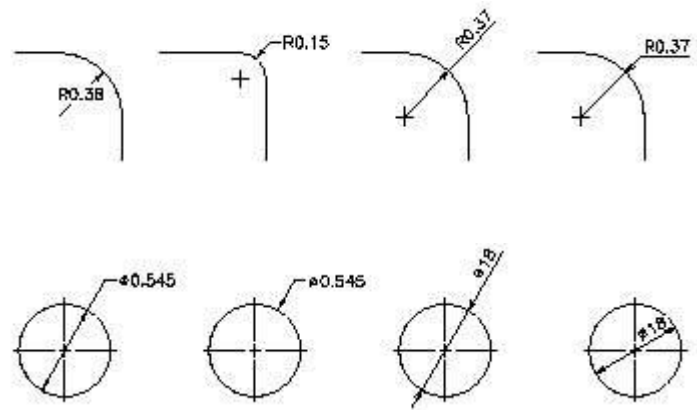
创建旋转标注，使用 **AddDimRotated** 方法。该方法要求三个坐标和尺寸线的角度作为输入。三个坐标分别为两个尺寸界线的起点和文字位置。角度必须提供弧度，并且是描述了尺寸线的旋转角度。

创建径向标注

径向标注是测量圆弧和圆的半径和直径。创建径向标注，使用 **AddDimRadial** 方法。

不同类型的径向标注的创建是依赖于圆或圆弧的大小、**TextPosition** 属性和在 **DIMUPT**、**DIMTOFL**、**DIMFIT**、**DIMTIH**、**DIMTOH**、**DIMJUST** 和 **DIMTAD** 标注系统变量的值。(系统变量的查询和设置使用 **GetVariable** 和 **SetVariable** 方法。)

对于水平标注文字，如果尺寸线的角度大于水平方向的 **15** 度，而且标注是在圆或圆弧的外侧时，**AutoCAD** 将绘制一条折向线，也称为平线或偏向线。该折向线有一个箭头长，并且另一侧放置着标注文字，由下图可以看到：



创建径向标注，使用 **AddDimRadial** 或 **AddDimDiametric** 方法。这些方法要求三个值作为输入：圆或圆弧的圆心坐标、引线附加点的坐标和引线的长度。

这些方法使用 **LeaderLength** 参数作为从 **ChordPoint**(弦点)到多段线的转折点(如无多段线段则到终止点)的距离。

创建径向标注

本例在模型空间中创建一径向标注

```
Sub Ch5_CreateRadialDimension()
```

```
Dim dimObj As AcadDimRadial
```

```
Dim center(0 To 2) As Double
```

```
Dim chordPoint(0 To 2) As Double
```

```
Dim leaderLen As Integer
```

```
' 定义标注
```

```
center(0) = 0
```

```
center(1) = 0
```

```
center(2) = 0
```

```
chordPoint(0) = 5
```

```
chordPoint(1) = 5
```

```
chordPoint(2) = 0
```

```
leaderLen = 5
```

```
' 在模型空间中创建径向标注
```

```
Set dimObj = ThisDrawing.ModelSpace. _
```

```
AddDimRadial(center, chordPoint, leaderLen)
```

```
ZoomAll
```

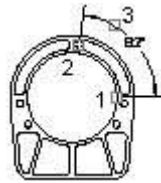
```
End Sub
```

注意 **LeaderLength** 设置只在创建标注时使用。当标注被保存后，修改 **LeaderLength** 值将不会影响标注的显示。

LeaderLength 属性不用于计算引线的位置。该值不被保存，因此在读取该属性时经常会返回 0。

创建角度标注

角度标注是测量两条线或三个点之间的角度。例如，你可以用它来测量圆的两个半径之间的角度。尺寸线显示为圆弧。



要创建角度标注，使用 **AddDimAngular** 方法。该方法需要三个值作为输入：角度顶点、尺寸界线起点和文字位置。**AngleVertex** 是圆或圆弧的中心，或被标注的两个线的公共顶点。尺寸界线起点为两条尺寸界线通过的点。

AngleVertex 可以和一个起点相同。如果你需要尺寸界线，它们将会自动添加。

创建一个角度标注

该例在模型空间中创建一个角度标注。

```
Sub Ch5_CreateAngularDimension()
```

```
Dim dimObj As AcadDimAngular
```

```
Dim angVert(0 To 2) As Double
```

```
Dim FirstPoint(0 To 2) As Double
```

Dim SecondPoint(0 To 2) As Double

Dim TextPoint(0 To 2) As Double

' 定义标注

angVert(0) = 0

angVert(1) = 5

angVert(2) = 0

FirstPoint(0) = 1

FirstPoint(1) = 7

FirstPoint(2) = 0

SecondPoint(0) = 1

SecondPoint(1) = 3

SecondPoint(2) = 0

TextPoint(0) = 3

TextPoint(1) = 5

TextPoint(2) = 0

' 在模型空间中创建角度标注

```
Set dimObj = ThisDrawing.ModelSpace. _
```

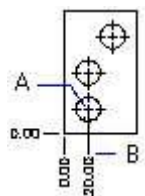
```
AddDimAngular(angVert, FirstPoint, SecondPoint, TextPoint)
```

```
ZoomAll
```

```
End Sub
```

创建坐标标注

坐标，或基准，标注测量从称为基准的原点的垂直距离到标注的特征上，例如一个零件中的孔。这些标注通过基准来维持特征精确的偏移量以防止了累积出错。



坐标标注由带引线的 X 或 Y 坐标组成。X-基准坐标标注测量由 X 轴的基准取特征的距离。Y-基准坐标标注测量由 Y 轴的同样距离。AutoCAD 使用当前 UCS 的原点以确定测量坐标。它使用坐标的绝对值。

不管在当前标注样式中文字方向是怎样定义的，文字始终与坐标引线对齐。可接受默认的文字或输入你自己的文字。

要创建坐标标注，使用 **AddDimOrdinate** 方法。该方法需要输入三个值：一个指定要标注的点(A)的坐标、一个指定引线终点(B)的坐标和一个指定标注是 X 基准坐标标注还是 Y-基准坐标标注的布尔值标记。如果输

入布尔值标记为 **TRUE**，则该方法创建一个 **X**-基准坐标标注。如果输入 **FALSE**，它将创建一个 **Y**-基准坐标标注。

创建坐标标注

该例在模型空间中创建一个坐标标注。

```
Sub Ch5_CreatingOrdinateDimension()
```

```
Dim dimObj As AcadDimOrdinate
```

```
Dim definingPoint(0 To 2) As Double
```

```
Dim leaderEndPoint(0 To 2) As Double
```

```
Dim useXAxis As Long
```

```
' 定义标注
```

```
definingPoint(0) = 5
```

```
definingPoint(1) = 5
```

```
definingPoint(2) = 0
```

```
leaderEndPoint(0) = 10
```

```
leaderEndPoint(1) = 5
```

```
leaderEndPoint(2) = 0
```

```
useXAxis = 5
```

```
' 在模型空间中创建坐标标注
```

```
Set dimObj = ThisDrawing.ModelSpace. _
```

```
AddDimOrdinate(definingPoint, _
```

```
leaderEndPoint, useXAxis)
```

```
ZoomAll
```

```
End Sub
```

编辑标注

和 AutoCAD 中的其它图形对象一样，可以用对象提供的标准方法和属性编辑标注。

以下属性对于大多数标注对象有效：

Rotation

为尺寸线指定旋转角度(弧度)

StyleName

指定标注样式的名称。

TextOverride

指定标注的文本字符串

TextPosition

指定标注文字的位置

TextRotation

指定标注文字的旋转角度

Measurement

指定标注的实际测量距离

另外，某些标注对象提供了编辑尺寸界线原点和引线长度的属性。

以下方法包含了对标注对象的编辑：

ArrayPolar

创建圆周阵列

ArrayRectangular

创建矩形阵列

Copy

复制标注对象。

Erase

删除标注对象。

Mirror

镜像标注对象。

Move

移动标注对象。

Rotate

旋转标注对象。

ScaleEntity

比例缩放标注对象。

替代标注文字

所显示的尺寸值可以使用 **TextOverride** 属性来替换。使用该方法可完全替换尺寸的显示值，或可附加文字到该值中。本例附加一些文字到值中以使得字符串和标注值都能显示出来。

```
Sub Ch5_OverrideDimensionText()
```

```
Dim dimObj As AcadDimAligned
```

```
Dim point1(0 To 2) As Double
```

```
Dim point2(0 To 2) As Double
```

```
Dim location(0 To 2) As Double
```

' 定义标注

```
point1(0) = 5#: point1(1) = 3#: point1(2) = 0#
```

```
point2(0) = 10#: point2(1) = 3#: point2(2) = 0#
```

```
location(0) = 7.5: location(1) = 5#: location(2) = 0#
```

' 在模型空间创建一对齐标注对象

```
Set dimObj = ThisDrawing.ModelSpace. _
```

```
AddDimAligned(point1, point2, location)
```

' 更改标注的文本字符中

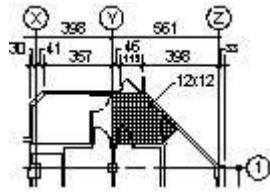
```
dimObj.TextOverride = "该值为 <>"
```

```
dimObj.Update
```

```
End Sub
```

利用标注样式

一个命名的标注样式是一个确定标注外观的设置组合。使用命名的标注样式，可建立和执行图形设计标准。



所有标注是使用当前的标注样式来创建的。如果在创建标注前未定义或应用某个标注样式，AutoCAD 将使用默认的样式：**STANDARD**。设定当前的标注样式，使用 **ActiveDimStyle** 属性。

要建立一个父标注样式，可通过命名和保存样式开始。新的样式是基于当前样式建立的，它包括了所有后来更改到布局的标注部件(**DDIM** 几何对话框)，在这里注解是指主单位和替换单位、公差和文字。

要创建一个新的标注样式，使用 **Add** 方法。该方法需要提供新的标注样式的名称。

AutoCAD ActiveX 自动操作允许增加新的标注样式和更改当前的标注样式。也可通过 **StyleName** 属性更改关于到给定标注的标注样式。

也可以复制现有的样式或设置替代。使用 **CopyFrom** 方法以从一个源对象中复制标注样式到新的标注样式。这个源对象可以是另外的 **DimStyle** 对象，一个标注、公差或引线对象，或者甚至是一个 **Document** 对象。如果从其它标注样式、公差或引线对象的设置中复制样式，当前的设置，包括任何对象，将复制到新的样式中。如果从 **Document** 对象中复制样式，当前标注样式，还有任何的图形替代，将复制到新的样式中。

复制标注样式和替代

本例创建三个新的标注样式并分别从文档中、给定的标注样式中及给定的标注中复制当前设置到每一新的标注样式。在运行本例前通过以下的适当调整，你将会找到所创建的标注样式的不同。

- 1.创建一新的图形，并激活该图形。
- 2.在新的图形中创建一新的线性标注。该标注是图形中唯一的对象。
- 3.使用 **OPM**，更改尺寸线的颜色为黄色。
- 4.更改 **DIMCLRD** 系统变量值为 5(蓝色)。

5.运行以下例子:

```
Sub Ch5_CopyDimStyles()
```

```
Dim newStyle1 As AcadDimStyle
```

```
Dim newStyle2 As AcadDimStyle
```

```
Dim newStyle3 As AcadDimStyle
```

```
Set newStyle1 = ThisDrawing.DimStyles.Add _
```

```
("Style 1 copied from a dim")
```

```
Call newStyle1.CopyFrom(ThisDrawing.ModelSpace(0))
```

```
Set newStyle2 = ThisDrawing.DimStyles.Add _
```

```
("Style 2 copied from Style 1")
```

```
Call newStyle2.CopyFrom(ThisDrawing.DimStyles.Item _
```

```
("Style 1 copied from a dim"))
```

```
Set newStyle2 = ThisDrawing.DimStyles.Add _
```

```
("Style 3 copied from the running drawing values")
```


Call newStyle2.CopyFrom(ThisDrawing)

End Sub

打开 **DIMSTYLE** 对话框。现存可以看到有 3 个标注样式的列表。样式 1 具有黄色的尺寸线。样式 2 与样式 1 完成一样。样式 3 具有蓝色的尺寸线。

本节内容：

替代标注样式

替代标注样式

每一标注都可能接受替代标注样式中的设置。台下属性对大部分标注对象有效：

AltRoundDistance

指定替代单位的圆整。

AngleFormat

指定角度标注的单位格式。

Arrowhead1Block, Arrowhead2Block

指定尺寸线的自定义箭头的图块。

Arrowhead1Type, Arrowhead2Type

指定尺寸线的箭头类型。

ArrowheadSize

指定尺寸线箭头、引线箭头和转多段线的大小。

CenterMarkSize

指定半径和直径标注的中心标记大小。

CenterType

指定半径和直径标注的中心标记类型。

DecimalSeparator

指定在十进制标注和公差值中用于小数分隔符的字符。

DimensionLineColor

指定标注、引线 and 公差对象的尺寸线颜色。

DimensionLineWeight

指定尺寸线的线宽。

DimLine1Suppress, DimLine2Suppress

指定尺寸线的隐藏。

DimLineInside

指定只在尺寸界线内显示尺寸线。

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

指定尺寸界线到定义该标注的原点的偏移距离。

ExtensionLineWeight

设置尺寸界线的线宽。

ExtLine1EndPoint, ExtLine2EndPoint

指定尺寸界线的终点。

ExtLine1StartPoint, ExtLine2StartPoint

指定尺寸界线的起点。

ExtLine1Suppress, ExtLine2Suppress

指定尺寸界线的隐藏。

Fit

指定文字和箭头在尺寸界线内侧或外侧的放置位置。

ForceLineInside

指定在文字置于尺寸界线外时是否绘制尺寸界线之间的尺寸线。

FractionFormat

指定在标注及公差中小数值的格式。

HorizontalTextPosition

指定标注文字的水平对齐。

LinearScaleFactor

指定线性标注测量值的全局比例因子。

PrimaryUnitsPrecision

指定标注或公差的主单位所显示的小数位数。

SuppressLeadingZeros, SuppressTrailingZeros

指定消除标注值的前导或后继零。

SuppressZeroFeet, SuppressZeroInches

指定消除标注值中的零英尺和零英寸。

TextColor

指定标注和公差对象的文字颜色。

TextGap

指定当断开尺寸线放置标注文字时标注文字和尺寸线之间的距离。

TextHeight

指定标注或公差文字的高度。

TextInside

指定是否标注文字绘制于尺寸界线的内侧。

TextInsideAlign

指定除坐标标注外的其它所有标注类型中标注文字在尺寸界线内的位置。

TextMovement

指定当移动文字时标注文字是怎样绘制的。

TextOutsideAlign

指定除坐标标注外的其它所有标注类型中标注文字在尺寸界线外的位置。

TextPosition

指定标注文字的位置。

TextPrecision

指定角度标注文字的精度。

TextPrefix

指定标注值的前缀。

TextRotation

指定标注文字的旋转角度。

TextSuffix

指定标注值的后缀。

ToleranceDisplay

指定是否显示标注文字的公差。

ToleranceHeightScale

指定相对于标注文字高度的公差值高度的比例因子。

ToleranceJustification

指定相对于名义标注文字的公差值的垂直对齐。

ToleranceLowerLimit

指定标注文字的最小公差界限。

TolerancePrecision

指定在主标注中公差值的精度。

ToleranceSuppressLeadingZeros

指定消除公差值中的前导零。

ToleranceSuppressTrailingZeros

指定消除公差值中的后继零。

ToleranceUpperLimit

指定标注文字的最大公差界限。

UnitsFormat

指定除角度标注外的所有标注的单位格式。

VerticalTextPosition

指定相对尺寸线的文字垂直位置。

为对齐标注输入用户定义后缀

本例在模型空间中创建一对齐标注并使用 **TextSuffix** 属性允许用户更改标注的文字后缀。

```
Sub Ch5_AddTextSuffix()
```

```
Dim dimObj As AcadDimAligned
```

```
Dim point1(0 To 2) As Double
```

```
Dim point2(0 To 2) As Double
```

```
Dim location(0 To 2) As Double
```

```
Dim suffix As String
```

```
' 定义标注
```

```
point1(0) = 0: point1(1) = 5: point1(2) = 0
```

```
point2(0) = 5: point2(1) = 5: point2(2) = 0
```

```
location(0) = 5: location(1) = 7: location(2) = 0
```

```
' 在模型空间中创建对齐标注对象
```

```
Set dimObj = ThisDrawing.ModelSpace. _
```

```
AddDimAligned(point1, point2, location)
```

```
ThisDrawing.Application.ZoomAll
```

' 允许用户更改标注的文字后缀

```
suffix = InputBox("为标注输入新的文字后缀" _
```

```
, "设置标注后缀", ":SUFFIX")
```

' 将更改应用到标注中

```
dimObj.TextSuffix = suffix
```

```
ThisDrawing.Regen acAllViewports
```

```
End Sub
```

在模型空间和图纸空间中标注

可以在图纸空间和模型空间中绘制标注。然而，如果你所标注要的几何体是在模型空间中，最好是在模型空间中绘制标注，因为 **AutoCAD** 会放置定义点到几何体绘制的地方。

如果你在图纸空间中绘制标注来描述模型空间中的几何体，当使用编辑全集或在模型空间视口中改动了显示倍率时图纸空间中的标注将不会随之更改。当将视图由图纸空间切换到模型空间时图纸空间中的标注将停留在原先的位置。

如果在图纸空间中标注并且线性标注的全局比例因子(**DIMLFAC** 系统变量)设置为小于 **0**，则测量距离是绝对值的 **DIMLFAC** 倍。如果在模型空间中标注，不论 **DIMLFAC** 小于 **0**，该值均为 **1.0**。如果在 **Dim** 提示下更改该变量并选择视口选项，则 **AutoCAD** 会为 **DIMLFAC** 计算一个值。**AutoCAD** 计算模型空间到图纸空间的比例并将该值作为负值填入 **DIMLFAC** 中。

创建引线及注解

引线是将一些注解连接到图形中的某一特征的线。引线与其注解是关联的，也就是说如果修改了注解，引线将因此而更新。不要将 **Leader** 对象与 AutoCAD 自动生成的作为尺寸线一部分的引线相混淆。

本节内容：

创建引线

添加注解到引线

关联引线

编辑关联引线

编辑引线

创建引线

可在图形中的任何点或特征中创建引线并控制其显示。引线可为直线段或光滑样条曲线。引线颜色由当前尺寸线颜色控制。引线是通过当前标注样式中的全局标注比例所控制。箭头的类型和大小，是由当前样式的第一个所定义。

称为钩线的小段线通常是连接注解到引线上。如果最后一段引线段与水平方向的角度大于 **15** 度则钩线将与多行文字和特征控制框显示在一起。钩线有半个箭头的长短。如果引线没有注解，则不出现钩线。



要创建引线，使用 **AddLeader** 方法。该方法需要输入三个值：创建引线所在的坐标数组、注解对象(如果引线无注解则用 **NULL**)，和所创建的引线类型。引线类型指的是引线是为直线或为光滑样条曲线。它同时也确定引线是否有箭头。使用以下常量中的一个来指定引线的类型：**acLineNoArrow**、**acLineWithArrow**、**acSplineNoArrow** 或 **acSplineWithArrow**。这些常量为相互独立的。

创建引线

本例在模型空间中创建一引线。该引线没有关联注解。

```
Sub Ch5_CreateLeader()
```

```
Dim leaderObj As AcadLeader
```

```
Dim points(0 To 8) As Double
```

```
Dim leaderType As Integer
```

```
Dim annotationObject As AcadObject
```

```
points(0) = 0: points(1) = 0: points(2) = 0
```

```
points(3) = 4: points(4) = 4: points(5) = 0
```

```
points(6) = 4: points(7) = 5: points(8) = 0
```

```
leaderType = acLineWithArrow
```

```
Set annotationObject = Nothing
```

```
' 在模型空间中创建引线对象
```

```
Set leaderObj = ThisDrawing.ModelSpace. _
```

AddLeader(points, annotationObject, leaderType)

ZoomAll

End Sub

添加注解到引线

引线注解可以是 **Tolerance**(公差)、**MText**(多行文字)或 **BlockRef**(图块参照)对象。可以创建一新的注解或复制添加现有的注解。注解只能在创建时添加。

要在引线创建时添加注解，可通过 **AddLeader** 方法输入注解。

关联引线

引线是与它们的注解相关联着，所以当注解移动时，引线的终点也会跟着移动。当你移动文字和特征控制框注解时，最后的引线段将根据注解和引线倒数第二点的相对位置来决定引线是最后一个线段在依附于注解的左侧还是右侧。如果注解的中点是在倒数第二个点的右侧，则引线依附于右侧；否则依附于左侧。

不管用 **Erase**、**Add**(添加到图块中)或 **Wblock** 方法删除图形的任一对象都将断开引线对象的关联性。如果引线和其注解是通过单一步骤一起复制，则新复制的对象也相关联。如果它们是分别被复制，则不会关联。不管是什么原因断开了其关联性，例如，通过单独复制引线对象或删除注解，转多段线将从引线上被移去。

关联引线到注解上

本例创建一多行文字对象。然后创建一引线对象并使用多行文字对象作为注解。

```
Sub Ch5_AddAnnotation()
```

```
Dim leaderObj As AcadLeader
```

```
Dim mtextObj As AcadMText
```

```
Dim points(0 To 8) As Double
```

```
Dim insertionPoint(0 To 2) As Double
```

```
Dim width As Double
```

```
Dim leaderType As Integer
```

```
Dim annotationObject As Object
```

```
Dim textString As String, msg As String
```

```
' 在模型空间中创建多行文字对象
```

```
textString = "Hello, World."
```

```
insertionPoint(0) = 5
```

```
insertionPoint(1) = 5
```

```
insertionPoint(2) = 0
```

```
width = 2
```

```
Set mtextObj = ThisDrawing.ModelSpace. _
```

```
AddMText(insertionPoint, width, textString)
```

```
' 引线数据
```

```
points(0) = 0: points(1) = 0: points(2) = 0
```

```
points(3) = 4: points(4) = 4: points(5) = 0
```

```
points(6) = 4: points(7) = 5: points(8) = 0
```

```
leaderType = acLineWithArrow
```

```
' 在模型空间中创建引线对象并关联多行文字对引线上
```

```
Set annotationObject = mtextObj
```

```
Set leaderObj = ThisDrawing.ModelSpace. _
```

```
AddLeader(points, annotationObject, leaderType)
```

```
ZoomAll
```

```
End Sub
```

编辑关联引线

除了引线和注解的关联关系以外，引线和其注解在图形中是完全独立的对象。编辑引线不会影响注解，同时编辑注解也不会影响引线。

尽管文字注解是使用 **DIMCLRT**、**DIMTXT** 和 **DIMTXSTY** 系统变量所定义的颜色、高度和样式来创建的，但它不能通过这些系统变量来修改，因为它不是真正的标注对象。文字注解必须其它多行文字对象一样的方法来编辑。

使用 **Evaluate** 方法来求得引线和其关联着的注解的关系。该方法将在需要时更新引线几何图案。

编辑引线

对引线注解的任何改动所造成其位置的改变将影响其相关联的引线的终点位置。同样道理，旋转注解也将造成引线的钩线(如果有)跟着旋转。

重新改变引线的大小，可以调整其比例。比例缩放只缩放所选定的对象。例如，如果你调整了引线的比例，注解还是停留在原先的位置。

另外，你也可移动、镜像和旋转引线。使用 **ScaleEntity**、**Move**、**Mirror** 和 **Rotate** 方法来编辑引线。也可用 **StyleName** 属性来修改关联于注解的文字样式。

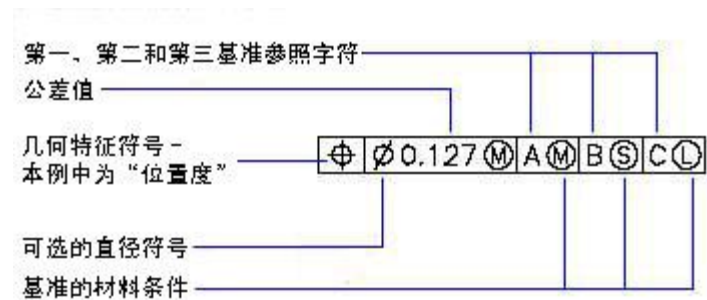
创建形位公差

形位公差显示特征的形状、轮廓、方向、位置和跳动的偏差。可以向特征控制框中添加形位公差。这些框中包含单个标注的所有公差信息。

要创建形位公差使用 **AddTolerance** 方法。该方法需要输入三个值：包含公差符号的文本字符串、在图形中放置公差的位置和指定公差方向的方向矢量。

也可以复制、移动、删除、比例缩放和旋转公差。

特征控制框由至少两个框格组成。第一个框格包含一个几何特征符号，表示所用公差的几何特征，例如形状、方向或跳动。形状公差控制直线度、平面度、圆度、圆柱度，以及直线和表面的轮廓度。在图例中，特性就是位置。



第二个框格包含公差值。在公差值前有一个指定符号，公差值后又一个包容条件符号。

创建形位公差

本例在模型空间中创建一简单的形位公差。

```
Sub Ch5_CreateTolerance()
```

```
Dim toleranceObj As AcadTolerance
```

```
Dim textString As String
```

```
Dim insertionPoint(0 To 2) As Double
```

```
Dim direction(0 To 2) As Double
```

```
' 定义公差对象
```

```
textString = "这里是特征控制框"
```

```
insertionPoint(0) = 5
```

```
insertionPoint(1) = 5
```

```
insertionPoint(2) = 0
```

```
direction(0) = 1
```

```
direction(1) = 1
```

```
direction(2) = 0
```

```
' 在模型空间中创建公差对象
```

```
Set toleranceObj = ThisDrawing.ModelSpace. _
```

```
AddTolerance(textString, insertionPoint, direction)
```

```
ZoomAll
```

```
End Sub
```


本节内容：

编辑公差

编辑公差

公差受以下几个系统变量影响：**DIMCLRD** 控制特征控制框的颜色；**DIMCLRT** 控制公差文字的颜色；**DIMGAP** 控制特征控制框与文字之间的间隙；**DIMTXT** 控制公差文字的大小；**DIMTXTSTY** 控制公差文字的样式。使用 **SetVariable** 方法可设置系统变量的值。

第六章 定义菜单和工具栏

AutoCAD ActiveX 自动操作为在 AutoCAD 进程中自定义菜单和工具栏提供了很大的方便。

使用 AutoCAD ActiveX/VBA，可以编辑或引用现有的菜单结构，或可完全替换当前菜单结构。也可以处理工具栏和右键菜单。

菜单的自定义是通过将应用程序中的任务罗列出来或通过多个步骤的任务压缩到一个单一的菜单项中来提高生产力。

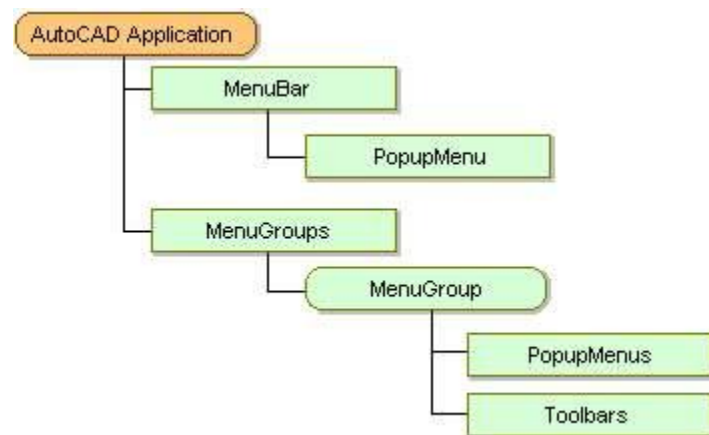
理解 MenuBar 和 MenuGroups 集合

AutoCAD ActiveX 提供多种菜单相关对象。其中两个最主要的是 MenuBar 集合和 MenuGroups 集合。

MenuBar 集合包含显示在 AutoCAD 菜单栏的所有菜单。

MenuGroups 集合包含在当前 AutoCAD 进程中加载的菜单组。这些菜单组包含在 AutoCAD 进程中启用的所有菜单，部分或全部的菜单可能会显示在 AutoCAD 菜单栏上。另外，菜单组也包含在当前 AutoCAD 进程中启用的所有工具栏。

菜单组也可以表现为平铺菜单、屏幕菜单或数字化仪菜单。



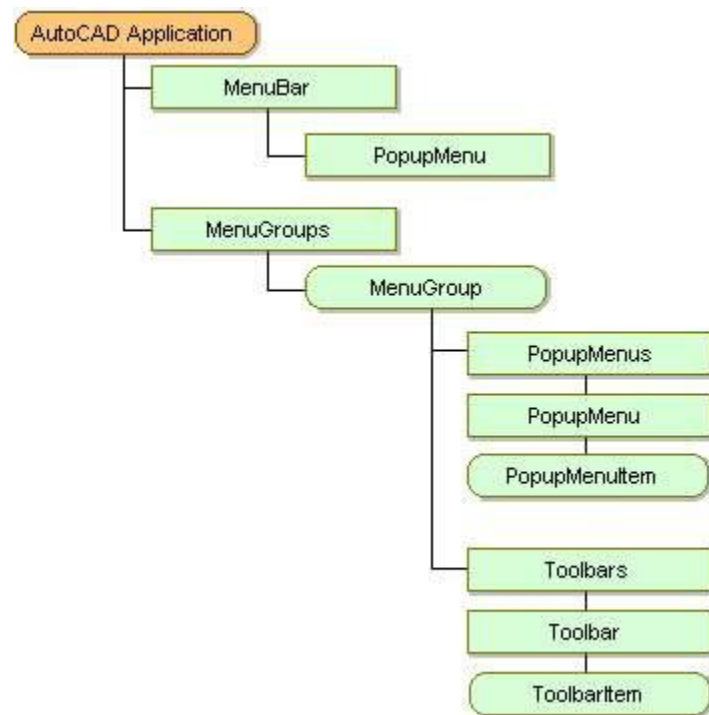
本节内容：

研究 MenuGroups 集合

研究 MenuGroups 集合

MenuGroups 集合包含在当前 AutoCAD 进程中加载的菜单组。每一菜单组包含一个 PopupMenus 集合和一个 Toolbars 集合。PopupMenu 集合包含所有在菜单组中的菜单。同样，Toolbars 集合包含所有在菜单组中的工具栏。

每一 PopupMenu 实际上是包含了显示于菜单上的每一菜单项的单独对象的集合。同样，每一 Toolbar 也是包含了显示于工具栏上的每一工具栏项的单独对象的集合。



加载菜单组

菜单组是使用 Load 方法加载到 AutoCAD 的。当使用 Load 方法时，设定 BaseMenu 参数为 TRUE 时则加载一新的菜单组到菜单栏上。这将把菜单组作为基本菜单加载，就象 AutoCAD 的 MENU 命令一样。

要将新的菜单组作为局部菜单加载，可忽略 BaseMenu 参数。这个方法加载的菜单组与 AutoCAD 的 MENU LOAD 命令的方式一样。当加载到 MenuGroups 集合后，局部菜单可通过使用 InsertMenuInMenuBar 方法或 InsertInMenuBar 方法插入到菜单栏中。

当菜单组加载后，通过该菜单组定义的所有菜单和工具栏都可以使用。可以

添加新的菜单到菜单栏

从菜单栏中移去菜单

重新排列菜单栏中的菜单

添加新的项目到现有的菜单或工具栏中

从现有的菜单或工具栏中移去项目

创建新的菜单和工具栏

浮动或泊留工具栏

启用或禁用菜单和工具栏项

选中或不选中菜单项

更改菜单或工具栏的名称标记、标签或帮助字符串

重新分配关联于菜单或工具栏项的宏

本例加载菜单文件 **acad.mnc**

ThisDrawing.Application.MenuGroups.Load

"acad.mnc"

注意：不能使用 **ActiveX** 自动操作来编辑图像平铺菜单项、屏幕菜单或数字化仪菜单。然而，可使用 **ActiveX** 自动操作来加载或卸载这些菜单类型。对于这些菜单类型的更多信息，请参见 **AutoCAD 自定义手册第四章"自定义菜单"**。

本节相关内容：

创建新的菜单组

创建新的菜单组

AutoCAD ActiveX 不允许通过程序创建新的(空)菜单组。然而，可加载现有的菜单组并使用新的名称另外保存菜单组到一个新的菜单文件。然后再编辑该菜单组以包含所需要的结构。这个创建新的菜单组的过程是基于现有菜单组具有自动提供基础菜单(例如"文件"、"窗口"、"帮助")的优势。

用新的文件名保存菜单组

以下示例如何在菜单组集合中将第一个菜单组用"MyMenu.mnc"名保存。

```
ThisDrawing.Application.MenuGroups.Item(0).
```

—

```
SaveAs "MyMenu.mnc", acMenuFileCompiled
```

改变菜单条

正如我们所见到的，如果一个新的菜单组被加载为基础菜单，它就完全可以取代菜单条。同时，在菜单条上的独立菜单也可以增加、删除或重排。

本节主题

在菜单条上插入菜单

在菜单条上删除菜单

在菜单条上重排菜单项

在菜单条上插入菜单

用 `InsertMenuInMenuBar` 或 `InsertInMenuBar` 方法可以在菜单条上增加一个现有的菜单。两种方法的结果是一样的。

两种方法的不同之处在于调用对象的位置不同。运用 `InsertMenuInMenuBar` 方法是从 `PopupMenu` 集合中调用对象。运用这种方法你可以将这个集合中的任意菜单插入到菜单条上的指定位置。这种方法要求输入被插入菜单的名称以及需要插入在菜单条上的位置。

运用 `InsertInMenuBar` 方法是直接调用 `PopupMenu` 对象进行插入。这种方法只要求输入需要插入在菜单条上的位置。由于是直接调用 `PopupMenu` 对象进行插入，所以菜单名不需要输入。

你可以选择任意一个你认为方便的方法来使用。

在菜单条上插入一个菜单

以下的例子创建了一个被称为 `TestMenu` 的新菜单并把它加入到了菜单条中，这个菜单项被赋值为 `OPEN` 命令。创建后这个命令就可以在菜单条上显示了。

```
Sub Ch6_InsertMenu()
```

' 为当前菜单组定义变量

```
Dim currMenuGroup As AcadMenuGroup
```

```
Set currMenuGroup = ThisDrawing.Application. _
```

```
MenuGroups.Item(0)
```

```
' 创建新的菜单
```

```
Dim newMenu As AcadPopupMenu
```

```
Set newMenu = currMenuGroup.Menus.Add("TestMenu")
```

```
' 为菜单项声明变量
```

```
Dim newItem As AcadPopupMenuItem
```

```
Dim openMacro As String
```

```
' 赋值宏字符串为 VB 语句"ESC ESC _open "并创建菜单项
```

```
openMacro = Chr(3) + Chr(3) + Chr(95) + "open"
```

```
+ Chr(32)
```

```
Set newItem = newMenu.AddMenuItem(newMenu.Count
```

```
+ 1, _
```

```
"Open", openMacro)
```

' 在菜单条上显示菜单

```
currMenuGroup.Menus.InsertMenuInMenuBar "TestMenu",
```

```
""
```

End Sub

从菜单条上删除菜单

用 `RemoveMenuFromMenuBar` 或 `RemoveFromMenuBar` 方法可以从菜单条上删除一个菜单。两种方法的结果是一样的。

两种方法的不同之处在于调用对象的位置不同。运用 `RemoveMenuFromMenuBar` 方法是从 `PopupMenu` 集合中调用对象。这种方法要求输入被删除菜单的名称或需要删除的菜单在菜单条上的位置。

运用 `RemoveFromMenuBar` 的方法是直接调用 `PopupMenu` 对象进行删除。这种方法什么都不需要输入。

由于是直接调用 `PopupMenu` 对象进行删除，所以菜单名不需要输入。

你可以选择任意一个你认为方便的方法去使用。

从菜单条上删除一个菜单

以下的例子将刚才加到菜单条中的 `TestMenu` 菜单从菜单条上删除。

' 从菜单条中删除菜单

```
currMenuGroup.Menus.RemoveMenuFromMenuBar ("TestMenu")
```

注意：从菜单条上删除的菜单仍然在其赋值的菜单组下存在。只是使用者暂时在菜单条上看不见而已。

在菜单条上重排菜单项

要在菜单条上重排菜单，可插入和删除菜单直到菜单达到合理的结构。

将第一个菜单移动到菜单条的尾部

以下的例子示范了如何将菜单条上的第一个菜单移动到菜单条的末尾。

```
Sub Ch6_MoveMenu()
```

```
' 定义变量以保存将被移动的菜单
```

```
Dim moveMenu As AcadPopupMenu
```

```
Dim MyMenuBar As AcadMenuBar
```

```
Set MyMenuBar = ThisDrawing.Application.menuBar
```

```
' 设置 t moveMenu 等于在菜单条上显示的第一个菜单
```

```
Set moveMenu = MyMenuBar.Item(0)
```

```
' 从菜单条中删除第一个菜单
```

```
MyMenuBar.Item(0).RemoveFromMenuBar
```

' 将菜单添加到菜单条上最后的位置

```
moveMenu.InsertInMenuBar (MyMenuBar.count)
```

End Sub

创建和编辑下拉菜单和快捷菜单

ACAD ActiveX/VBA 可以制定两种形式的菜单：下拉菜单和快捷菜单(有时称为光标菜单)。两种菜单都是层叠菜单。快捷菜单能够提供进入常用命令项(如对象捕捉)的快捷方式。

一个下拉菜单可以包括 999 个菜单项。一个快捷菜单可以包括 499 个菜单项。两个限量都包括了层级中的所有菜单。如果一个菜单中的菜单项超出了这个限制，ACAD 会忽略额外项目。如果一个下拉菜单或快捷菜单的长度超出图形屏幕外，它将被缩短到屏幕的长度。

下拉菜单即是从菜单条上向下拉的菜单，而快捷菜单总是出现在图形屏幕的十字光标处或附近。两种菜单的操作方法是相同的，只是快捷菜单的图标不出现在菜单条上。快捷菜单的图标是找不到的，要进入快捷菜单需通过一个基础菜单组中的单一菜单。具有 **ShortcutMenu** 属性的就是快捷菜单。如果在菜单组中的某一菜单的 **ShortcutMenu** 属性显示为 **TURE**，那么这个菜单就是快捷菜单了。

本节主题

建立新菜单

在菜单中增加新的菜单项

在菜单中增加分隔符

对菜单赋值一个加速键

建立一个层叠式子菜单

从菜单中删除一个菜单项

研究菜单项的属性

建立新菜单

为了增加一个新的菜单，可使用 **Add** 方法以在 **PopupMenu** 集合中添加一个新的 **PopupMenu** 对象。

如果要建立一个新的快捷菜单，你首先要删除一个现有的快捷菜单，因为每个菜单组中只能有一个快捷菜单。如果这个菜单组中没有快捷菜单，你可以用标签"POP0"来增加一个菜单。这就是告诉 **ACAD** 你想创建一个快捷菜单。

这种增加菜单的方法要求输入菜单的名称(或称为标签)。这个名称也就成为它将来在菜单条上的名称。你也可以通过这个名称在菜单集中方便地找到这个菜单。

菜单的名称可以是一个简单的字符串，也可以包括一些特殊的代码。要了解这些特殊代码的完整列表，你可以在"AutoCAD 自定义手册"的"下拉菜单和快捷菜单标签的语法"中查询。

在创立了菜单之后，你可以使用该菜单的 **Name** 属性来改变菜单的名称。

建立一个新的弹出菜单

以下示例在 **MenuGroups** 集合中第一个菜单组中增加一个名为"TestMenu"的弹出菜单。

```
Sub Ch6_CreateMenu()
```

```
Dim currMenuGroup As AcadMenuGroup
```

```
Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)
```

' 创建新的菜单

```
Dim newMenu As AcadPopupMenu
```

```
Set newMenu = currMenuGroup.Menus.Add("TestMenu")
```

```
End Sub
```

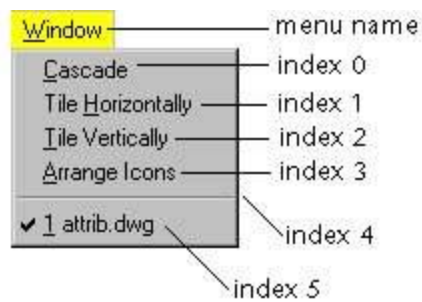
在菜单中增加新的菜单项

用 **AddMenuItem** 方法可以在菜单中增加新的菜单项。运用这种方法可以在指定的菜单中增加一个新的 **PopupMenuItem** 对象。

使用 **AddMenuItem** 方法需要输入四项参数：索引、标签、标记、宏。

索引

索引参数是一个整数，它指定了新的菜单项在菜单中的位置。索引是在菜单的标题后由位置零(0)做为初始位置的。要想在菜单的末尾新增一个菜单项，必须将索引参数设置得同菜单的 **Count** 属性相同。(菜单的 **Count** 属性表示菜单中的菜单项的总数。)



在上图中你可以看到第一个索引位置是零(0), 而分隔符被视为一个单独的菜单项有它自己的索引位置。

上图中的菜单的 **Count** 属性应该为 6。如果要在"Tile

Horizontally"和"Tile **Vertically**"之间新增一个菜单项, 需将索引参数设置为 2, 这就是 **Tile**

Vertically 菜单项的索引。这将把新的菜单项插入到索引 2 的位置, 并且将随后的索引都下移一个位置。

一旦菜单项被创建, 你就不能通过索引属性来改变菜单项的索引。如果要改变一个现有菜单的索引, 你只能删除这个菜单项后在别一个位置重新建立它, 另一种方法是增加或删除周围的菜单项使之到达一个合适的位置。

标签

标签是定义菜单项的内容和格式的一个字符串。菜单项标签能包括 **DIESEL** 字符串表示式, 它们能在每一次被显示的时候有条件地改变标签。要想知道更多关于 **DIESEL** 字符串表示式的用法, 请参看 **AutoCAD** 自定义手册中的"菜单中的 **DIESEL** 表示式"。

除了 **DIESEL** 字符串表示式外, 菜单项标签还能包含特殊的代码。举例而言, 如果一个特征前有一个**&**符, 则说明这个特征是一个加速键。要了解这些特殊代码的完整列表, 请参看 **AutoCAD** 自定义手册中的"下拉菜单和快捷菜单标签的语法"。

用户所见到的菜单项的文字称为标题,它起源于标签, 并且用来说明标签中所包含的所有 **DIESEL** 字符串表示式和特殊代码。例如, 标"编辑(&E)"就生成了标题"编辑(E)"。

一个菜单项建立后, 你可以用 **Label** 属性来改变这个菜单项的标签。

标记

标记，或称为名称标记，是一个由字母和数字及下划线组成的字符串。在一个给出的菜单中，这个字符串唯一地定义了这个菜单项。

一个菜单项建成后，你可以通过改变 **Tag** 属性来改变这个菜单项的标记。

宏

当一个菜单项被选定后，宏是执行一些特殊动作的一系列命令。菜单的宏不仅能成为完成一项任务时的按键的记录，而且可以是命令、**AutoLISP**、**DIESEL** 或

ActiveX 程序代码的复合体。想要了解关于宏的更多应用，请参看 **AutoCAD** 自定义手册中“菜单宏”。

一个菜单项建成后，你可以通过改变 **Macro** 属性来改变这个菜单项的宏。

在弹出菜单中增加菜单项

以下示例如何创建一个名为 **TestMenu** 的新菜单以及在菜单中增加新的菜单项。这个新增的菜单项命名为“**Open**”

，并用宏赋值这个菜单项具有“**OPEN**”命令。

```
Sub Ch6_AddAMenuItem()
```

```
Dim currMenuGroup As AcadMenuGroup
```

```
Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)
```

```
' 建立一个新菜单
```

```
Dim newMenu As AcadPopupMenu
```

```
Set newMenu = currMenuGroup.Menus.Add("TestMenu")
```

' 在新菜单中增加菜单项

Dim newMenuItem As AcadPopupMenuitem

Dim openMacro As String

' 指定宏为 VBA 表达式"ESC ESC _open "

openMacro = Chr(3) + Chr(3) + Chr(95) + "open"
+ Chr(32)

Set newMenuItem = newMenu.AddItem _

(newMenu.count + 1, "Open", openMacro)

' 在菜单条上显示菜单

newMenu.InsertInMenuBar _

(ThisDrawing.Application.menuBar.count + 1)

End Sub

在菜单中增加分隔符

用 **AddSeparator** 方法可在菜单中添加一个分隔符。这种方法创建了一个新的 **PopupMenuItem** 对象并把它加入到了指定菜单。这种 **PopupMenuItem** 对象被赋值为 **acSeparator** 类型。这类菜单项可以通过 **Type** 属性找到。

用 **AddMenuItem** 方法只需输入它的索引参数。这个索引参数是一个整数，它指定了分隔符在菜单中的位置。这个索引是在菜单的标题后由位置零(0)做为初始位置的。

增加菜单的分隔符的示例，请参看"启用和禁用菜单项"。

对菜单项赋值一个加速键

通过 **AutoCAD ActiveX/VBA** 指定菜单的 **Label** 属性，可以对菜单项赋值一个加速键。要定义一个加速键，插入符号&的 ASCII 的等价位码在用作加速键的标签前。例如，标签 **Chr(Asc("&"))+"Edit"**将被显示为"**E**dit"，而字符"**E**"就被用作是加速键。

还有其它方法可以对 **AutoCAD** 菜单和命令指定加速键，但它们在 **AutoCAD ActiveX/VBA** 中不能使用。想要了解这些方法，请参看 **AutoCAD** 自定义撰中的"加速键"。

对菜单项增加一个加速键

以下的例子重复了"在弹出菜单中增加菜单项"，并对"**TestMenu**"和"**Open**"菜单都增加了加速键。其中"**S**"是做为 **TestMenu** 菜单的加速键，而"**O**"则是 **Open** 菜单的加速键。

```
Sub Ch6_AddAMenuItem()
```

```
Dim currMenuGroup As AcadMenuGroup
```



```
Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)
```

```
' 增加一个新菜单
```

```
Dim newMenu As AcadPopupMenu
```

```
Set newMenu = currMenuGroup.Menus.Add _
```

```
("Te" + Chr(Asc("&")) + "stMenu")
```

```
' 在新菜单中增加菜单项
```

```
Dim newItem As AcadPopupMenuItem
```

```
Dim openMacro As String
```

```
' 指定宏为 VBA 表达式"ESC ESC _open "
```

```
openMacro = Chr(3) + Chr(3) + Chr(95) + "open"  
+ Chr(32)
```

```
Set newItem = newMenu.AddMenuItem _
```

```
(newMenu.count + 1, Chr(Asc("&")) _
```

```
+ "Open", openMacro)
```

```
' 在菜单条上显示菜单
```

```
newMenu.InsertInMenuBar _
```

```
(ThisDrawing.Application.menuBar.count + 1)
```

```
End Sub
```

建立层叠式子菜单

要建立一个层叠式子菜单，需用 **AddSubmenu** 方法。这种方法创建了一个新的 **PopupMenuItem** 对象并把它加入到了指定菜单中。这种特殊的 **PopupMenuItem** 对象被赋值为 **acSubmenu** 类型。

用 **AddSubmenu** 方法需要输入三项参数：索引、标签、标记。

索引参数是一个整数，它指定了这个新的菜单项在菜单中的位置。索引是在菜单的标题后由位置零(0)做为初始位置的。要想在菜单的末尾新增一个菜单项，必须将索引参数设置得同菜单的 **Count** 属性相同。(菜单的 **Count** 属性即为这个菜单中的菜单项的总数。)

标签是定义菜单项的内容和格式的一个字符串。用户所见到的说明菜单项的文字称为标题,它起源于标签,并且用来说明标签中所包含的所有 **DIESEL** 字符串表示式和特殊代码。例如,标签"&Edit"就生成了标?quot;Edit"。

标记,或称为名称标记,是一个由字母和数字及下划线组成的字符串。在一个给出的菜单中,这个字符串唯一地定义了这个菜单项。

返回新菜单

用 `AddSubMenu` 方法不能返回它建立的 `PopupMenuItem` 对象，而是返回这个子菜单所指的新菜单。这个作为 `PopupMenu` 对象被返回的新菜单，能象一个普通菜单那样被组装。

要想了解关于组装一个菜单的更多内容，请参看"在菜单中增加一个新菜单项"

建立并组装一个子菜单

以下示例如何建立一个叫"**TestMenu**"的新菜单，并为它增加一个叫"**OpenFile**"的子菜单。这个子菜单再和一个叫"**Open**"的菜单项组合，当执行这个命令时可以打开一张图。最后，让这个菜单在菜单条上显示。

```
Sub Ch6_AddASubMenu()
```

```
Dim currMenuGroup As AcadMenuGroup
```

```
Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)
```

```
' 建立一个新菜单
```

```
Dim newMenu As AcadPopupMenu
```

```
Set newMenu = currMenuGroup.Menus.Add("TestMenu")
```

```
' 增加子菜单
```

```
Dim FileSubMenu As AcadPopupMenu
```

```
Set FileSubMenu = newMenu.AddSubMenu("", "OpenFile")
```

```
' 在子菜单中增加一个菜单项
```

```
Dim newItem As AcadPopupMenu
```

```
Dim openMacro As String
```

```
' 指定宏为 VBA 表达式 "ESC ESC _open "
```

```
openMacro = Chr(3) + Chr(3) + Chr(95) + "open"  
+ Chr(32)
```

```
Set newItem = FileSubMenu.AddMenuItem _
```

```
(newMenu.count + 1, "Open", openMacro)
```

```
' 菜单条上显示菜单
```

```
newMenu.InsertInMenuBar _
```

```
(ThisDrawing.Application.menuBar.count + 1)
```

```
End Sub
```

在菜单上删除菜单项

用菜单项中的 **Delete** 方法可以删除菜单上的菜单项。

在一个菜单中删除一个菜单项

以下示例如何在菜单条的最后一个菜单的末尾新增一个菜单项，然后再删除这个菜单项。

```
Sub Ch6_DeleteMenuItem()
```

```
Dim LastMenu As AcadPopupMenu
```

```
Set LastMenu = ThisDrawing.Application.menuBar. _
```

```
Item(ThisDrawing.Application.menuBar.count - 1)
```

```
' 增加一个新菜单项
```

```
Dim newItem As AcadPopupMenuItem
```

```
Dim openMacro As String
```

```
' 指定宏为 VBA 表达式"ESC ESC _open "
```

```
openMacro = Chr(3) + Chr(3) + Chr(95) + "open"  
+ Chr(32)
```

```
Set newMenuItem = LastMenu.AddMenuItem _
```

```
(LastMenu.count + 1, "Open", openMacro)
```

' 从菜单中删除这个菜单项

```
newMenuItem.Delete
```

```
End Sub
```

研究菜单项的属性

所有的菜单项具有以下属性：

标记

标记，或称为名称标记，是一个由字母和数字及下划线组成的字符串。在一个给出的菜单中，这个字符串唯一地定义了这个菜单项。标记也定义了与这个菜单项相应的加速键(键盘键次序)。

你可以通过用 **Tag** 属性来读或写标记的值。

标签

标签是定义菜单项的内容和格式的一个字符串。

菜单项标签可包含 **DIESEL** 字符串表示式，它们能在每一次被显示的时候有条件地改变标签。要想知道更多关于 **DIESEL** 字符串表示式的用法，请参看 **AutoCAD 自定义手册**中的"菜单中的 **DIESEL** 表示式"。

你可以通过使用 **Label** 属性来读或写标签的值。

标题

标题是用户看到的说明菜单项的文字。它的属性是只读的，它源于通过移去任何 **DIESEL** 字符串表示式 **Label** 属性。

你可以通过使用 **Caption** 属性来读取标题的值。

宏

当一个菜单项被选定后，宏是执行一些特殊动作的一系列命令。菜单的宏不仅能完成一项任务时的按键记录，而且可以是命令、**AutoLISP**、**DIESEL** 或 **ActiveX** 程序代码的复合体。想要了解关于宏的更多应用，请参看 **AutoCAD** 自定义手册中的"菜单宏"。

你可以通过使用 **Macro** 属性来读或写宏的值。

帮助字符

当用户将选定的命令亮显后，帮助字符就是这个命令的说明文字，它出现在 **AutoCAD** 的状态栏上。

你可以通过运用 **HelpString** 属性来读或写帮助字符的值。

启用

使用 **Enable** 属性你可以启用或禁用一个菜单项。你也可以通过读取 **Enable** 属性来判断这个这个菜单项当前是启用还是禁用状态。使用这个属性来启用或禁用一个菜单项无需在这个菜单项的 **DIESEL** 表达式中进行任何设置。

要了解如何禁用菜单项，请参看"启用和禁用菜单项"。

选定

选用 **Check** 属性你可以选中或不选中一个菜单项。你也可以通过读取 **Check** 属性来判断这个这个菜单项当前是选中还是不选中的。运用这个属性来选中或不选中一个菜单项无需在这个菜单项的 **DIESEL** 表达式中进行任何设置。

索引

菜单项的索引指明了菜单项在所属菜单中的位置。一个菜单的索引位置通常是由位置 **0** 开始的。例如，如果这个菜单项是菜单中的第一项，它的索引位置就是位置 **0**，如果它是菜单中的第二个菜单项，则它的索引位置就是位置 **1**，以此类推。

类型

使用 **Type** 属性可以判断菜单项的类型。一个菜单项可以是以下几种类型之一：常规菜单、分隔符或子菜单的标题。如果这个菜单项是常规菜单项，该属性返回 **acMenuItem**，如果这个菜单项是一个分隔符，该属性返回 **acMenuSeparator**，如果这个菜单项是子菜单的标题，该属性则返回 **acSubMenu**。

子菜单

具有 **SubMenu** 属性的菜单就是子菜单。如果这个菜单项是 **acSubMenu** 类型，该属性返回作为子菜单依附的菜单或内嵌菜单。内嵌菜单是被做为 **PopupMenu** 对象返回。

如果这个菜单项不是 **acSubMenu** 类型，则该属性返回 **NULL**。

父对象

通过查看 **Parent** 属性，你可以找到一个菜单项所从属的菜单。该属性返回菜单项所从属的菜单。父菜单是被做为 **PopupMenu** 对象返回。

启用和禁用菜单项

以下示例了如何建立一个称为"**TestMenu**"的新菜单，并在新菜单中插入两个菜单项。使用 **Enable** 属性可以将第二个菜单项禁用，最后将这个菜单显示在菜单条上。


```
Sub Ch6_DisableMenuItem()
```

```
Dim currMenuGroup As AcadMenuGroup
```

```
Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)
```

```
' 新菜单
```

```
Dim newMenu As AcadPopupMenu
```

```
Set newMenu = currMenuGroup.Menus.Add("TestMenu")
```

```
' 在新菜单中增加两个菜单项和一个菜单分隔符
```

```
Dim MenuEnable As AcadPopupMenuItem
```

```
Dim MenuDisable As AcadPopupMenuItem
```

```
Dim MenuSeparator As AcadPopupMenuItem
```

```
Dim openMacro As String
```

```
' 赋值宏的 VB 表达式为"ESC ESC _open "
```

```
openMacro = Chr(3) + Chr(3) + Chr(95) + "open"
```

```
+ Chr(32)
```

```
Set MenuEnable = newMenu.AddItem _  
  
(newMenu.count + 1, "OpenEnabled", openMacro)  
  
Set MenuSeparator = newMenu.AddSeparator("")  
  
Set MenuDisable = newMenu.AddItem _  
  
(newMenu.count + 1, "OpenDisabled", openMacro)
```

```
' 禁用第二个菜单项
```

```
MenuDisable.Enable = False
```

```
' 在菜单条上显示菜单
```

```
newMenu.InsertInMenuBar _
```

```
(ThisDrawing.Application.menuBar.count + 1)
```

建立并编辑工具栏

你可以使用 AutoCAD ActiveX/VBA 在一个现有的菜单组中建立并编辑工具栏。

本节主题

建立新工具栏

在工具栏上增加新的工具栏按钮

在工具栏上增加分隔符

定义工具栏按钮的图标

建立弹出工具栏

浮动或固定工具栏

在工具栏上删除工具栏按钮

研究工具栏项的属性

建立新工具栏

要建立一个新的工具栏，要用 **Add** 命令在工具栏集中增加一个新的工具栏对象。

工具栏的名称

使用 **Add** 方法要求输入新工具栏的名称。这个名称必须是一个不含标点符号(除了破折号-或下划线_)的字符串。这个名称是在集合中识别工具栏的最简易的方法。

在工具栏建立后你可以改变工具栏的名称。要改变现有工具栏的名称，使用工具栏的 **Name** 属性。

建立一个新工具栏

以下示例如何在 **MenuGroups** 集合中的第一个菜单组中建立一个称为 **TestToolbar** 的新工具栏。

```
Sub Ch6_CreateToolbar()
```

```
Dim currMenuGroup As AcadMenuGroup
```

```
Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)
```

' 建立新的工具栏

```
Dim newToolbar As AcadToolbar
```

```
Set newToolbar = currMenuGroup.Toolbars.Add("TestToolbar")
```

```
End Sub
```

在工具栏中增加新的工具栏按钮

用 **AddToolbarButton** 方法可以在工具栏中增加新的工具栏按钮。这种方法创建了一个新 **ToolbarItem** 对象并把它加入到了指定工具栏中。只有当这个工具栏是可见的情况下，你才可以对这个工具栏增加工具栏按钮。

使用 **AddToolbarButton** 方法需要输入五项参数：索引、名称、帮助字符、宏、弹出按钮。

索引

索引参数是一个整数，它指定了这个新的工具栏项在工具栏中的位置。索引是在工具栏的标题后由位置零(0)做为开始位置的。要想在一个工具栏的末尾增加一个新的工具栏按钮，必须将索引参数设置得同工具栏的 **Count** 属性相同。(菜单的 **Count** 属性表示了这个工具栏中的工具栏按钮的总数。)

一旦一个工具栏按钮建立后，你不可以通过 **Index** 属性来改变这个按钮的索引。要想改变一个已有工具栏按钮的索引，你只能删除这个按钮，然后再在其它位置重新建立，或者增加(删除)它周围的工具栏按钮以便使它到达理想的位置。

名称

名称是用来定义工具栏按钮的字符串。这个名称必须是一个不含标点符号(除了破折号-或下划线_)的字符串。当箭头指向某个工具栏按钮时, 这个字符串就被做为是该工具的提示。

当一个工具栏按钮建立后, 你可以通过使用 **Name** 的属性来改变工具栏按钮的名称。

帮助字符

当用户将选定的菜单突出后, 帮助字符就是这个命令的说明文字, 出现在 **AutoCAD** 的状态栏上。

一旦一个工具栏按钮建立后, 你可以运用 **HelpString** 属性来改变按钮的帮助字符。

宏

当一个工具栏按钮被选定后, 宏是执行一些特殊动作的一系列命令。工具栏的宏不仅能成为完成一项任务时的按键记录, 而且可以是命令、**AutoLISP**、**DIESEL** 或

ActiveX 程序代码的复合体。想要了解关于宏的更多应用, 请参看 **AutoCAD** 自定义手册中的"菜单宏"一章。

当一个工具栏按钮建立后, 你可以通过 **Macro** 属性来改变按钮的宏。

弹出按钮

FlyoutButton 参数是一个可选的标记, 它用来说明新建按钮是不是一个弹出按钮。如果新建按钮是一个弹出按钮, 那么它的 **FlyoutButton** 参数就必须设为 **TRUE**。如果新建按钮不是一个弹出按钮, 那么它的 **FlyoutButton** 参数就必须设为 **FALSE** 或忽略这个参数。

对新工具栏增加按钮

以下示例建立一个新工具栏并在工具栏上增加一个按钮。当选定这个按钮时, 这个按钮被赋值为一个执行 **OPEN** 命令的宏。

```
Sub Ch6_AddButton()
```

```
Dim currMenuGroup As AcadMenuGroup
```

```
Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)
```

```
' 建立一个新工具栏
```

```
Dim newToolbar As AcadToolbar
```

```
Set newToolbar = currMenuGroup.Toolbars.Add("TestToolbar")
```

```
' 在新工具栏上增加一个按钮
```

```
Dim newButton As AcadToolbarItem
```

```
Dim openMacro As String
```

```
' 赋值这个宏的 VB 表达式为"ESC ESC _open "
```

```
openMacro = Chr(3) + Chr(3) + Chr(95) + "open"  
+ Chr(32)
```

```
Set newButton = newToolbar.AddToolbarButton _
```

```
("", "NewButton", "Open a file.",  
openMacro)
```

End Sub

对一个工具栏增加分隔符

用 **AddSeparator** 方法可以对一个工具栏增加分隔符。这种方法建立了一个新的 **PopupToolBarItem** 对象并把它加入到了指定的工具栏中。这种 **PopupToolBarItem** 对象被赋值为 **acSeparator** 类型。通过 **Type** 属性可以得到这种类型的工具栏按钮。

使用 **AddToolBarButton** 方法需要输入一项参数：索引。索引参数是一个整数，它指定了分隔符在工具栏中的位置。这个索引是在工具栏的标题后由位置零(0)做为开始位置的。

定义工具栏按钮的图像

使用 **SetBitmaps** 和 **GetBitmaps** 方法可以定义一个工具栏按钮的图像。

使用 **SetBitmaps** 方法需要输入两项参数：**SmallIconName**(小图标名称)和 **LargeIconName**(大图标名称)。

小图标名称

小图标名称确定了小图标资源的 ID 字符(16 x 15 位图)。该名称必须是一个不含标点符号(除了破折号(-)或下划线(_)外的字符串)，并且必须包含 **.bmp** 的扩展名。小图标可以是系统的位图也可以是用户自定义的位图。用户自定义的位图必须是适当的尺寸而且必须存在于支持路径中。

大图标名称

大图标名称确定了大图标资源的 ID 字符(16 x 15 位图)。该名称必须是一个不含标点符号(除了破折号(-)或下划线(_)外的字符串)，并且必须包含 **.bmp** 的扩展名。小图标可以是系统的位图也可以是用户自定义的位图。用户自定义的位图必须是适当的尺寸而且必须存在于支持路径中。

查询一个现有工具栏以便找到工具栏按钮的图标的名称

```
Sub Ch6_GetButtonImages()
```

```
Dim Button As AcadToolbarItem
```

```
Dim Toolbar0 As AcadToolbar
```

```
Dim MenuGroup0 As AcadMenuGroup
```

```
Dim SmallButtonName As String
```

```
Dim LargeButtonName As String
```

```
Dim msg As String
```

```
Dim ButtonType As String
```

```
' 第一个菜单组中选取第一个工具栏
```

```
Set MenuGroup0 = ThisDrawing.Application. _
```

```
MenuGroups.Item(0)
```

```
Set Toolbar0 = MenuGroup0.Toolbars.Item(0)
```

```
' 清除字符变量
```



```
SmallButtonName = ""
```

```
LargeButtonName = ""
```

```
' 为消息框建立一个标题并显示被询问的工具栏
```

```
msg = "Toolbar: " + Toolbar0.Name + vbCrLf
```

```
Toolbar0.Visible = True
```

```
' 重复为工具栏的每个按钮收集数据，
```

```
' 如果这个工具栏是一个普通按钮或弹出按钮，
```

```
' 则为这个按钮收集大图标和小图标的名称。
```

```
For Each Button In Toolbar0
```

```
ButtonType = Choose(Button.Type + 1, "Button",
```

```
—
```

```
"Separator", "Control", "Flyout")
```

```
msg = msg & ButtonType & ": "
```

```
If Button.Type = acToolbarButton Or _
```

```
Button.Type = acToolbarFlyout Then
```

```
Button.GetBitmaps SmallButtonName, _
```

```
LargeButtonName
```

```
msg = msg + SmallButtonName + ", " _
```

```
+ LargeButtonName
```

```
End If
```

```
msg = msg + vbCrLf
```

```
Next Button
```

```
' 显示结果
```

```
MsgBox msg
```

```
End Sub
```

建立弹出工具栏

使用 **AddToolBarButton** 方法可以对一个工具栏增加一个工具栏按钮。这种方法创建了一个新的 **ToolBarItem** 对象并把它加入到设计中的工具栏。

使用 **AddToolBarButton** 方法需要输入五种参数：索引、名称、帮助字符、宏和弹出按钮。通过将 **Flyout** 参数设为 **TRUE**，这个新的按钮将被设为一个弹出按钮。这种方法产生的结果将形成一个新的弹出工具栏。这个弹出工具栏可以象一个普通的工具栏那样被组装。

要想了解增加关于组装一个工具栏的更多内容，请参看"在一个工具栏中增加一个新的工具栏按钮"。

建立一个弹出工具栏按钮

以下的例的例子中建立了两个工具栏，第一个工具栏中包含了一个弹出按钮。第二个工具栏就是第一个工具栏上的弹出按钮的附件。

```
Sub Ch6_AddFlyoutButton()
```

```
Dim currMenuGroup As AcadMenuGroup
```

```
Set currMenuGroup = ThisDrawing.Application. _
```

```
MenuGroups.Item(0)
```

```
' 建立第一个工具栏
```

```
Dim FirstToolbar As AcadToolbar
```

```
Set FirstToolbar = currMenuGroup.Toolbars. _
```

```
Add("FirstToolbar")
```

' 在菜单条上的第一个菜单中增加一个弹出按钮

```
Dim FlyoutButton As AcadToolbarItem
```

```
Set FlyoutButton = FirstToolbar.AddToolbarButton _
```

```
("", "Flyout", "Demonstrates  
a flyout button", _
```

```
"OPEN", True)
```

' 建立第二个工具栏，这个工具样将通过弹出按钮

' 而被附带在第一个工具栏上。

```
Dim SecondToolbar As AcadToolbar
```

```
Set SecondToolbar = currMenuGroup.Toolbars. _
```

```
Add("SecondToolbar")
```

' 在下一个工具栏上增加一个按钮

```
Dim newButton As AcadToolbarItem
```

```
Dim openMacro As String
```

```
' 赋值这个宏的 VB 表达式为"ESC ESC _open "
```

```
openMacro = Chr(3) + Chr(3) + Chr(95) + "open"  
+ Chr(32)
```

```
Set newButton = SecondToolbar.AddToolBarButton _
```

```
("", "NewButton", "Open a file.",  
openMacro)
```

```
' 将第二个工具栏附在第一个工具栏的弹出按钮上
```

```
FlyoutButton.AttachToolBarToFlyout currMenuGroup.Name,  
_
```

```
SecondToolbar.Name
```

```
' 显示第一个工具栏， 隐藏第二个工具栏。
```

```
FirstToolbar.Visible = True
```

```
SecondToolbar.Visible = False
```

```
End Sub
```

浮动和固定工具栏

工具栏可以进行程序化的浮动或固定。

用 **Float** 方法可以浮动一个工具栏。这种方法需要输入三项参数：上侧、左侧及浮动行数。上侧和左侧参数指定了工具栏到上侧和左侧边缘的像素位置。浮动行数指定了建立一个水平工具栏需要的行数。这个数字必须大于等于 1。工具栏的按钮必须在行数两边均等分布。如果是垂直的工具栏，这个数字指的是列数。

对工具栏使用 **Dock** 方法可以固定一个工具栏。这种方法需要输入三项参数：侧、行和列。侧参数指定了在这个操作过程中工具栏在屏幕的哪一侧，你可以设工具栏在屏幕的上、下、左或右侧。行和列参数指定了工具栏固定到已有固定工具栏中的哪一行和哪一列。

你可以用 **Docked** 属性来查询这个工具栏是否被固定。如果是固定的，**Docked** 属性为 **TRUE**，反之，如果是浮动的，则这个参数为 **FALSE**。

固定工具栏

以下示例如何建立一个新的工具栏并在工具栏上设三个按钮，然后这个工具栏被显示并固定在屏幕的左侧。

```
Sub Ch6_DockToolbar()
```

```
Dim currMenuGroup As AcadMenuGroup
```

```
Set currMenuGroup = ThisDrawing.Application. _
```

```
MenuGroups.Item(0)
```

' 建立新的工具栏

Dim newToolbar As AcadToolbar

Set newToolbar = currMenuGroup.Toolbars. _

Add("TestToolbar")

' 在新的工具栏上增加三个按钮

' 这有一个按钮将依附于相同的宏。

Dim newButton1 As AcadToolbarItem

Dim newButton2 As AcadToolbarItem

Dim newButton3 As AcadToolbarItem

Dim openMacro As String

' 赋值宏的 VB 表达式为"ESC ESC _open "

openMacro = Chr(3) + Chr(3) + Chr(95) + "open"

+ Chr(32)

Set newButton1 = newToolbar.AddToolbarButton _

("", "NewButton1", "Open a
file.", openMacro)

Set newButton2 = newToolbar.AddToolbarButton _

("", "NewButton2", "Open a
file.", openMacro)

Set newButton3 = newToolbar.AddToolbarButton _

("", "NewButton3", "Open a
file.", openMacro)

'显示工具栏

newToolbar.Visible = True

'将工具栏设置在屏幕左边

newToolbar.Dock = acToolbarDockLeft

End Sub

从工具栏上删除一个工具按钮

要在工具栏上删除一个工具按钮，可使用工具按钮上 **Delete** 方法。你只能删除在工具栏上可见的工具按钮。

研究工具栏项的属性

所有工具栏项具有以下属性

Tag(标记)

标记，或称为名称标记，是一个由字母和数字及下划线组成的字符串。在一个给出的菜单中，这个字符串唯一地定义了这个菜单项。当一个工具栏项建立后，一个新的标记就被自动赋值了。

你可以通过使用 **Tag** 属性来读或写标记的值。

Name(名称)

名称是用来识别工具栏项的字符串。它也是用于工具提示的文字说明的字符串。也就是说当用户使用鼠标或其它点设备点在工具栏项上时，名称就是显示这个工具栏项上的文字说明。

你可以通过使用 **Name** 的属性来读或写名称的值。

Macro(宏)

当一个工具栏项被选定后，宏是执行一些特殊动作的一系列命令。工具栏的宏不仅能成为完成一项任务时的按键记录，而且可以是命令、**AutoLISP**、**DIESEL** 或

ActiveX 程序代码的复合体。想要了解关于宏的更多应用，请参看 **AutoCAD** 自定义手册中的"菜单宏"一章。

你可以通过使用 **Macro** 的属性来读或写宏的值。

Help String(帮助字符串)

帮助字符串是做为一个工具栏按钮的说明文字出现在 **AutoCAD** 的状态栏的。

你可以通过使用 **HelpString** 属性来读或写帮助字符串的值。

Index(索引)

工具栏项的索引指明了工具栏项在所属工具栏中的位置。一个工具栏的索引位置通常是由位置 **0** 开始的。

例如，如果这个工具栏项是工具栏中的第一项，它的索引位置就是位置 **0**，如果它是工具栏中的第二个工具栏项，则它的索引位置就是位置 **1**，以此类推。

你可以通过查询 **Index** 属性来读取工具栏项的索引位置。

Type(类型)

一个工具栏项可以是以下几种类型之一：常规工具栏按钮、分隔符、弹出工具栏按钮或特殊控制按钮。如果这个工具栏项是常规工具栏按钮，它的属性显示为 **acButton**；如果这个工具栏项是一个分隔符，它的属性显示为 **acToolButtonSeparator**；如果这个工具栏项是一个弹出按钮，它的属性则显示为 **acFlyout**；如果这个工具栏项是一个特殊控制按钮，它的属性显示为 **acControl**。

你可以用 **Type** 属性来判断工具栏项的类型。

Flyout(弹出按钮)

如果这个工具栏项是 **acFlyout** 类型，这个属性使得这个工具栏被做为一个弹出工具栏而附带。

这个弹出工具栏又被做为一个 **Toolbar** 的对象。

如果这个菜单项不是 **acFlyout** 类型，则它的属性显示为 **NULL**。

你可以使用 **Flyout** 属性来找出工具栏项中的弹出工具栏。

Parent(父对象)

这个属性返回工具栏项所从属的工具栏。父工具栏是被做为一个 **Toolbar** 对象的。

你可以使用 **Parent** 属性找到工具栏项所从属的工具栏。

Toolbar Properties(工具栏属性)

还有一些属性适用于工具栏上的所有工具项。这些属性包括：这个工具栏是固定的或是浮动的，可见的或是不可见的，使用用大图标或是小图标。

建立宏

当一个工具栏项被选定后，宏是执行一些特殊动作的一系列命令。工具栏的宏不仅能成为完成一项任务时的按键记录，而且可以是命令、**AutoLISP**、**DIESEL** 或 **ActiveX** 程序代码的复合体。

如果你想在菜单的宏中包含命令参数，你必须了解这个命令所要求的参数的次序。菜单的宏中的每一个动作都是有意义的，甚至空格也是有意义的。正如 **AutoCAD** 会被修订和增强一样，不同命令的提示符的次序(有时甚至是命令的名称)可能会改变。因此，当你升级到一个新的 **AutoCAD** 版本时，可能会要求对你的自定义菜单做小小的改动。

当从一个菜单项传来命令输入时，**PICKADD** 和 **PICKAUTO** 系统变量的设置要分别假设为 **1** 和 **0**。这就保证了同旧 **AutoCAD** 版本的兼容性并且使用户自定义过程更容易，因为你无需再去检查这些变量的设置。

本节主题

宏字符所映射的 **ASCII** 等效值

宏的终止

暂停以等待用户输入

取消一个命令

宏的重复

单一对象选择方式模式

宏字符所映射的 ASCII 等效值

下表提供了用于菜单宏中的特殊字符以及它们被用于 VB 和 VBA 时的 ASCII 等效值。当为 Macro 属性建立字符串时，对这些特殊字符使用 ASCII 等效值是很重要的。

用于菜单和工具栏的宏中的特殊字符

字符	ASCII 等效值	说明
;	chr(59)	执行 ENTER
^M	chr(94) + chr(77)	执行 ENTER
^	chr(94) + chr(124)	执行 TAB
SPACEBAR	chr(32)	输入空格；在一个菜单项的命令次序之间键入空格等于按 SPACEBAR
\	chr(92)	暂停等待用户输入
_	chr(95)	转换随后的 AutoCAD 命令及关键字
+	chr(43)	在下一行(如果是最后一个字符)继续执行菜单的宏。
=*	chr(61) + chr(42)	显示当前的顶层图象、下拉菜单或快捷菜单
*^C^C	chr(42) + chr(94) + chr(67)chr(94) + chr(67)	一个重复项的前缀
\$	chr(36)	加载一个菜单区或引入一个条件 DIESEL 宏表达式
^B	chr(94) + chr(66)	切换捕捉开或关(CTRL+B)

^C	chr(94) + chr(67)	取消命令(CTRL+C)
ESC	Chr(3)	取消命令(ESC)
^D	chr(94) + chr(68)	切换坐标开或关(CTRL+D)
^E	chr(94) + chr(69)	设置下一个等轴平面(CTRL+E)
^G	chr(94) + chr(71)	切换栅格的开或关(CTRL+G)
^H	chr(94) + chr(72)	执行退格
^O	chr(94) + chr(79)	切换正交的开或关(CTRL+O)
^P	chr(94) + chr(80)	切换菜单提示的开或关
^Q	chr(94) + chr(81)	显示所有 DOS 提示、状态列表和 输入到打印机。(CTRL+Q)
^T	chr(94) + chr(84)	切换数字仪的开或关(CTRL+T)
^V	chr(94) + chr(86)	改变当前的视口(CTRL+V)
^Z	chr(94) + chr(90)	零字符限制了在一个菜单项的末 尾自动增加空格键。

宏的终止

当执行一个宏时，在处理命令次序之前，**AutoCAD** 在宏的末尾设置了一个空格。**AutoCAD** 处理以下的菜单宏就象你已经键入了 **line** 和空格键一样。

line

有时这样做却不受欢迎；例如，**TEXT** 或 **DIM** 命令必须用 **ENTER** 键终止，而不能用空格。同样的，有时需要用一个以上的空格(或 **ENTER**)来完成一个命令，但是有些文本编辑器不允许你建立一个带空格的行。由于这些问题就产生了两种特殊的规定。

当宏中出现一个分号(;)时，**AutoCAD** 就用 **ENTER** 代替。

如果行的末尾有一个控制字符、一个反斜线符(\)、一个加号(+)或一个分号(;)时，**AutoCAD** 就不在后面加空格。

看以下的宏

```
erase \;
```

如果该项只是以一个反斜线符(\)结尾(需要用户输入)，它就不能完成 **ERASE** 功能，因为 **AutoCAD** 没有在反斜线符(\)后加一个空格。因此，在用户输入之后，这个宏用了一个分号(;)来强迫产生一个回车键。又如以下例子：

```
ucs
```

```
ucs ;
```

```
text \.4 0 DRAFT Inc;;;Main St.;;;City, State;
```

选择第一个宏并在命令行输入 **ucs** 和空格键，就会出现以下提示：

```
Enter an option [New/Move/orthoGraphic/Prev/Restore/Save/Del/Apply/?/World]
```

```
<World>:
```

选择第二个宏并在命令行输入 **ucs**、空格键和一个分号(;)，将接受缺省值。在屏幕上，第一个和第二个宏并没有明显的区别，因此，你无须把它们放在同一个菜单上。

选择第三个宏就会显示一个起点的提示，然后分三行绘制地址。在三个分号(;;;)中，第一个分号终止文本字符串，第二个可重复 **TEXT** 命令，第三个将调用紧随先前行的缺省位置。

注意：所有特殊的符号必须用他们的 **ASCII** 等效值输入。要查询这些 **ASCII** 等效值，请参看"宏字符所映射的 **ASCII** 等效值"

暂停等待用户输入

有时候这样的做法是很有用的，它可以在宏中需要输入点时放置一个反斜杠(\)来接受键盘或点设备输入。

circle \1

layer off \;

第一个宏暂停后需要用户输入中心点，然后从宏中读取半径为 1。注意在反斜线符(\)后没有空格。下一个宏暂停后要求用户输入一个层的名称，然后关闭该层并退出 LAYER 命令。如果你按了空格(空白)或回车(;)时 LAYER 命令通常提示提示下一操作并退出。

通常情况下，当一个项目输入后，宏就会恢复。因此，它不可能建立接受一个可变数字输入(如对象选择集)并继续下去的宏。然而，SELECT 命令却是例外的；一个反斜杠将暂时挂起的执行直到对象选择完成。例如，看看以下宏：

```
select \change previous ;properties
```

```
color red ;
```

该宏用了 SELECT 命令以创建一个包含一个或多个对象的选择集。它随后执行 CHANGE 命令，使用了 Previous 选项来引用该选择集，并将所有选定对象的颜色更改为红色。

因为反斜杠符号(\)会导致宏暂停下来等待用户输入，你不能在宏中将其用于其它用途。当指定文件目录路径时，使用斜杠(/)作为路径分隔符：例如：/direct/file。

以下情况将延迟宏的继续：

- 如果是需要输入一个点，对象捕捉方式将比输入的实际点优先。
- 如果用 X/Y/Z 点过滤器，宏将保持暂停直到整个点完成。
- 只对 SELECT 命令，宏是当对象选择完成后才会再继续。
- 如果用户用一个透明的命令来响应，宏将被挂起直到透明命令完成后才得以继续。
- 如果用户通过其它宏(提供选项或执行透明命令)来响应，原始的宏将被挂起，在新的选定项处理完成后挂起的宏才得以恢复。

取消一个命令

要想确定你没有上一级的未完成的命令，可以在宏中使用`^C^C`。这相当于在键盘上连续按两次 **ESC** 键。

虽然单一的`^C`可以取消大部分的命令，但从一个 **DIM** 命令返回到命令提示符必须使用`^C^C`。因此，`^C^C`可以在最大程度上确保 **AutoCAD** 返回到命令提示符。

宏循环

一旦你选择了一个命令，你很可能要多次重复该命令后才转到其它命令上。以下就是在多数人使用工具的方式：你先选择一个工具，用它做了很多事后，又换另一个工具，如此类推。为了避免在每使用一次工具前都重复地选择，**AutoCAD** 提供了一个命令循环功能，用一个空响应来激发。然而，你不能用这个功能去指定命令选项。

这个特性使你能够在更换到其它命令之前频繁地重复使用一个命令。如果一个宏以`*^C^C`开始，后面紧跟项的标签，这个宏就被保存在内存中。接下来的命令提示符都由这个宏回答，直到宏被 **ESC** 或选择其它的宏而终止。

不要在由`*^C^C`字符串开始的宏中用`^C`；这会取消宏循环。

以下示例重复进行命令处理的方法：

`*^C^CMOVE Single`

`*^C^CCOPY Single`

`*^C^CERASE Single`

`*^C^CSTRETCH Single Crossing`

`*^C^CROTATE Single`

`*^C^CSCALE Single`

宏循环对图标菜单中的项无效。

单一对象选择模式的使用

单一对象选择就是将对象的选择定为一个单一的选择模式，禁用通过对象选择的普通对象行为，并且使得这个选择返回被后来选项选中的第一个对象。这在宏中非常容易实现。例如，可以用以下的宏：

```
*^C^CERASE single
```

这个宏终止了当前的命令并用单一选择项来激活 **ERASE** 命令。在你选择了这个项之后，你可以指向这个单一的对象使它被擦去，或者可以指向一个空白区来指定一个窗口。用这种方式选择的对象就被擦去了，并且宏是重复的(由于第一位的那个星号)，所以你可以擦去其它的一此东西。单一选择方式可产生 **AutoCAD** 的更多的动态交互。

对菜单项和工具栏项增加状态栏帮助

状态栏帮助信息是程序本身的帮助系统的很重要的一部分。当一个菜单项或工具栏项被选中时，它们是出现在状态栏的简易的、描述性的信息。所有菜单项和工具栏项的状态栏帮助都被包含在这个项的 **HelpString** 属性中。

当菜单项或工具栏项第一次建立时，**HelpString** 的属性是空的。

为一个菜单项增加状态栏帮助

以下示例如何建立一个称为"TestMenu"的新菜单，并在新菜单中增加菜单项"打开"。然后再通过 **HelpString** 属性对这个菜单项赋值状态栏帮助。

```
Sub Ch6_AddHelp()
```

```
Dim currMenuGroup As AcadMenuGroup
```

```
Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)
```

' 建立新菜单

Dim newMenu As AcadPopupMenu

Set newMenu = currMenuGroup.Menus.Add _

("Te" + Chr(Asc("&")) + "stMenu")

' 对新菜单增加菜单项

Dim newItem As AcadPopupMenuItem

Dim openMacro As String

' 为宏赋值一个 VBA 表达式"ESC ESC _open "

openMacro = Chr(3) + Chr(3) + Chr(95) + "open"
+ Chr(32)

' 建立菜单项

Set newItem = newMenu.AddMenuItem _

```
(newMenu.count + 1, "打开"&Chr(Asc("&"))
```

```
—
```

```
+ "O)", openMacro)
```

```
' 为菜单项增加状态栏帮助
```

```
newMenuItem.HelpString = "撕开 AutoCAD 图形文件。"
```

```
' 在菜单条上显示菜单
```

```
newMenu.InsertInMenuBar _
```

```
(ThisDrawing.Application.menuBar.count + 1)
```

```
End Sub
```

在右键菜单中增加条目

右键菜单或快捷菜单，是包含在 AutoCAD 基础菜单组中的特殊菜单。这个菜单只在用户按住 **SHIFT** 键并点击鼠标右键时才出现。

在一个基础菜单组中，当一个菜单的 **ShortcutMenu** 属性为 **TRUE** 时，这个菜单就是快捷菜单。你可以遵循 "为一个菜单添加新的菜单项" 中列出的步骤为快捷菜单增加菜单项。

新的菜单组可能有也可能没有一个快捷菜单。要想为一个菜单组建立一个快捷菜单，须遵循"建立新菜单"一章中列出的指示，并且要把 **POP0** 做为这个新菜单的标签。

在右键菜单的末尾新增一个菜单项

以下示例如何在右键菜单的末尾新增菜单项"OpenDWG".

```
Sub Ch6_AddMenuItemToShortcutMenu()
```

```
Dim currMenuGroup As AcadMenuGroup
```

```
Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)
```

```
'找到快捷菜单并赋值它为 shortcutMenu 变量
```

```
Dim scMenu As AcadPopupMenu
```

```
Dim entry As AcadPopupMenu
```

```
For Each entry In currMenuGroup.Menus
```

```
If entry.ShortcutMenu = True Then
```

```
Set scMenu = entry
```

```
End If
```

```
Next entry
```

```

' 在快捷菜单中增加菜单项

Dim newMenuItem As AcadPopupMenu.Item

Dim openMacro As String

' 为宏赋值 VBA 表达式"ESC ESC _open "

openMacro = Chr(3) + Chr(3) + Chr(95) + "open"
+ Chr(32)

Set newMenuItem = scMenu.AddMenuItem _
("", Chr(Asc("&"))) _
+ "OpenDWG", openMacro)

End Sub

```

第七章 使用事件

事件就是由 AutoCAD 发出的公告或信息，它通知你当前进程中的状态或警告你某事的发生。例如，当打开一张图，BeginOpen 事件就被引发。该事件包含被打开图形的名称。当关闭一张图时又会激发另一个事件。通过这些信息你可以写一个子程序或是事件管理器，它们可以通过这些事件来追踪用户在画某个局部图形中所用的时间。

了解 AutoCAD 中的事件

在 AutoCAD 中有三种类型的事件：

应用程序级事件对应在 AutoCAD 程序和它环境下的变化。这些事件对应的是打开、保存、关闭、打印、新建、AutoCAD 命令的发出、装载或卸载 ARX 和 LISP 程序、系统变量的改变和应用程序窗口的改变。

文档级事件对应的是某个指定文档或它的内容的改变。这些事件对应增加、删除或修改对象、激活一个快捷菜单、首选设置的改变、绘图窗口的改变和图形的再生。也有一些文档级事件对应打开、关闭和打印图纸、从图中装载或卸载 ARX 和 LISP 程序。

对象级事件对应的是指定对象的改变。通常只有一个对象级事件。只要对象被修改，相应的对象级事件就被引发。

事件对应的子程序被称为“事件处理器”，当它们所对应的每个事件每次被引发时，这些子程序都会被自动地执行。信息被包含在事件中，例如在 BeginOpen 事件中的图名就通过参数被传送到事件处理器中。

编写事件处理器的方法

必须记住，事件只是提供状态信息或在 AutoCAD 中发生的活动。尽管事件处理器能被编写成对那些事件有响应，但是当事件处理器被引发时，AutoCAD 经常是在处理命令的过程中。因此，如果事件处理器是在与 AutoCAD 以及它的数据库一起提供安全的操作时，事件处理器是有一定限制的。

不要依靠事件的顺序。

当编写事件处理器时，不要认为事件的顺序会按你想象的精确顺序出现。例如，如果你发出一个 OPEN 命令，那么事件 BeginCommand、BeginOpen、EndOpen 和 EndCommand 就会被引发。然而，它们可以不按以上的顺序发生。你唯一可以确定的是 Begin 事件会在与它相应的 End 事件之前发生。在以上的例子中，事件被引发的顺序可以是 BeginCommand、BeginOpen、EndCommand 和 EndOpen,也可以是 BeginCommand、EndCommand、BeginOpen 和 EndOpen。

不要依靠操作的顺序。

如果你先删除对象 1 然后再删除对象 2，不要以为你会先收到对象 1 的 `ObjectErased` 事件然后再收到对象 2 的 `ObjectErased` 事件，你也可能是先收到对象 2 的 `ObjectErased` 事件然后再收到对象 1 的 `ObjectErased` 事件。

不要从一个事件处理器中尝试任何交互式的功能。

要想在一个事件处理器内部尝试执行交互式的功能可能会引起很严重的问题，因为当一个事件被引发时 `AutoCAD` 可能正在执行某个命令的过程中。因此，你必须避免使用输入-获取的方法，例如，`GetPoint`、`GetEntity`、`GetKeyword`，等等，还有选择设置操作和来自事件处理器内部的 `SendCommand` 方法。

不要在一个事件处理器的内部启动对话框。

对话框被认为是交互式的功能而干扰 `AutoCAD` 的当前操作。然而，消息框和警告框不被认为是交互式的功能而可以安全地执行。

除了已经引发事件的对象，你可以在数据库中对任何对象写入数据。

很明显，任何引发事件被引发的对象会一直在 `AutoCAD` 中打开使用，而且通常这些操作正在进行中。因此，要避免对同一对象的事件处理器中的对象写入任何信息。然而，你可以安全地从正在引发事件的对象中读取信息。例如，假设你在设计一个铺满瓷砖的地板，你建立了一个事件处理器作为地板边界的附件，当你改变地板的大小的时候，事件处理器将自动地加或减去瓷砖以满足新的面积要求。事件处理器能读取地板边界的新范围，但是它自己不能对边界做任何改变。

不要在事件管理器中执行任何将引发相同事件的动作。

如果你在一个事件处理器中执行了相同的命令而引发了同样的事件，你就会建立了一个无限循环。例如，你千万不能从 `BeginOpen` 事件的内部打开一张图，否则，`AutoCAD` 就会一直打开更多的图直到达到可以被允许打开图纸的最大数。

记住当 `AutoCAD` 正在显示一个对话框模式时，事件是不会被引发的。

处理应用程序级事件

应用程序级事件并不是一直存在于 AutoCAD 的 VBA 中的，也就是说，当一个 VBA 项被装载时，应用程序级事件并不会自动被激活。应用程序级事件必须被 VBA 和所有其它的 ActiveX 自动操作控制器激活。

一旦应用程序级事件被引发，你就有了很多可用的事件，它们包括：

AppActivate

仅在主程序窗口是活动窗口之前被引发。

AppDeactivate

仅在主程序窗口不是活动窗口之前被引发。

ARXLoaded

当一个 ObjectARX 程序装载时被引发。

ARXUnloaded

当一个 ObjectARX 程序卸载时被引发。

BeginCommand

在一个命令发出但还没完成时被立即引发。

BeginFileDrop

当一个文件被拖到主程序窗口时被引发。

BeginLISP

在 AutoCAD 接到求一个 LISP 表达式的请求后，立即被引发。

BeginModal

仅在一个模式对话框显示前被引发。

BeginOpen

在 AutoCAD 接到打开一个现有图形的请求后，立即被引发。

BeginPlot

在 AutoCAD 接到打印图形的请求后，立即被引发。

BeginQuit

仅在一个 AutoCAD 进程结束之前被引发。

BeginSave

在 AutoCAD 接到保存图形的请求后，立即被引发。

EndCommand

当一个命令完成后立即被引发。

EndLISP

在完成 LISP 表达式的求值后被引发。

EndModal

仅在一个模式对话框消失后被引发

EndOpen

当 AutoCAD 完成了打开一个现有图形之后，立即被引发。

EndPlot

当一个文件已被发送到打印机后被引发。

EndSave

当 AutoCAD 完成了保存图形后被引发。

LISPCancelled

当取消 LISP 表达式的求值后被引发。

NewDrawing

仅在创建一个新图形之前被引发。

SysVarChanged

当系统变量的值改变时被引发。

WindowChanged

当应用程序窗口有变化时被引发。

WindowMovedOrResized

仅在应用程序窗口移动或调整大小后被引发。

本章主题

激活应用程序级事件

激活应用程序级事件

在你可以用应用级事件之前，你必须建立一个新的类模块，并声明一个带事件的 **AcadApplication** 类型的对象。例如，假定一个新的类模块已经建立并命名为 **EventClassModule**，这个新的类模块将包含带 **VBA** 关键字 **WithEvents** 的应用程序的声明。

建立一个新的类模块并声明一个带事件的应用程序对象：

- 1 在 **VBA IDE** 中，插入一个类模块。从插入菜单中选择“类模块”。
- 2 在工程窗口中选择新的类模块。
- 3 在属性窗口中将这个类模块的名称改为 **EventClassModule**。
- 4 用 **F7** 或通过选择查看代码菜单对象打开类模块的代码窗口。
- 5 在类的代码窗口中添加以下行：

```
Public WithEvents App As AcadApplication
```

当新的对象是被声明为带事件后，它就出现在类模块的对象下拉列表中，并且你可以在类模块中为新对象编写事件程序。（当你在对象框中选择了新对象，对这个对象有效的事件就列在过程下拉列表中。）

但是，在程序执行之前，你必须用 **Application** 对象在类模块中连接被声明的对象，你可以用任何模块中的以下代码来完成这一过程。

连接被声明的对象和 **Application** 对象

- 1 在主模块的编码窗口中，添加以下行到声明段中：

```
Dim X As New EventClassModule
```

- 2 在相同的窗口中，加入以下子程序：

```
Sub InitializeEvents()
```

```
Set X.App = ThisDrawing.Application
```

End Sub

3 在你的主模块的代码中，添加一个调用到 **InitializeApp** 子程序中。

Call InitializeEvents

一旦 **InitializeEvents** 程序执行后，在类模块中的 **App** 对象就会专门指向 **Application** 对象。并且当事件发生时，这个类模块中的任何事件过程都会运行。

当图形被拖到 **AutoCAD** 时提示继续

以下的例子截取一个文件在被拖动到 **AutoCAD** 时的装载过程。这时一个对话框就被打开，它包含了被装载的文件名以及三个按钮“是/否/继续”，这可以使用户选择是否继续装载这个文件。如果用户选择取消这个操作，这个结果就通过 **BeginFileDrop** 事件的 **Cancel** 参数返回，而这个文件就不被装载了。

```
Public WithEvents ACADApp As AcadApplication
```

```
Sub Example_AcadApplication_Events()
```

```
’ 本例初始化公共变量（ACADApp）
```

```
’ 它用于截取 AcadApplication 事件
```

```
,
```

```
’ 首先要运行该过程！
```

```
’ 我们可以从 ThisDocument 对象中获取应用程序，
```

```
’ 但它需要有打开了的图形，
```

```
’ 所以我们直接从系统中取得。
```

```
Set ACADApp = GetObject(, "AutoCAD.Application")
```

```
End Sub
```

```
Private Sub ACADApp_BeginFileDrop _
```

```
(ByVal FileName As String, Cancel As Boolean)
```

’ 本例截取一个应用程序 BeginFileDrop 事件。

’

’ 该事件在图形文件拖入 AutoCAD 中时被引发。

’

’ 要引发该示例事件：

’ 1) 确定运行本例时初始化公共变量

’ (命名为 ACADApp)链接到本事件。

’

’ 2) 从 Windows 桌面或 Windows 资源管理器中

’ 拖动 AutoCAD 图形文件到 AutoCAD 应

’ 用程序中

’

’ 使用 “Cancel” 变量以停止加载拖动的文件，还有

’ “FileName” 变量可让用户注意到哪个文件将被拖

’ 进来。

```
If MsgBox("AutoCAD 将加载文件 " & FileName & vbCrLf _  
    & "是否要继续加载该文件？ ", _  
    vbYesNoCancel + vbQuestion) <> vbYes Then  
Cancel = True  
End If  
End Sub
```

处理文档级事件

文档级事件持久存在于 **AutoCAD VBA** 中。也就是说，当一个 **VBA** 工程被装载时，文档级事件就自动启用。

然而，它们并不能被所有的控制器启用，例如 **VB**。文档级事件只能被其它的 **ActiveX**

自动操作控制器引发。

一旦文档级事件被启用，你就有了很多可利用的事件，它们包括：

Activate

当文档窗口为活动窗口时被引发。

BeginClose

仅在文档被关闭时被引发

BeginCommand

在一个命令发出但还没完成时被立即引发。

BeginDoubleClick

当用户双击图中的物体后被引发。

BeginLISP

在 AutoCAD 接到求一个 LISP 表达式的值的请求后，立即被引发。

BeginPlot

在 AutoCAD 接收到打印图形的请求后，立即被引发。

BeginRightClick

当用户用右键点图形窗口时被引发。

BeginSave

在 AutoCAD 接收到保存图形的请求后，立即被引发。

BeginShortcutMenuCommand

在用户用右键点图形窗口之后，并且在快捷菜单以命令方式出现之前时被引发。

BeginShortcutMenuDefault

在用户用右键点绘图窗口之后，并且在快捷菜单以默认方式出现之前时被引发。

BeginShortcutMenuEdit

在用户用右键点图形窗口之后，并且在快捷菜单以编辑方式出现之前时被引发。

BeginShortcutMenuGrip

在用户用右键点图形窗口之后，并且在快捷菜单以栅格模式出现之前时被引发。

BeginShortcutMenuOsnap

在用户用右键点图形窗口之后，并且在快捷菜单以对象捕捉方式出现之前时被引发。

Deactivate

在图形窗口不是活动窗口时被引发。

EndCommand

当一个命令完成后立即被引发。

EndLISP

在完成 LISP 表达式的求值后被引发。

EndPlot

当一个文件被发送到打印机后被引发。

EndSave

当 AutoCAD 完成了保存图形后被引发。

EndShortcutMenu

当快捷菜单出现后被引发。

LayoutSwitched

当用户转换到另一个而已时被引发。

LISPCancelled

当取消 LISP 表达式的求值时被引发。

ObjectAdded

当图形中增加了一个对象时被引发。

ObjectErased

当图形中的某个对象被删除时被引发。

ObjectModified

当图形中的某个物体被修改时被引发。

SelectionChanged

当当前先拾取的选择集改变时被引发。

WindowChanged

当文档窗口有变化时被引发。

WindowMovedOrResized

仅在图形窗口被移动或被调整大小后被立即引发。

本章主题

在 VBA 外的环境下启用文档级事件

在 VBA 外的环境下编制文档级事件程序

在 VBA 环境下编制文档级事件程序

在 VBA 外环境启用文档级事件

当你可以 VB 环境或其它 VBA 以外的环境下使用文档级事件之前，你必须建立一个新的类模块，并声明一个带事件的 **AcadApplication** 类型的对象。例如，假定一个新的类模块已经建立并命名为 **EventClassModule**，这个新的类模块将包含带 VBA 关键字 **WithEvents** 的应用程序的声明。

建立一个新类模块并声明一个带事件的 **Document** 对象：

- 1 在 VBA IDE 中，插入一个类模块。从插入菜单中，选择类模块。
- 2 在工程窗口中选择新的类模块。
- 3 在工程窗口中将类模块的名称改为 **EventClassModule**。
- 4 用 F7 或通过选择菜单项查看代码打开类的代码窗口。
- 5 在类的代码窗口中，加入以下行：

```
Public WithEvents Doc As AcadDocument
```

当新的对象被声明为带事件后，它就出现在类模块的对象下拉列表中，并且你可以在类模块中为新对象编写事件过程。（当你在对象框中选择了新对象，对该对象有效的事件就列在过程下拉列表中。）

但是，在程序执行之前，你必须连接类模块中被声明的对象到 **Document** 对象，你可以在任何模块中的以下代码来完成这一过程。

连接被声明的对象到文档对象

- 1 在主模块的代码窗口中，在声明段加入以下行：

```
Dim X As New EventClassModule
```

2 在相同窗口中，加入以下子程序：

```
Sub InitializeEvents()
```

```
Set X.Doc = ThisDrawing
```

```
End Sub
```

3 在你的主模块的代码中，添加对 **InitializeApp** 子程序的调用：

```
Call InitializeEvents
```

一旦 **InitializeEvents** 过程执行后，在类模块中的 **Doc** 对象就会指向所创建的 **Document** 对象。并且当事件发生时，这个类模块中的任何事件过程都会运行。

在 **VBA** 外的环境下编制文档级事件程序

一旦文档级事件被引发，你可以从类模块代码窗口的对象下拉表中找到可利用的 **Doc** 类变量。在表中选择 **Doc** 类后，可用的事件表就会出现在过程下拉列表中。你只需选择你要编写事件处理器的事件，则处理器的框架就自动建立了。

在 **VBA** 环境下编制文档级事件程序

正如“处理文档事件”提到的，当一个 **VBA** 工程被装载时文档级事件就自动被激活。要在 **VBA** 环境下为文档级事件编写事件处理器，你只需在代码窗口中的对象下拉列表中选择 **AcadDocument**，文档中可用的事件就会出现在过程下拉列表中。你只需选择你要编写事件处理器的事件，则处理器的框架就自动建立了。

必须注意，由这种代码方式产生的事件处理器只适用于当前活动的图形。要为一个指定的图形建立事件处理器，首先必须遵守

“在 VBA 外的环境下启用文档级事件”中列出的步骤。它可以使你能为一个指定的文档建立事件。

在 **BeginShortcutMenuDefault** 和 **EndShortcutMenu** 事件中更新快捷菜单

以下的例子运用了 **BeginShortcutMenuDefault** 事件的事件处理器在快捷菜单的开始增加了"OpenDWG"菜单项。然后 **EndShortcutMenu** 事件的事件处理器又删除了这个新增项，使它不会被永久保存在用户菜单的构造中。

```
Private Sub AcadDocument_BeginShortcutMenuDefault _  
    (ShortcutMenu As AutoCAD.IAcadPopupMenu)  
On Error Resume Next  
’ 添加菜单项到光标菜单中  
Dim newItem As AcadPopupMenu.Item  
Dim openMacro As String  
openMacro = Chr(27) + Chr(27) + Chr(95) + "open" + Chr(32)  
Set newItem = ShortcutMenu.AddMenuItem _  
    (0, Chr(Asc("&"))) _  
    + "OpenDWG", openMacro)  
End Sub
```

```
Private Sub AcadDocument_EndShortcutMenu _  
    (ShortcutMenu As AutoCAD.IAcadPopupMenu)  
On Error Resume Next  
ShortcutMenu.Item("OpenDWG").Delete  
End Sub
```

处理对象级事件

对象级事件并不是一直存在于 AutoCAD 的 VBA 中的，也就是说，当一个 VBA 程序被装载时，对象级事件并不会自动被激活。对象级事件必须被 VBA 和所有其它的 ActiveX 自动操作控制器激活。

一旦对象级事件被激活，你就有以下的事件可用：

Modified

当图中的某个对象被修改时引发。

本章主题

激活对象级事件

激活对象级事件

在你使用对象级事件之前，你必须建立一个新的类模块，并声明一个对象为带事件的 `AcadObject` 类型。

例如，假定一个新的类模块已经建立并命名为 `EventClassModule`，这个新的类模块将包含带 VBA 关键字 `WithEvents` 的应用程序的声明。

建立一个新的类模块并声明一个带事件的 `Circle` 对象

- 1 在 VBA IDE 中，插入一个类模块。从插入菜单中选择类模块。
- 2 在工程窗口中选择新的类模块。
- 3 在工程窗口中将类模块的名称改为 `EventClassModule`。
- 4 用 `F7` 或通过选择菜单项查看代码打开类模块的代码窗口。
- 5 在类模块的代码窗口中，增加以下行：

Public WithEvents Object As AcadCircle

当新的对象被声明是带事件的，它就出现在分类模块的对象下拉列表中，并且你可以在类模块中为新对象编写事件过程。（当你在对象框中选择了新对象，对这个对象有效的事件就列在过程下拉列表中。）

但是，在程序执行之前，你必须连接类模块中被声明的对象到 **Circle** 对象，你可以用任何模块中的以下代码来完成这一过程。

连接被声明的对象到 **Automation** 对象

1 在主模块的代码窗口中，添加以下行到声明段中：

```
Dim X As New EventClassModule
```

2 在相同窗口中，建立一个叫"MyCircle"的圆并把它初始化为包含事件：

```
Sub InitializeEvents()  
Dim MyCircle As AcadCircle  
Dim centerPoint(0 To 2) As Double  
Dim radius As Double  
centerPoint(0) = 0#: centerPoint(1) = 0#: centerPoint(2) = 0#  
radius = 5#  
Set MyCircle = ThisDrawing.ModelSpace.AddCircle(centerPoint, radius)  
Set X.Object = MyCircle  
End Sub
```

3 在你的主模块的代码中，添加对 **InitializeApp** 子程序的调用：

```
Call InitializeEvents
```

一旦 **InitializeEvents** 程序执行后，在类模块中的圆对象就会指向被建立的圆对象。并且当事件发生时，这个分类模块中的任何事件过程都会运行。

注意：当在 VBA 编写代码时，你必须为每个激活为 **Modified** 事件的对象提供一个事件处理器，如果你不能提供一个事件处理器，VBA 可能会发生意外终止。

只要一个封闭的多义线被更新就显示它的面积

以下示例如何建立一个带事件的细多义线。只要多义线被改变，多义线的事件处理器就可以显示多义线的新面积。要想触发这个事件，你只需在 AutoCAD 中改变多义线的尺寸。记住，在事件处理器活动之前，你必须先运行 **CreatePLineWithEvents** 子程序。

```
Public WithEvents PLine As AcadLWPolyline
```

```
Sub CreatePLineWithEvents()
```

```
’ 本例创建一细多义线
```

```
Dim points(0 To 9) As Double
```

```
points(0) = 1: points(1) = 1
```

```
points(2) = 1: points(3) = 2
```

```
points(4) = 2: points(5) = 2
```

```
points(6) = 3: points(7) = 3
```

```
points(8) = 3: points(9) = 2
```

```
Set PLine = ThisDrawing.ModelSpace. _
```

```
  AddLightWeightPolyline(points)
```

```
PLine.Closed = True
```

```
ThisDrawing.Application.ZoomAll
```

```
End Sub
```

```
Private Sub PLine_Modified _
```

```
  (ByVal pObject As AutoCAD.IAcadObject)
```

```
’ 该事件当多义线大小变化时引发。
```

```
’ 如果多义线被删除时 modified 事件仍然会被引发，
```

```
’ 所以使用错误处理器来避免从删除了的对象中读取数据。
```

```
On Error GoTo ERRORHANDLER
```

```
MsgBox "对象" & pObject.ObjectName & " 的面积为: " _  
& pObject.Area  
Exit Sub
```

ERRORHANDLER:

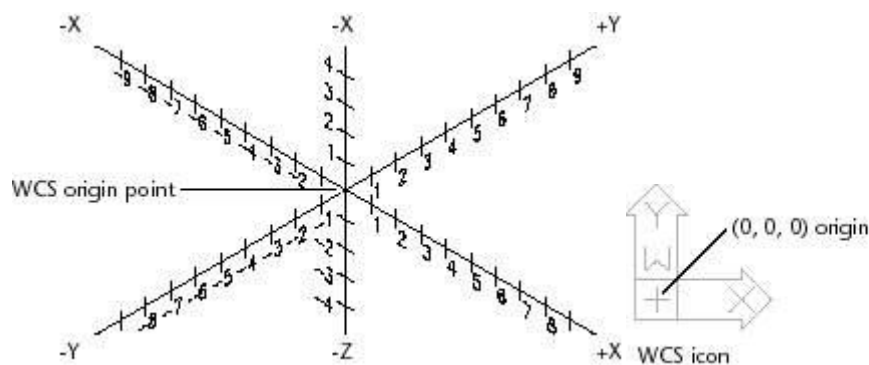
```
MsgBox Err.Description  
End Sub
```

第八章 在三维空间下工作

多数图纸是用二维视图来表现三维的物体。虽然这种方法在建筑学和工程学上广泛应用，但它是有限制的：这些图纸必须要在视觉上能够说明这是三维对象的二维表示图。另外，因为这些视点是独立建立的，所以可能会有很多错误或不明确的地方。结果，你可能想建立一个真实的三维模型来代替二维的表现手法。你可以用 AutoCAD 绘图工具来建立详细的、真实的三维对象并且用多种的方法来操作它们。

指定三维坐标

指定三维坐标和指定二维坐标一样，只是增加了第三维，Z 轴。在三维坐标中绘图时，你必须同时在三维 WCS 坐标和用户坐标系(UCS)中指定 X、Y 和 Z 的值。



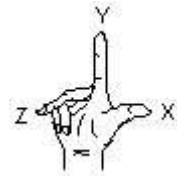
本节主要内容：

右手规则

输入 X、Y、Z 坐标

右手规则

在三维坐标系统中，当你知道了 **X** 轴和 **Y** 轴的方向时，右手规则决定了 **Z** 轴的正轴方向。右手规则同时也决定了在三维空间中轴的正旋转方向。



要决定 **X** 轴、**Y** 轴和 **Z** 轴的正向，先将你的右手的背面靠近屏幕。将你的拇指指向 **X** 轴的正向，如图示伸展你的食指和中指，并将你的食指指向 **Y** 轴的正向。那么你的中指显示的方向就是 **Z** 轴的正向。要决定一个轴的正旋转方向，先将你右手拇指指向这个轴的正向，然后如图示弯曲你其余的手指，那么你的这些手指所示的方向就是这个轴旋转的正向。

输入 **X**、**Y**、**Z** 坐标

输入三维 **WCS** 坐标和输入二维 **WCS** 坐标类似。除了要指定 **X** 和 **Y** 的值外，你还要指定一个 **Z** 值。和二维坐标一样，需要用变量将坐标传给 **ActiveX** 方法和属性，并用它查询坐标。

以下的子程序首先建立一条有三个顶点的二维多义线，然后再建立一条有三个顶点的三维多义线。注意，数组的长度(包括顶点)已经扩展到包含所创建的三维多义线的 **Z** 坐标。这个子程序通过查询两条多义线中的坐标并在对话框中显示坐标而结束。

定义和查询二维和三维多义线坐标

以下的例子建立了两条多义线，每条都包含三个坐标。第一条多义线是二维的，第二条是三维的。然后示例了查询多义线的坐标并在一个对话框中显示坐标。

```
Sub Ch8_Polyline_2D_3D()  
Dim pline2DObj As AcadLWPolyline  
Dim pline3DObj As AcadPolyline
```



```
Dim points2D(0 To 5) As Double
```

```
Dim points3D(0 To 8) As Double
```

```
' 定义三个二维多义线点
```

```
points2D(0) = 1: points2D(1) = 1
```

```
points2D(2) = 1: points2D(3) = 2
```

```
points2D(4) = 2: points2D(5) = 2
```

```
' 定义三个三维多义线点
```

```
points3D(0) = 1: points3D(1) = 1: points3D(2) = 0
```

```
points3D(3) = 2: points3D(4) = 1: points3D(5) = 0
```

```
points3D(6) = 2: points3D(7) = 2: points3D(8) = 0
```

```
' 建立二维细多义线
```

```
Set pline2DObj = ThisDrawing.ModelSpace. _
```

```
    AddLightWeightPolyline(points2D)
```

```
pline2DObj.Color = acRed
```

```
pline2DObj.Update
```

```
' 建立三维多义线
```

```
Set pline3DObj = ThisDrawing.ModelSpace. _
```

```
    AddPolyline(points3D)
```

```
pline3DObj.Color = acBlue
```

```
pline3DObj.Update
```

’ 查询多义线的坐标

```
Dim get2Dpts As Variant
```

```
Dim get3Dpts As Variant
```

```
get2Dpts = pline2DObj.Coordinates
```

```
get3Dpts = pline3DObj.Coordinates
```

’ 显示坐标

```
MsgBox ("二维多义线(红色): " & vbCrLf & _  
get2Dpts(0) & ", " & get2Dpts(1) & vbCrLf & _  
get2Dpts(2) & ", " & get2Dpts(3) & vbCrLf & _  
get2Dpts(4) & ", " & get2Dpts(5))
```

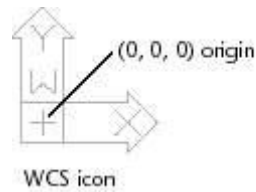
```
MsgBox ("三维多义线(蓝色): " & vbCrLf & _  
get3Dpts(0) & ", " & get3Dpts(1) & ", " & _  
get3Dpts(2) & vbCrLf & _  
get3Dpts(3) & ", " & get3Dpts(4) & ", " & _  
get3Dpts(5) & vbCrLf & _  
get3Dpts(6) & ", " & get3Dpts(7) & ", " & _  
get3Dpts(8))
```

```
End Sub
```

定义用户坐标系统

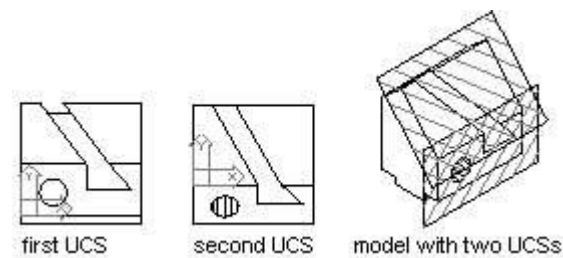
你可以定义一个用户坐标系统(UCS)对象来改变原点(0, 0, 0)位置和 XY 平面及 Z 轴的定位。你可以在三维空间的任何位置定位和定向一个 UCS，而且你也可以按你的需求定义、保存、调用多个用户坐标。坐标的输入和显示是相对于当前的 UCS 的。

如果你想在三维空间下做更多的工作，你可以定义几个用户坐标系统，每一个坐标对应不同的构造要求而有不同的原点和定向。



要想表达 UCS 的原点和定向，你可以用 `UCSIconAtOrigin` 属性在 UCS 的原点显示 UCS 的图标。如果这个 UCS 图标已经打开(参看 `UCSIconOn` 属性)却不是在原点显示，那么它就被 `UCSORG` 系统变量定义为在 WCS 坐标显示。

UCS 在三维空间是十分有用的。你会发现它可以利用现有的几何形状来排列坐标系统，比通过计算出一个三维点的精确位置来排列更容易。



你可以象在模型空间一样，在图纸空间定义一个新的 UCS；然而，在图纸空间的 UCS 受到了二维操作的限制。**AutoCAD** 会保留最后十个分别在模型空间和图纸空间建立的坐标系统的踪迹。

你可以用 **Add** 方法建立一个新的用户坐标系统。这种方法需要输入四个值：原点坐标、X 轴坐标和 Y 轴坐标、UCS 的名称。

在 **AutoCAD ActiveX Automation** 中的所有坐标都被输入到世界坐标系统(WCS)中。用 `GetUCSMatrix` 方法可以返回一个特定的 UCS 的转换矩阵。用这个转换矩阵可以找出相应的 WCS 坐标。

用 **Document** 对象中的 **ActiveUCS** 属性可使一个 **UCS** 激活。如果活动的 **UCS** 有变动，必须重新设置新的 **UCS** 对象作为活动的 **UCS**。要想重新设置活动的 **UCS**，只需用已更新的 **UCS** 对象再调用 **ActiveUCS** 属性就可以了。

建立新的 **UCS**，使之成为活动，并且转换一点的坐标到 **UCS** 坐标

以下的子程序建立一个新的 **UCS**，并设它为图形中活动的 **UCS**。然后这个程序要求用户在图中选择一点，并返回这个点的 **WCS** 坐标和 **UCS** 坐标。

```
Sub Ch8_NewUCS()  
    ' 定义所需的变量  
    Dim ucsObj As AcadUCS  
    Dim origin(0 To 2) As Double  
    Dim xAxisPnt(0 To 2) As Double  
    Dim yAxisPnt(0 To 2) As Double  
    ' 定义 UCS 点  
    origin(0) = 4: origin(1) = 5: origin(2) = 3  
    xAxisPnt(0) = 5: xAxisPnt(1) = 5: xAxisPnt(2) = 3  
    yAxisPnt(0) = 4: yAxisPnt(1) = 6: yAxisPnt(2) = 3  
  
    ' 添加 UCS 到 UserCoordinatesSystems 集合  
    Set ucsObj = ThisDrawing.UserCoordinateSystems. _  
        Add(origin, xAxisPnt, yAxisPnt, "New_UCS")  
    ' 显示 UCS 图标  
    ThisDrawing.ActiveViewport.UCSIconAtOrigin = True  
    ThisDrawing.ActiveViewport.UCSIconOn = True  
  
    ' 使新的 UCS 成为活动的 UCS  
    ThisDrawing.ActiveUCS = ucsObj  
    MsgBox "当前的 UCS 为：" & ThisDrawing.ActiveUCS.Name _  
        & vbCrLf & "在图中拾取一个点。"
```

’ 查找点的 WCS 和 UCS 坐标

```
Dim WCSPnt As Variant
```

```
Dim UCSPnt As Variant
```

```
WCSPnt = ThisDrawing.Utility.GetPoint(, "输入一个点: ")
```

```
UCSPnt = ThisDrawing.Utility.TranslateCoordinates _  
    (WCSPnt, acWorld, acUCS, False)
```

```
MsgBox " WCS 坐标为: " & WCSPnt(0) & ", " _  
    & WCSPnt(1) & ", " & WCSPnt(2) & vbCrLf & _  
    " UCS 坐标为: " & UCSPnt(0) & ", " _  
    & UCSPnt(1) & ", " & UCSPnt(2)
```

```
End Sub
```

坐标转换

用 **TranslateCoordinates** 方法可以将一个点或一段位移由一个坐标系统转换到另一个坐标系统。一个点变量，称为 **OriginalPoint**，可以被视为一个三维点或一个三维位移矢量。这个变量由 **Boolean** 变量-

Disp 来区分。如果 **Disp** 变量被设为 **TRUE** 的话，**OriginalPoint** 变量就被视为一个位移矢量；反之，则被视为一个点。两个以上的变量可以决定这个 **OriginalPoint** 来自哪个坐标系统，也可以决定这个 **OriginalPoint** 要被转换到哪个坐标系统。以下的 **AutoCAD** 坐标系统可以被指定为 **From** 和 **To** 变量。

WCS

世界坐标系统即参照坐标系统。其它所有的坐标系统都是相对 **WCS** 定义的，**WCS** 是永远不改变的。相对于 **WCS** 测量的值可以忽略其它坐标系统的变化。除了特殊说明，所有传进或传出 **ActiveX** 方法和属性的点都用 **WCS** 表示。

UCS

用户坐标系统即工作中的坐标系统。用户指定一个 UCS 以便绘图更容易。所有传到 AutoCAD 命令的点，包括那些从 AutoLISP 程序和外部功能返回的，都是当前 UCS 的点(除了在命令提示符后用户在前面加了个*的点)。如果你想用程序将 WCS、OCS 或 DCS 坐标传到 AutoCAD 命令，你必须首先通过调用 TranslateCoordinates 方法将它们转换成 UCS。

OCS

对象坐标系统-由多义线和细多义线对象的某些方法和属性指定的点的值由这种坐标系统表达，与对象有关。这些点通常根据对象的用途被转换成 WCS、当前的 UCS 或当前的 DCS。相反的，在 WCS、UCS 或 DCS 中的点依靠相同的属性写进数据库之前，必须被转换成 OCS。要了解使用该坐标系统的方法和属性，请参看 AutoCAD 中的"ActiveX 和 VBA 参考"。

当从 OCS 转换坐标或转换坐标到 OCS 时，你必须输入 TranslateCoordinates 方法中的最后一个参数 OCS 法线。

DCS

显示坐标系统即对象在显示前被转换的坐标系统。DCS 的原点是被存在 AutoCAD 系统变量 TARGET 中的点，它的 Z 轴就是视图方向。换句话说，一个视口始终是它的 DCS 平面图。这些坐标可用于决定物体是从哪里显示给 AutoCAD 用户的。

PSDCS

图纸空间 DCS-该坐标系统只能从当前活动的模型空间视口的 DCS 转入或转出。这本来是一个二维的转换，如果 Disp 变量为 FALSE，X 和 Y 坐标总是按比例来偏移的。Z 坐标也是按比例的不转换。因此，可以用 Z 坐标来找到两个坐标系统之间的比例因子。PSDCS 只能被转换成当前的模型空间视口。如果转来的变量等于 PSDCS，那么输出的变量必须等于 DCS，反之亦然。

将 OCS 坐标转换为 WCS 坐标

以下的例子在模型空间建立了一条多义线。多义线的第一个顶点将同时显示为 OCS 和 WCS 坐标。从 OCS 到 WCS 的转换要求设置 TranslateCoordinates 方法的最后一个变量的 OCS 法线。

```

Sub Ch8_TranslateCoordinates()
' 在模型空间中创建多义线。
Dim plineObj As AcadPolyline
Dim points(0 To 14) As Double

' 定义二维多义线点
points(0) = 1: points(1) = 1: points(2) = 0
points(3) = 1: points(4) = 2: points(5) = 0
points(6) = 2: points(7) = 2: points(8) = 0
points(9) = 3: points(10) = 2: points(11) = 0
points(12) = 4: points(13) = 4: points(14) = 0

' 在模型空间中创建一细多义线对象
Set plineObj = ThisDrawing.ModelSpace.AddPolyline(points)

' 查找多义线第一个顶点的 X 和 Y 坐标
Dim firstVertex As Variant
firstVertex = plineObj.Coordinate(0)

' 使用 Elevation 属性找到多义线的 Z 坐标
firstVertex(2) = plineObj.Elevation

' 更改多义线的法线使两坐标系统产生明显的差异。
Dim plineNormal(0 To 2) As Double
plineNormal(0) = 0#
plineNormal(1) = 1#
plineNormal(2) = 2#
plineObj.Normal = plineNormal

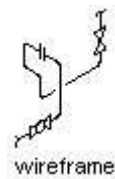
```

将 OCS 坐标译成 WCS 坐标

```
' Translate the OCS coordinate into WCS  
Dim coordinateWCS As Variant  
coordinateWCS = ThisDrawing.Utility.TranslateCoordinates _  
(firstVertex, acOCS, acWorld, False, plineNormal)  
  
' 显示这个点的坐标  
MsgBox "第一个顶点的坐标如下:" _  
& vbCrLf & "OCS: " & firstVertex(0) & ", " & _  
firstVertex(1) & ", " & firstVertex(2) & vbCrLf & _  
"WCS: " & coordinateWCS(0) & ", " & _  
coordinateWCS(1) & ", " & coordinateWCS(2)  
End Sub
```

建立三维对象

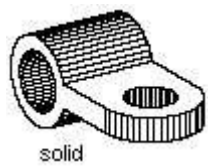
AutoCAD 支持类型的三维模型：框线、网面和实体。每一种类型都有它自己的创建和编辑技巧。



线框模型是一个三维物体的骨架描述。在线框模型中是没有表面的；它只包括描述物体边缘的点、线和曲线。你可以在三维空间的任何位置通过定位一个二维(平面的)物体来建立一个线框模型。AutoCAD 也提供了一些三维线框模型，例如三维多义线。因为组成线框模型的每一个对象都是被独立地绘制和定位的，所以画这种类型的模型经常是最耗时的。



表面模型不仅定义了三维物体的边缘，也定义了三维物体的表面，所以它比线框模型更精密。**AutoCAD** 表面模型用一个多边的网面来定义小面积的表面。因为这些网面的表面是平的。所以网面只能近似地描述曲面。



实体模型是最容易使用的三维模型。用 **AutoCAD** 实体模型，你可以通过建立基本的三维形状来做三维物体：方体、圆柱、球、楔子和圆环(环状物)。你还可以结合这些形状去建立更复杂的实体模型，如并集、差集或查找它们的交集(干涉)。你也可以用一个二维物体扫一个轨迹或用它绕一个轴旋转来建立一个实体。用 **AutoCAD**

设计中心，你也可以定义实心参量和维持三维模型与产生它们的二维视点之间的结合。

注意! 因为每一种模型用了不同的方法来构造三维模型，而且编辑方法对不同的模型会产生不同的影响，所以建议你不要混淆建模方法。模型间的转换只限实体转为表面或表面转为线框，而不可以从线框转为表面或由表面转为实体。

本节主要内容：

建立线框

建立网面

建立多表面网面

建立实体

建立线框

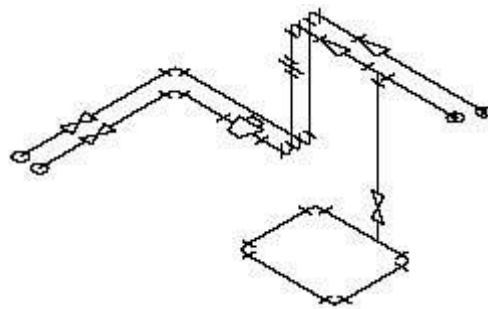
用 **AutoCAD** 你可以在三维空间的任何位置通过设置任何的二维平面物体来建立线框模型。你可以用几种方法在三维空间安置二维物体：

通过输入三维点来建立对象。你可以输入一个定义了 **X、Y、Z** 轴位置的点的坐标。

通过定义一个 **UCS** 建立一个你要绘图的默认坐标平面(**XY** 平面)

当你建立好对象后，将它移至三维空间的合适方位。

同样，你也可以建立一些线框对象，例如多义线，它可以在三维空间存在。用 **Add3Dpoly** 方法可以建立三维多义线。以下的图例是一个用三维多义线和二维抽象位置的组合在三维空间建立的一个三维建模程序。

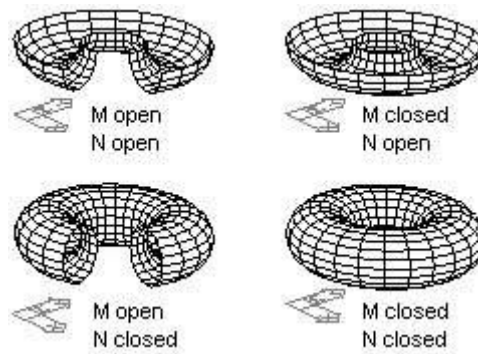


建立网面

一个矩形的网面(**PolygonMesh** 对象)表示一个对象的表面用的是平坦的小平面。网面的密度，或小平面的数目，被定义在一个以 **M** 和 **N** 为矩阵的项里，类似一个由行和列组成的栅格。**M** 和 **N** 分别指定了所有给出的顶点的行和列的位置。你可以在二维或三维中建立网面，但它们主要是被用于三维中。

如果你不需要知道可以由实体提供的物理特性(例如质量、重量、重心等等)的细节标准，但你又确实需要隐蔽、阴影、透视功能，而这些功能线框又不能提供，那么你可以用网面来表达。网面对想用不寻常的网面样式来建立几何体也是很有用的，例如一个山的三维地形模型。

一个网面可以是开放的也可以是封闭的。如果网面的起始端不碰到结束端，那么这个网面就对一个特定的方向打开，如下图所示：



用 **Add3Dmesh** 方法可以建立矩形的网面。这个方法要输入三个值：在 **M** 方向的顶点的数目、在 **N** 方向的顶点的数目以及一个包含在网面中的所有顶点坐标的变体数组。

一旦 **PolygonMesh** 建立，可以用 **Mclose** 和 **Nclose** 属性去闭合网面。

建立一个多边形网面

以下的例子建立了一个 **4 x 4** 多边形网面。然后调整活动视口的方向以便更容易看到这个网面的三维特性。

```
Sub Ch8_Create3DMesh()
```

```
Dim meshObj As AcadPolygonMesh
```

```
Dim mSize, nSize, Count As Integer
```

```
Dim points(0 To 47) As Double
```

’ 创建点矩阵

```
points(0) = 0: points(1) = 0: points(2) = 0
```

```
points(3) = 2: points(4) = 0: points(5) = 1
```

```
points(6) = 4: points(7) = 0: points(8) = 0
```

```
points(9) = 6: points(10) = 0: points(11) = 1
```

```
points(12) = 0: points(13) = 2: points(14) = 0
```

```
points(15) = 2: points(16) = 2: points(17) = 1
```

```
points(18) = 4: points(19) = 2: points(20) = 0
```

```
points(21) = 6: points(22) = 2: points(23) = 1
```

```
points(24) = 0: points(25) = 4: points(26) = 0
```

```
points(27) = 2: points(28) = 4: points(29) = 1
```

```

points(30) = 4: points(31) = 4: points(32) = 0
points(33) = 6: points(34) = 4: points(35) = 0
points(36) = 0: points(37) = 6: points(38) = 0
points(39) = 2: points(40) = 6: points(41) = 1
points(42) = 4: points(43) = 6: points(44) = 0
points(45) = 6: points(46) = 6: points(47) = 0

```

```
mSize = 4: nSize = 4
```

’ 在模型空间中创建三维网面

```

Set meshObj = ThisDrawing.ModelSpace. _
Add3DMesh(mSize, nSize, points)

```

’ 更改视口的视图方向以便更清楚地看到柱面

```

Dim NewDirection(0 To 2) As Double
NewDirection(0) = -1
NewDirection(1) = -1
NewDirection(2) = 1
ThisDrawing.ActiveViewport.direction = NewDirection
ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport
ZoomAll
End Sub

```

建立多表面网面

用 **AddPolyfaceMesh** 方法可以建立一个多表面网面，每个表面包含许多顶点。

建立一个多表面网面和建立矩形网面类似。要建立一个多表面网面，先指定所有顶点的坐标，然后通过输入这个面的所有至高点的顶点的数目来定义每个面。当你建立多表面网面时，你可以设置特殊的边界为看不见的，赋值它们的层，或指定它们的颜色。

要使一个边界看不见，就对这个边界的顶点的数目输入一个负的值。要想了解关于建立多表面网面更多内容，请参看"ActiveX 和 VBA 参考"中的 **AddPolyfaceMesh** 方法。

建立一个多表面网面

以下的例子在模型空间建立了一个多表面网面对象。然后调整活动视口的观看方向以便更容易显示这个网的三维特性。

```
Sub Ch8_CreatePolyfaceMesh()  
    ' 定义网面的至高点  
    Dim vertex(0 To 17) As Double  
    vertex(0) = 4: vertex(1) = 7: vertex(2) = 0  
    vertex(3) = 5: vertex(4) = 7: vertex(5) = 0  
    vertex(6) = 6: vertex(7) = 7: vertex(8) = 0  
    vertex(9) = 4: vertex(10) = 6: vertex(11) = 0  
    vertex(12) = 5: vertex(13) = 6: vertex(14) = 0  
    vertex(15) = 6: vertex(16) = 6: vertex(17) = 1  
  
    ' 定义表面列表  
    Dim FaceList(0 To 7) As Integer  
    FaceList(0) = 1  
    FaceList(1) = 2  
    FaceList(2) = 5  
    FaceList(3) = 4  
    FaceList(4) = 2  
    FaceList(5) = 3  
    FaceList(6) = 6  
    FaceList(7) = 5  
  
    ' 创建多表面网面  
    Dim polyfaceMeshObj As AcadPolyfaceMesh  
    Set polyfaceMeshObj = ModelSpace.AddPolyfaceMesh _
```

(vertex, FaceList)

’ 更改视口的视图方向以便更清楚地看到多表面网面

```
Dim NewDirection(0 To 2) As Double
NewDirection(0) = -1
NewDirection(1) = -1
NewDirection(2) = 1
ThisDrawing.ActiveViewport.direction = NewDirection
ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport
ZoomAll
End Sub
```

建立实体

一个实体(三维实体)表现的是一个对象的整个体积。实体是三维模型中信息最完整和最少不明确因素的类型。复合型实体也比线框结构和网面结构模型更容易构造和编辑。

你可以用以下基本实体形状的其中之一来建立实体，如方体、锥体、圆柱体、球体、圆环体、楔形体，或沿一个轨迹延伸一个二维对象，或者围绕一个轴旋转一个二维对象来建立实体。用以下其中之一的方法可以建立实体：

AddBox、AddCone、AddCylinder、AddEllipticalCone、AddEllipticalCylinder、AddExtrudedSolid、AddExtrudedSolidAlongPath、AddRevolvedSolid、AddSolid、AddSphere、AddTorus 或 AddWedge。

建立一个楔形实体

以下的例子在模型空间建立了楔形实体。然后调整活动视口的观看方向以便更容易显示这个楔形的三维特性。

```
Sub Ch8_CreateWedge()
Dim wedgeObj As Acad3DSolid
Dim center(0 To 2) As Double
```

```

Dim length As Double
Dim width As Double
Dim height As Double

' 定义楔形
center(0) = 5#: center(1) = 5#: center(2) = 0
length = 10#: width = 15#: height = 20#

' 在模型空间中创建楔形体
Set wedgeObj = ThisDrawing.ModelSpace. _
    AddWedge(center, length, width, height)

' 更改视口的视图方向
Dim NewDirection(0 To 2) As Double
NewDirection(0) = -1
NewDirection(1) = -1
NewDirection(2) = 1
ThisDrawing.ActiveViewport.direction = NewDirection
ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport
ZoomAll
End Sub

```

在三维中编辑

本节描述了如何通过旋转、阵列和镜像等编辑三维对象。

本节主要内容：

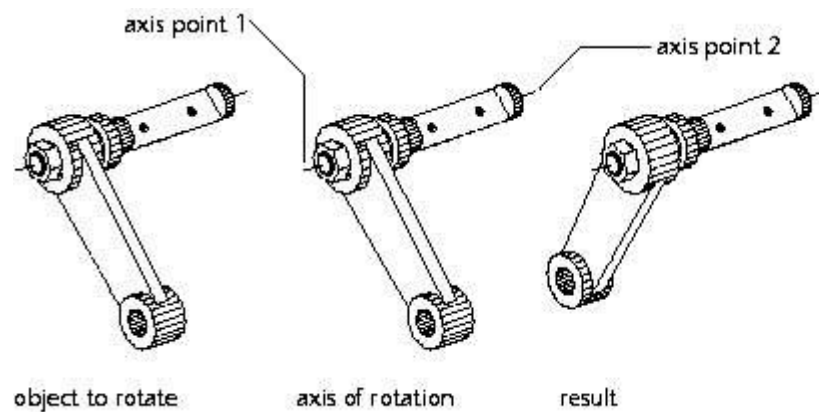
在三维中旋转

在三维中阵列

在三维中镜像

在三维中旋转

用 **Rotate** 方法，你可以在二维中围绕一个指定的点旋转对象。旋转的方向由 **WCS** 决定。用 **Rotate3D** 的方法，你可以在三维中围绕一个指定的轴旋转对象。**Rotate3D** 方法需要输入三个值：定义旋转轴的两点的 **WCS** 坐标和用弧度表示的旋转角。



要旋转一个三维物体，可以用 **Rotate** 或 **Rotate3D** 方法。

以下的子程序建立了一个三维方体并围绕一个轴旋转它。

建立一个三维方体并围绕一个轴旋转

以下示例如何建立一个三维方体，然后定义一个旋转轴，最后使方体围绕这个轴做 **30** 度角旋转。

```
Sub Ch8_Rotate_3DBox()  
Dim boxObj As Acad3DSolid  
Dim length As Double  
Dim width As Double  
Dim height As Double  
Dim center(0 To 2) As Double
```

’ 定义方体


```
center(0) = 5: center(1) = 5: center(2) = 0  
length = 5  
width = 7  
height = 10
```

’ 在模型空间中创建方体对象

```
Set boxObj = ThisDrawing.ModelSpace. _  
AddBox(center, length, width, height)
```

’ 用两点定义旋转轴

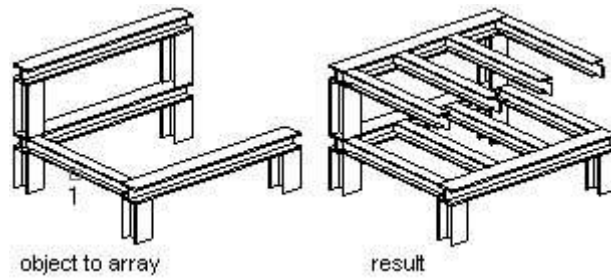
```
Dim rotatePt1(0 To 2) As Double  
Dim rotatePt2(0 To 2) As Double  
Dim rotateAngle As Double  
rotatePt1(0) = -3: rotatePt1(1) = 4: rotatePt1(2) = 0  
rotatePt2(0) = -3: rotatePt2(1) = -4: rotatePt2(2) = 0  
rotateAngle = 30  
rotateAngle = rotateAngle * 3.141592 / 180#
```

’ 旋转方体

```
boxObj.Rotate3D rotatePt1, rotatePt2, rotateAngle  
ZoomAll  
End Sub
```

在三维中阵列

用 **ArrayRectangular** 方法，你可以在三维中建立一个矩形的阵列。除了要指定列(X 方向)和行(Y 方向)的数目，你还要指定层(Z 方向)的数目。



建立一个三维的矩形阵列

以下的例子建立了一个圆，然后运用这个圆来建立一个 4 行、4 列和 3 层的矩形阵列。

```
Sub Ch8_CreateRectangularArray()
' 创建圆
Dim circleObj As AcadCircle
Dim center(0 To 2) As Double
Dim radius As Double
center(0) = 2: center(1) = 2: center(2) = 0
radius = 0.5
Set circleObj = ThisDrawing.ModelSpace. _
AddCircle(center, radius)

' 定义矩形阵列
Dim numberOfRows As Long
Dim numberOfColumns As Long
Dim numberOfLevels As Long
Dim distanceBwtnRows As Double
Dim distanceBwtnColumns As Double
Dim distanceBwtnLevels As Double
numberOfRows = 4
numberOfColumns = 4
numberOfLevels = 3
distanceBwtnRows = 1
```

```

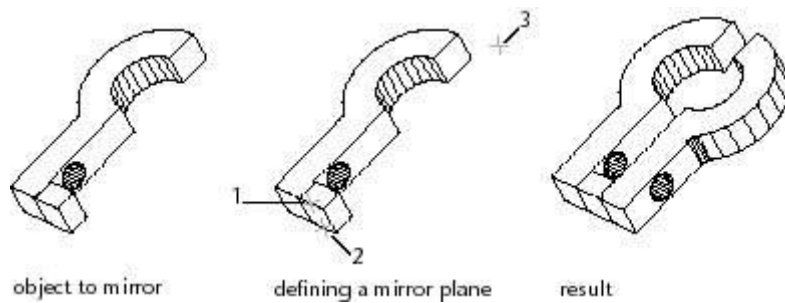
distanceBwtnColumns = 1
distanceBwtnLevels = 4

' 创建对象的阵列
Dim retObj As Variant
retObj = circleObj.ArrayRectangular _
(numberOfRows, numberOfColumns, _
    numberOfLevels, distanceBwtnRows, _
    distanceBwtnColumns, distanceBwtnLevels)
ZoomAll
End Sub

```

在三维中镜像

用 **Mirror3D** 方法，你可以沿着一个由三点指定的镜像平面对一个对象进行镜像。



在三维中镜像

以下的例子在模型空间建立了一个方体。然后将方体相对一个平面做镜像，再将镜像后的对象涂成红色。

```

Sub Ch8_MirrorABox3D()
' 创建方体对象
Dim boxObj As Acad3DSolid
Dim length As Double

```

```

Dim width As Double
Dim height As Double
Dim center(0 To 2) As Double
center(0) = 5#: center(1) = 5#: center(2) = 0
length = 5#: width = 7: height = 10#

' 在模型空间中创建方体(三维实体)对象
Set boxObj = ThisDrawing.ModelSpace. _
AddBox(center, length, width, height)

' 用三点定义镜像平面
Dim mirrorPt1(0 To 2) As Double
Dim mirrorPt2(0 To 2) As Double
Dim mirrorPt3(0 To 2) As Double

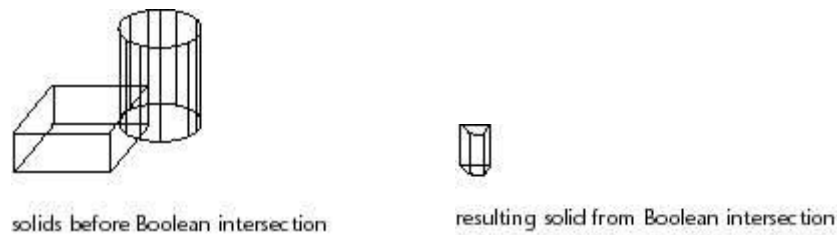
mirrorPt1(0) = 1.25: mirrorPt1(1) = 0: mirrorPt1(2) = 0
mirrorPt2(0) = 1.25: mirrorPt2(1) = 2: mirrorPt2(2) = 0
mirrorPt3(0) = 1.25: mirrorPt3(1) = 2: mirrorPt3(2) = 2

' 镜像方体
Dim mirrorBoxObj As Acad3DSolid
Set mirrorBoxObj = boxObj.Mirror3D _
(mirrorPt1, mirrorPt2, mirrorPt3)
mirrorBoxObj.Color = acRed
ZoomAll
End Sub

```

编辑三维实体

一旦你建立了一个实体，你可以通过将实体联合来建立更复杂的形状。你可以连接实体，互相切割实体或找出它们的公共体积(干涉部分)。可以用 **Boolean** 或 **CheckInterference** 方法来建立这些结合体。



以下的子程序在模型空间建立了一个方体和一个圆柱。然后找出它们的干涉部分，再利用干涉部分建立一个新的实体。为了方便观察，方体的颜色为白色，圆柱为青色，干涉实体为红色。

找出两个实体的干涉部分

以下的例子在模型空间建立了一个方体和一个圆柱。然后找出它们的干涉部分，再利用干涉部分建立一个新的实体。为了方便观察，方体的颜色为白色，圆柱为青色，干涉实体为红色。

```
Sub Ch8_FindInterferenceBetweenSolids()  
    ' 定义方体  
    Dim boxObj As Acad3DSolid  
    Dim length As Double  
    Dim width As Double  
    Dim height As Double  
    Dim center(0 To 2) As Double  
    center(0) = 5: center(1) = 5: center(2) = 0  
    length = 5  
    width = 7  
    height = 10  
  
    ' 在模型空间中创建方体对象并将其设为白色  
    Set boxObj = ThisDrawing.ModelSpace. _  
        AddBox(center, length, width, height)  
    boxObj.Color = acWhite
```

’ 定义圆柱体

```
Dim cylinderObj As Acad3DSolid
Dim cylinderRadius As Double
Dim cylinderHeight As Double
center(0) = 0: center(1) = 0: center(2) = 0
cylinderRadius = 5
cylinderHeight = 20
```

’ 创建圆柱体并置其为青色

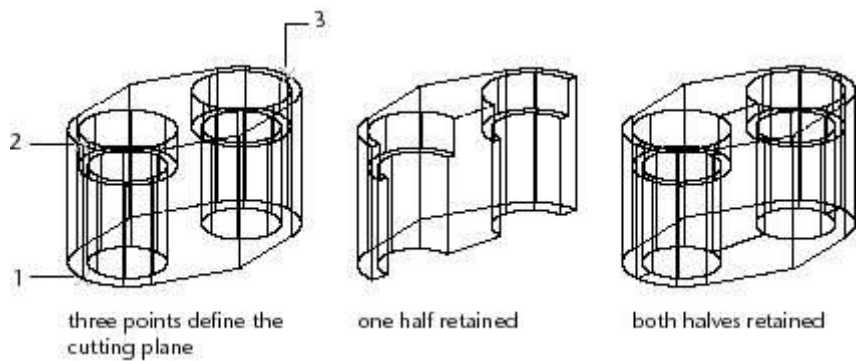
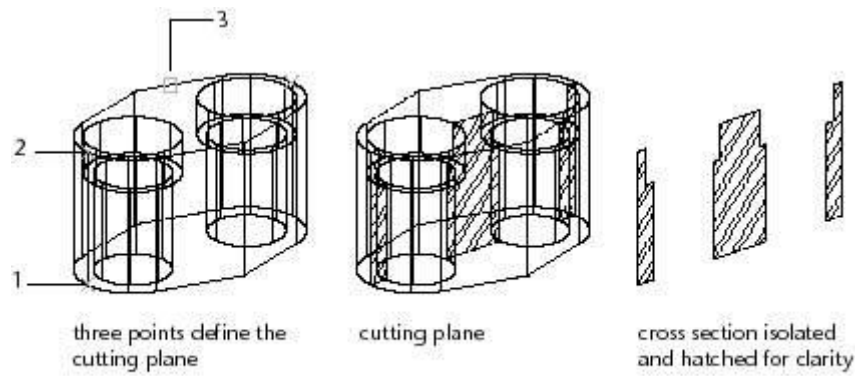
```
Set cylinderObj = ThisDrawing.ModelSpace.AddCylinder _
    (center, cylinderRadius, cylinderHeight)
cylinderObj.Color = acCyan
```

’ 查找两实体的干涉部分并建立新的实体。同时将其置为红色

```
Dim solidObj As Acad3DSolid
Set solidObj = boxObj.CheckInterference(cylinderObj, True)
solidObj.Color = acRed
ZoomAll
End Sub
```

可以通过获得一个实体的二维切面或用一个薄片将实体切为两部分来对这些实体进行修改。用 **SectionSolid**

d 方法可找到实体的切面，而 **SliceSolid** 方法可将一个实体用薄片切为两部分。



将一个实体切为两个实体

以下的例子在模型空间建立了一个方体。然后用一个有三点定义的平面来切这个方体。切成的部分成为一个三维实体。

```
Sub Ch8_SliceABox()
```

```
’ 定义方体对象
```

```
Dim boxObj As Acad3DSolid
```

```
Dim length As Double
```

```
Dim width As Double
```

```
Dim height As Double
```

```
Dim center(0 To 2) As Double
```

```
center(0) = 5#: center(1) = 5#: center(2) = 0
```

```
length = 5#: width = 7: height = 10#
```

’ 在模型空间中创建方体(三维实体)对象

```
Set boxObj = ThisDrawing.ModelSpace. _  
AddBox(center, length, width, height)  
boxObj.Color = acWhite
```

’ 用三个点定义切面

```
Dim slicePt1(0 To 2) As Double  
Dim slicePt2(0 To 2) As Double  
Dim slicePt3(0 To 2) As Double
```

```
slicePt1(0) = 1.5: slicePt1(1) = 7.5: slicePt1(2) = 0  
slicePt2(0) = 1.5: slicePt2(1) = 7.5: slicePt2(2) = 10  
slicePt3(0) = 8.5: slicePt3(1) = 2.5: slicePt3(2) = 10
```

’ 对方体进行切片并将新实体置为红色

```
Dim sliceObj As Acad3DSolid  
Set sliceObj = boxObj.SliceSolid _  
(slicePt1, slicePt2, slicePt3, True)  
sliceObj.Color = acRed  
ZoomAll  
End Sub
```

如网面一样，除了在隐藏、设阴影或渲染之外，实体一般被显示为线框。另外，你可以分析这些实体的普遍特征(体积、瞬间惯性、重心等等)。可以用以下特性去分析实体：**MomentOfInertia**、**PrincipalDirections**、**PrincipalMoments**、**ProductOfInertia**、**RadiiOfGyration** 和 **Volume**。

ContourlinesPerSurface 属性控制了用于显现线框的曲面部分的格状线的数目。**RenderSmoothness** 属性用来调整阴影物体和有隐藏线对象的光滑度。

第九章 定义布局及打印

当你用 **AutoCAD** 建立了你的图纸之后，你通常会将它打印出来。打印出来的图可以只包含视图，也可以包含视图更多的综合内容。在图纸空间，你可以建立称为移动视口的窗口，它显示了图纸的不同视图。根据你的需要，你可以打印出一个或几个视口的图纸，或者建立选项来决定该打印哪个视口的图以及这个图象如何与纸张相配。

了解模型空间和图纸空间

模型空间是你为你的模型建立几何形状的绘图环境。通常来说，当你开始在模型空间绘图时，你必须先指明你的图纸的界限以便来决定图纸环境的范围，而且你要用真实的长度单位来绘图。

图纸空间表现的是你的模型将被打印在图纸上的效果。在图纸空间中，你可以对你的图设置不同的视图，相互独立地按比例显示，也可以在图形中安排不同的视图。对一张图可以用许多不同的图纸空间来表现。

了解布局

你的图中所有的几何形状都被包含在布局中。模型空间的几何体是被包含在一个称为 **Model** 的单一布局中的。你不能重命名模型空间的布局，你也不可以建立另一个模型空间布局。每一张图只能有一个模型空间布局。

图纸空间的几何体也被包含在布局中。你可以在你的图中有许多不同的图纸空间布局，每一个对应不同的外形来打印。你可以改变图纸空间布局的名称。

在 **ActiveX Automation** 中，**ModelSpace** 对象包含了在模型空间布局的所有几何体。因为一张图中可以有不只一个的图纸空间布局，所以 **PaperSpace** 对象只指向最后一个活动的图纸空间布局。

本节主要内容：

了解布局与块的关系

了解打印配置

决定布局设置

了解布局与块的关系

任何布局的内容都被分布在两个不同的 **ActiveX** 对象之中：**Layout** 对象和 **Block** 对象。**Layout** 对象包含了打印设置和当布局出现在 **AutoCAD** 用户界面时的视觉属性。**Block** 对象包含了布局中的几何形体。

每一个 **Layout** 对象仅仅与一个 **Block** 对象相关联。运用 **Block** 属性可以访问一个与给定的布局相关联的 **Block** 对象。相应地，每一个 **Block** 对象也仅仅与一个 **Layout** 对象相关联。运用该块的 **Layout** 属性可以访问一个与给定的块相关联的 **Layout** 对象。

了解打印配置

当 **PlotConfiguration** 对象与 **Layout** 对象都包含同样的打印信息时，两者是类似的。不同之处在于 **Layout** 对象是与一个包含要打印的几何体的块对象相关联。而 **PlotConfiguration** 并不与一个特定的块对象相关联。它只是一个对任何几何体可用的打印设置的已命名的集合。

决定布局设置

布局设置控制了最后的打印输出。这些设置包括纸张大小、打印比例、打印区域、打印原点和打印设备的名称。了解如何运用布局设置可以确保打印出理想的图纸。所有的布局设置能够通过 **Layout** 对象的属性和命令来改变。

本节主要内容：

选择纸张尺寸和打印单位

调整打印原点

设置打印区域

设置打印比例

设置线宽比例

设置打印设备

选择纸的尺寸和打印单位

纸张大小的选择决定于你的系统的打印设备。每一种不同的绘图仪都对能用的纸张大小有一个标准列表。

你可以通过 **CanonicalMediaName** 属性来为一个布局选择纸张大小。

你也可以用 **PaperUnits** 属性来为你的布局指定打印单位。这个属性有个三选一的值，它们是：

acInches(英寸), **acMillimeters**(毫米), 或 **acPixels**(像素)。如果你的绘图仪被配置成光栅输出，那么你输出的尺寸必须指定为像素。

调整打印原点

打印原点在指定打印区域的左下角，它是由 **PlotOrigin** 属性控制的。通常情况下，打印原点设为(0, 0)。

但是，你也可以通过将 **CenterPlot** 属性设为 **TRUE** 来将图打印在纸的中心。将图设在图纸的中心则打印原点也随之改变。

设置打印区域

当你准备打印一个布局时，你可以指定打印区域来决定图中要包括哪些内容。要指定一个打印区域，可用 **PlotType** 属性，这个属性需要输入以下之一的变量：

acDisplay

打印当前模型空间所显示的所有内容。当在一个图纸空间布局打印的时候，这个选项是不能用的。

acExtents

打印落在当前选定空间边界内的所有内容。

acLimits

打印当前空间界限内的所有内容。

acView

打印由 **ViewToPlot** 属性命名的视图的内容。

acWindow

打印由 **SetWindowToPlot** 方法指定的窗口的所有内容。

acLayout

打印落在指定图纸大小界限内的所有内容。当要模型空间打印时，这个选项不可用。

当你建立了一个新的图纸空间布局时，默认的选项是 **acLayout**。

设置打印比例

通常来说，你会按对象的真实尺寸来绘图。当你打印图纸时，你可以用准确的比例来打印，也可以按图纸的大小来缩放打印。要指定一个打印比例，可以输入一个标准比例也可以输入一个自定义比例。

要输入一个标准的比例，首先要设 **UseStandardScale** 属性为 **TRUE**。然后你就可以用 **StandardScale** 属性来输入一个你想要的比例。

要输入一个自定义的比例。首先要设 **UseStandardScale** 属性为 **FALSE**。然后你就可以用 **SetCustomScale** 命令来输入一个自定义的比例。

当你只是想看看一个以前的草图时，通常是不需要用精确比例的。你可以用 **StandardScale** 属性的 **acScaleToFit** 值来在图纸上尽可能大地显示布局。

设置线宽比例

线宽可以按打印的比例相应地按比例显示。一般来说，线宽指定了打印对象的线粗细，而且会按线的宽度来打印而不会理会打印比例的。通常，你在打印一个布局时会用默认的打印比例 **1: 1**。然而，如果你想将一个 **E** 尺寸的图打印在一张 **A** 尺寸的纸上，你就可以按新的打印比例指定线宽。

要想线宽按比例显示，需将 **ScaleLineweights** 属性设为 **TRUE**，如果你不需要线宽按比例显示，那就设这个属性为 **FALSE**。

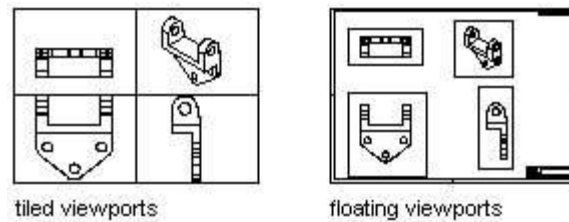
设置打印设备

打印设备的名称是在 **ConfigName** 属性中指定的。你可以将它设为你的系统中任何一个有效的设备名称。

如果你不设这个属性，图将被输送到你的系统的默认打印设备上。

了解视口

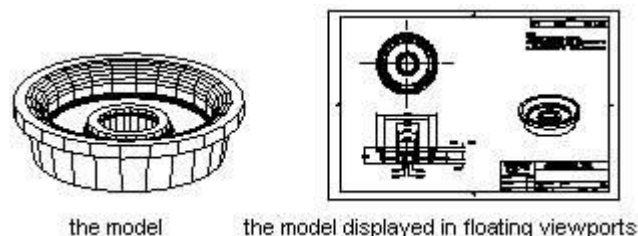
当你在模型空间工作时，你是在平铺视口(相对于 **ActiveX Automation** 中的 **Viewport** 对象)中绘制几何体。你可以同时显示一个或多个视口。如果显示了多个平铺视口，在一个视口中的编辑将会影响所有其它视口。然而，你可以单独对每一个视口进行放大倍率、观察点、栅格和捕捉设置。



在图纸空间，你通过浮动的图纸空间视口(相对于 **ActiveX Automation** 中的 **Pviewport** 对象)来包含你模型中的不同视图。浮动视口被作为对象来看待，你可以移动、调整大小和定形以建立一个合适的布局。你也可以直接在图纸空间视图中绘制对象，例如标题图块或注解，而不会影响模型本身。

你不能在图纸空间中编辑模型。要访问 **Pviewport** 对象中的模型，使用 **ActiveSpace** 属性从图纸空间切换至模型空间。这样你就可以在保持整个布局可见的情况下对模型进行操作。在 **Pviewport** 对象中，编辑和视图更改性能和 **Viewport** 对象中大致一样。然而，你可更多地控制单独的视图。例如，你可在某些视口中冻结或关闭图层而不影响其它视口。你可打开或关闭整个视图显示。你也可在视口和缩放视图之间相对于整个布局来对齐视图。

以下例图显示了模型和显示于图纸空间的视图的不同。每一图纸空间的图像代表一个带有不同视图的 **Pviewport** 对象。在一个视图中，标注图层被冻结。注意在图纸空间中所绘制的标题图块、边框和注解不会在模型空间视图中显示出来。同样，包含视口边框的图层也被关闭。



当你在 Viewport 对象中操作，ActiveSpace 属性必须被设置为 acModelSpace。当你在 Pviewport 对象中操作，你可设置 ActiveSpace 属性为 acModelSpace 或 acPaperSpace 中的一个，因此它允许你在需要时可在图纸空间和模型空间中切换。

Pviewport 对象、视口对象、ActiveSpace 属性设置

视口类型	状态	用法
Pviewport	ActiveSpace = acPaper space	通过建立浮动视口来安排布局，并增加标题块、边界和注解。 编辑不会影响模型。
Pviewport	ActiveSpace = acModel space	在浮动视口中编辑模型或改变视图。你可以在单独的视口中关闭或冻结布局。
Viewport	ActiveSpace = acModel space	将屏幕切换为平铺视口以便编辑模型的不同视图。

在 AutoCAD ActiveX Automation 中，ActiveSpace 属性是用来控制 TILEMODE 系统变量的。

设置 ThisDrawing.ActiveSpace = acModelSpace 是和设置 TILEMODE = on 相同的,而设置 ThisDrawing.ActiveSpace = acPaperSpace 也等同于设置 TILEMODE = off。

同样的，Mspace 属性在 AutoCAD 中同时是 MSPACE 和 PSPACE 命令的等效物。设置 ThisDrawing.MSpace = TRUE 等同于用 MSPACE 命令：它转换到了模型空间。设置 ThisDrawing.MSpace = FALSE 等同于用 PSPACE 命令：它转换到了图纸空间。

另外，你在设置 Mspace 属性为 TRUE 之前必须用 Display 方法。这个 Display 方法初始化了某些必须在切换至模型空间之前设置的图解设置。在 AutoCAD 中，这个是在后台完成的。然而，在 ActiveX Automation 界面中，程序员必须进行这个初始化的操作。

注意：记住，在你可以设置 Mspace 属性为 TRUE 之前，你必有用 Display 方法为至少一个 Pviewport 对象打开显示。如果没有打开显示将会通过 Mspace 属性导致一个异议出现。

本节主要内容：

切换至图纸空间布局

切换至模型空间布局

建立图纸空间视口

改变视口的视图及内容

在图纸空间按比例缩放阴影图案的线型

在被打印的视口中消隐线

切换至图纸空间布局

从模型空间，可以用以下步骤切换至最后活动的图纸空间布局。

1 设置 `ActiveSpace` 属性为 `acPaperSpace`:

```
ThisDrawing.ActiveSpace = acPaperSpace
```

2 切换 `Mspace` 属性为 `FALSE`:

```
ThisDrawing.MSpace = FALSE
```

当你处于图纸空间时，AutoCAD 会将图纸空间的 UCS 图标显示在图形区的左下角。十字光标指示出图纸空间布局区域(并不是这个视口的视图)现在可以编辑了。

切换至模型空间布局

从图纸空间，可以用以下步骤切换至模型空间的浮动视口或模型空间的平铺视口。

切换至浮动视口

1 用 `Display` 方法将绘图设置初始化:

```
ThisDrawing.ActivePViewport.Display
```

```
TRUE
```

2 切换 `Mspace` 属性为 `TRUE`:

```
ThisDrawing.MSpace = TRUE
```

以上可以使你处于模型空间的浮动视口。

注意：在你想切换至模型空间之前，你必须建立浮动视口。

要切换至平铺视口，还要加上以下步骤：

设 **ActiveSpace** 属性为 **acModelSpace**:

```
ThisDrawing.ActiveSpace = acModelSpace
```

建立图纸空间视口

图纸空间视口是由 **AddPViewport** 方法建立的。这个方法需要输入一个中心点及新视口的宽度和高度。在建立新视口之前，要用 **ActiveSpace** 属性将图纸空间设为当前的空间(通常是将 **TILEMODE** 设为 0)。

在建立了一个 **Pviewport** 对象之后，你可以设置这个视图自己的属性，例如观察方向(**Direction** 属性)、透视图的焦距(**LensLength** 属性)以及栅格显示(**GridOn** 属性)。你也可以控制视口自己的属性，例如图层(**Layer** 属性),线型(**Linetype** 属性),以及线型比例 (**LinetypeScale** 属性)。

从模型空间切换至图纸空间

以下的例子将图切换至图纸空间，建立一个浮动视口，设置视图，并激活视口。

```
Sub Ch9_SwitchToPaperSpace()  
    ' 设置活动的空间为图纸空间  
    ThisDrawing.ActiveSpace = acPaperSpace  
  
    ' 建立图纸空间视口  
    Dim newVport As AcadPViewport  
    Dim center(0 To 2) As Double  
    center(0) = 3.25
```



```
center(1) = 3
center(2) = 0
Set newVport = ThisDrawing.PaperSpace. _
AddPViewport(center, 6, 5)

' 改变视口的视图方向
Dim viewDir(0 To 2) As Double
viewDir(0) = 1
viewDir(1) = 1
viewDir(2) = 1
newVport.direction = viewDir

' 激活视口
newVport.Display = True

' 切换至模型空间
ThisDrawing.MSpace = True

' 设置 newVport 为当前视口
' (通常情况下不是必须这样做但却是个好主意)
ThisDrawing.ActivePViewport = newVport

' 在模型空间中范围缩放
ZoomExtents

' 关闭模型空间编辑状态
ThisDrawing.MSpace = False

' 在图纸空间中范围缩放
```

```
ZoomExtents
```

```
End Sub
```

前述的代码中步骤的次序是重要的。通常来说，事情必须按它们在 **AutoCAD** 命令行发生的顺序来做。唯一例外的动作包括定义视图及激活视口。

注意：要想设置或改变视图的性质(视图方向、焦距等等)，必须将视口对象的 **Display** 方法设为关(**FALSE**)，并且在你能设置当前的视口之前，必须先将 **Display** 方法设为开(**TRUE**)。

建立四个浮动视口

以下的例子选自"从模型空间切换至图纸空间"，接着上面的程序建立了四个浮动视口，并分别地将每个视口的视图设为顶视、前视、右视和等轴视。每一个视图按图纸空间的比例的一半来按比例显示。为了确保在每一个视口都能看到效果，你最好在试这个程序之间建立一个三维实体。

```
Sub Ch9_FourPViewports()  
Dim topVport, frontVport As AcadPViewport  
Dim rightVport, isoVport As AcadPViewport  
Dim pt(0 To 2) As Double  
Dim viewDir(0 To 2) As Double  
ThisDrawing.ActiveSpace = acPaperSpace  
ThisDrawing.MSpace = True  
' 取得现有的 Pviewport 并将其设为 topVport  
pt(0) = 2.5: pt(1) = 5.5: pt(2) = 0  
Set topVport = ThisDrawing.ActivePViewport  
' No need to set Direction for top view  
topVport.center = pt  
topVport.width = 2.5  
topVport.height = 2.5  
topVport.Display True  
ThisDrawing.MSpace = True  
ThisDrawing.ActivePViewport = topVport
```

```

ZoomExtents
ZoomScaled 0.5, acZoomScaledRelativePSpace
' 建立并设置 frontVport
pt(0) = 2.5: pt(1) = 2.5: pt(2) = 0
Set frontVport = ThisDrawing.PaperSpace. _
AddPViewport(pt, 2.5, 2.5)
viewDir(0) = 0: viewDir(1) = 1: viewDir(2) = 0
frontVport.direction = viewDir
frontVport.Display acOn
ThisDrawing.MSpace = True
ThisDrawing.ActivePViewport = frontVport
ZoomExtents
ZoomScaled 0.5, acZoomScaledRelativePSpace
' 建立并设置 rightVport
pt(0) = 5.5: pt(1) = 5.5: pt(2) = 0
Set rightVport = ThisDrawing.PaperSpace. _
AddPViewport(pt, 2.5, 2.5)
viewDir(0) = 1: viewDir(1) = 0: viewDir(2) = 0
rightVport.direction = viewDir
rightVport.Display acOn
ThisDrawing.MSpace = True
ThisDrawing.ActivePViewport = rightVport
ZoomExtents
ZoomScaled 0.5, acZoomScaledRelativePSpace
' 建立并设置 isoVport
pt(0) = 5.5: pt(1) = 2.5: pt(2) = 0
Set isoVport = ThisDrawing.PaperSpace. _
AddPViewport(pt, 2.5, 2.5)
viewDir(0) = 1: viewDir(1) = 1: viewDir(2) = 1

```

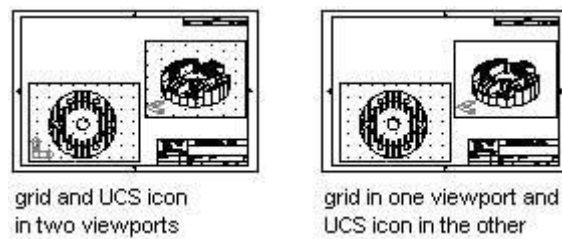
```

isoVport.direction = viewDir
isoVport.Display acOn
ThisDrawing.MSpace = True
ThisDrawing.ActiveViewport = isoVport
ZoomExtents
ZoomScaled 0.5, acZoomScaledRelativePSPACE
' 完成：在所有视口中执行一次 regen
ThisDrawing.Regen True
End Sub

```

改变视口视图及内容

要在一个视口对象中改变视图，你必须处在模型空间并且这个视口是活动的。



在一个浮动视口中编辑图纸

- 1 在模型空间中，设置 **ActiveViewport** 属性使视口成为活动的。

```
ThisDrawing.ActiveViewport = MyViewportObject
```

2 编辑图纸

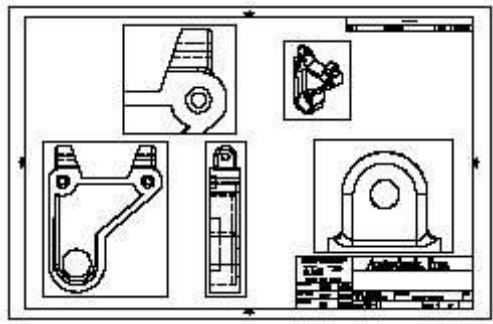
你也可以在图纸空间建立一个对象，例如注解、尺寸或标题块。但是，你必须先设置 **ActiveSpace** 属性为 **FALSE**，然后用 **Mspace** 属性转入图纸空间。在图纸空间建立的对象只有在图纸空间才能用。

本节主要内容：

相对于图纸空间缩放视图

相对于图纸空间缩放视图

在你打印之前，你可以为每个视图建立一个比例缩放因子。相对于图纸空间按比例缩放视图就为每个被显示的视图建立了一个一致的比例。例如，以下的图例显示了一个有几个视口的图纸空间视图-每个设为不同的比例和视图。要想准确打印缩放的视图，你必须相对于图纸空间按比例缩放每个视图，而不是相对于先前的视图或相对于完整比例的模型。



当你位于图纸空间时，比例因子表示的是被打印的图纸和在视口中显示的模型的真实尺寸之间的比率。要得到这个比例，必须将图纸空间和模型空间划分开。例如，要得到一个 **1: 4** 的图纸，你必须指定一个比例因子为一个图纸空间单位对四个模型空间单位(**1: 4**)。

用 **ZoomScaled** 方法可以相对于图纸空间单位按比例缩放视口。这个方法需要输入三个变量：要进行缩放的视口，比例因子和如何应用这个比例因子。第三个变量是可选择的，它决定了如何应用这个比例因子：

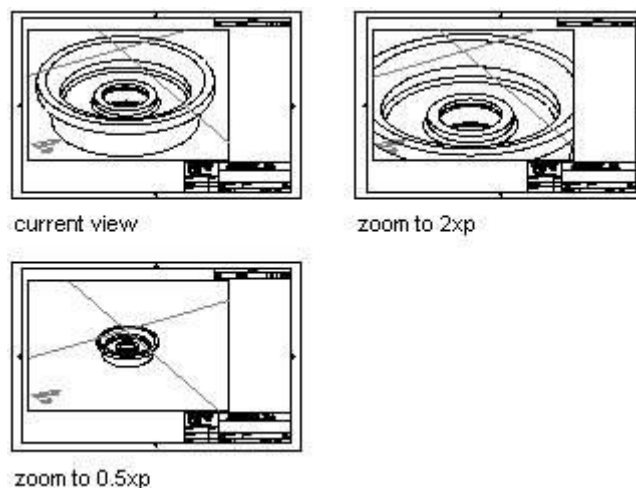
相对于图纸的界限

相对当前的视图

相对于图纸空间单位

要想指定这个缩放比例相对于图纸空间单位，就对这个变量输入 **acZoomScaledRelativePSPACE** 常数。

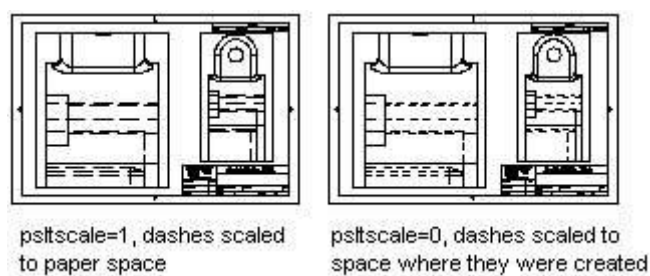
如图所示，如果你相对于图纸空间单位输入缩放比例为 **2**，在视口中的比例就增加为图纸空间单位的尺寸的两倍。如果你相对于图纸空间单位输入缩放比例为 **0.5**，在视口中的比例就变为图纸空间单位的尺寸的一半。这个模型就以它真实尺寸的一半来打印了。



在图纸空间中缩放线型样式

在图纸空间中，你可以用两种方法比例缩放任何类型的线型。这个比例可以基于对象被建立的空间(模型空间或图纸空间)的图纸单位。线型比例也可以以图纸空间单位为基础成为一个统一的比例。你可以用 **PSLTSCALE** 系统变量去维持对象在不同视口用不同的缩放比例显示时用相同的线型。它也会影响在三维视图中线的显示。

在以下图示中，在模型空间中线的线型样式是通过 **PSLTSCALE** 系统变量在图纸空间中统一地按比例缩放的。注意在这两个视口中即使对象有不同的缩放比例，但是线型的缩放比例是相同的。



用 **SetVariable** 方法可以设置 **PSLTSCALE** 系统变量的值。

在被打印视口中的消隐线

如果你的图包含三维表面、网面、突出物、表面或实体，当你打印这个视口时你可以在指定的视口中消除隐藏线。

用给出视口的 **RemoveHiddenLines** 属性可以在图纸空间视口(**PViewport** 对象)隐藏图纸空间的图中的线。这个属性接受一个 **Boolean** 值。输入 **TRUE** 可以消除图中的隐藏线，输入 **FALSE** 可以在图中显示出隐藏线。

用 **PlotHidden** 属性可以在模型空间视口(视口对象)隐藏模型空间的图中的线，这个属性可以在 **Layout** 对象中找到。这个属性接受一个 **Boolean** 值。输入 **TRUE** 可以消除图中的隐藏线，输入 **FALSE** 可以在图中显示出隐藏线。

打印图纸

你可以在模型空间打印你的图纸，也可以打印你在图纸空间已准备好的图纸。当你在建立一个图纸空间布局之前观察或检验你的图时，最好就是在模型空间中先打印出来。一旦你的模型已经建立，你就可以准备和打印一个图纸空间布局。

打印与两个 **ActiveX Automation** 对象有关：布局对象和打印对象。布局对象包含了一个给定布局的打印设置。打印对象包含了发动和监控一个打印次序的方法和属性。

本节主要内容：

执行基本打印

在模型空间中打印

在图纸空间中打印

执行基本打印

从 **Plot** 对象中你可以用以下的方法及属性：

PlotToFile

输出至文件。

PlotToDevice

输出至绘图机或打印机

DisplayPlotPreview

显示一个指定图的预览

SetLayoutsToPlot

确定将在下一个步的 **PlotToFile** 或 **PlotToDevice** 中被打印的布局列表。

StartBatchMode

发动一个批打印

QuietErrorMode

切换打印错误报告的表彰错误模式

NumberOfCopies

指定打印份数。

BatchPlotProgress

得到当前批打印的状态，或终止批打印

SetLayoutsToPlot 方法必须在 **PlotToDevice** 或 **PlotToFile** 方法之前调用。如果 **SetLayoutsToPlot** 没有被调用，

或是输入一个 **NULL** 来调用，则当前活动的布局将被打印。

NumberOfCopies 指定了打印的份数。如果在每次 **PlotToDevice** 调用之前，

这个属性没有被重设，则用上次在 **NumberOfCopies** 属性中指定的值。

批量打印是用来提供支持 **BatchPlot** 功能的应用。在进行一个批量打印之前，先设 **QuietErrorMode** 为 **TRUE** 以便有一个连续的打印时间。用 **StartBatchMode** 方法可以开始一个批量打印任务。用 **BatchPlotProgress** 属性可以来检查批量打印的进展，或用它来终止批量打印。

在模型空间中打印

通常地，当你打印一张大图(如地形图)时，你可以指定一个缩放比例来将真实的图纸单位转换成被打印的英寸或毫米。然而，当你在模型空间打印时，如果没有设置指定打印到系统打印机，则会用默认值打印当前的显示:按图纸大小缩放，0 度旋转，偏移量为(0, 0)。要想改变打印设置，可以改变与模型空间相关联的 **Layout** 对象的属性。

打印一个活动的模型空间布局的范围

以下的例子首先检查确定当前活动的空间是模型空间。然后建立了几个打印设置。最后用 **PlotToDevice** 方法发送打印。

```
Sub Ch9_PrintModelSpace()  
    ' 检查当时空间是否为模型空间  
    If ThisDrawing.ActiveSpace = acPaperSpace Then  
        ThisDrawing.MSpace = True  
        ThisDrawing.ActiveSpace = acModelSpace  
    End If  
  
    ' 设置打印范围和打印区域的比例  
    ThisDrawing.ModelSpace.Layout.PlotType = acExtents  
    ThisDrawing.ModelSpace.Layout. _  
        StandardScale = acScaleToFit  
  
    ' 设置打印份数为 1  
    ThisDrawing.Plot.NumberOfCopies = 1  
  
    ' 开始打印  
    ThisDrawing.Plot.PlotToDevice  
End Sub
```

可以用 **ConfigName** 属性来指定打印设备的名称。在 **PlotToDevice** 方法中通过指定一个 **PC3** 文件，这个设备可以被取代。

从图纸空间打印

你可以同时在图纸空间打印一个或以上的图纸。如"从模型空间打印"中示范，你可以打印当前活动的布局图，或者也可以通过指定布局的名称来打印。

打印两个图纸空间的布局图

以下的例子将两个图纸空间的布局图("Layout1"和"Layout2")发送到默认的打印设备。注意这两个布局必须以这个代码在图中运行。以下的例子首先建立了一个包含被打印的布局名称的字符数组。然后用该数组输入到 **SetLayoutsToPlot** 方法中。接着设置被打印的份数，最后将图发送到默认的打印设备。

```
Sub Ch9_PrintPaperSpace()  
    ' 建立输出用的图纸空间布局  
    Dim strLayouts(0 To 1) As String  
    Dim varLayouts As Variant  
    strLayouts(0) = "Layout1"  
    strLayouts(1) = "Layout2"  
    varLayouts = strLayouts  
    ThisDrawing.Plot.SetLayoutsToPlot varLayouts  
  
    ' 设置打印份数为 1  
    ThisDrawing.Plot.NumberOfCopies = 1  
  
    ' 开始打印  
    ThisDrawing.Plot.PlotToDevice  
End Sub
```

第十章-高级绘图与组织技术

用户获得一些经验后，可以利用 **AutoCAD?** 的许多高级功能进一步加强用户的应用程序。

可以在图形中包含光栅图像，例如航空拍摄图像、卫星拍摄图像、数字照片以及计算机渲染图像。关于光栅图像的详细信息（除了本节中的信息以外），请参见用户手册。

除了加强用户图形的视觉图像以外，**AutoCAD** 还提供了若干功能以协助用户组织数据，允许用户进一步扩充图形中对象的智能。

使用光栅图像

用户可以利用 **AutoCAD** 将光栅图像添加到基于矢量的 **AutoCAD**

图形中，然后查看和打印生成的文档。有许多理由可以让你将光栅图像放到矢量的图形中，包括扫描文档、传真、或胶片图形，使用卫星或数字图像，或创建水印或标志等效果，也可以添加一个计算机软件制作的渲染图像。

本节内容包括：

附着和缩放光栅图像

管理光栅图像

修改图像和图像边界

剪裁图像

附着和缩放光栅图像

可以将图像放到图形文件中，但它们实际上并不属于图形文件。图像通过路径名称或数据管理文档

ID 链接到图形文件。链接图像的路径可以随时进行更改或删除。要附着图像，请使用 **AddRaster** 方法在图形中创建

Raster 对象。这个方法需要输入四个值：要附着的图像文件的名称，在图形中放置图像的插入点，图像的

缩放比例，以及图像的旋转角度。请记住，**Raster**

对象代表独立的图像链接，而不是图像本身。

在用户附着图像之后，可以多次重新附着，为每个附件创建新的 **Raster**

对象。每个附件都有自己的剪裁边界以及亮度、对比度、褪色度和透明度设置。一个图像可以裁剪成许多块，并且在图形中分别排放。

在创建 **Raster** 对象时可以设置光栅图像缩放比例，使图像的几何图形比例与在 **AutoCAD** 图形中创建的几何图形的比例相匹配。选择要附着的图像时，图像将以

1 个图像测量单位对 1 个 **AutoCAD** 测量单位的缩放比例插入。要设置图像的缩放比例，用户需要了解图像上的几何图形比例以及要使用的测量单位（英寸、英尺等等），以便定义

1 个 **AutoCAD** 单位。图像文件必须包含定义 **DPI**（即，每英寸点数）以及图像中像素数目的分辨率信息。

如果图像包含分辨率信息，**AutoCAD** 会将此信息与用户在图形中提供用来缩放图像的缩放比例和

AutoCAD 测量单位结合起来。例如，如果光栅图像是扫描的设计图，比例为 1 英寸等于 50 英尺（即，1:600），而用户的

AutoCAD 图形被设置为 1 个单位代表 1 英寸，那么要设置该图像的缩放比例，应将 **AddRaster**

方法的 **ScaleFactor** 参数设置为 600。这样，**AutoCAD** 中插入图像的比例即可使图像中的几何图形与图形中的矢量几何图形对齐。

注意 如果没有在附着的图像文件中定义分辨率信息，**AutoCAD**

将以图像的初始宽度作为 1 个单位进行计算。插入图像之后，以 **AutoCAD** 单位表示的图像宽度即等于缩放比例。

附着光栅图像

本例在模型空间添加光栅图像。本例使用 **watch.jpg**（可在 **sample**

目录中找到此文件）。如果没有此图像，或者此图像位于其他目录，请为 **imageName** 变量插入有效的路径和文件名。

```

Sub Ch10_AttachingARaster()
Dim insertionPoint(0 To 2) As Double
Dim scalefactor As Double
Dim rotationAngle As Double
Dim imageName As String
Dim rasterObj As AcadRasterImage
imageName = "C:/Program Files/AutoCAD 2004/sample/watch.jpg"
insertionPoint(0) = 5
insertionPoint(1) = 5
insertionPoint(2) = 0
scalefactor = 2
rotationAngle = 0

On Error GoTo ERRORHANDLER
' 在模型空间中附着光栅图像
Set rasterObj = ThisDrawing.ModelSpace.AddRaster _
(imageName, insertionPoint, _
scalefactor, rotationAngle)
ZoomAll
Exit Sub
ERRORHANDLER:
MsgBox Err.Description
End Sub

```

管理光栅图像

用户可以使用 **Raster** 对象的特性管理光栅图像名、文件名和文件路径。

本节内容包括：

更改图像文件路径

命名图像

更改图像文件路径

可以使用 **ImageFile** 特性来查询或更改图像的路径和文件名。由此特性设置的路径是

AutoCAD 查找此图像的实际路径。

如果 **AutoCAD** 无法定位图形（例如，如果用户将文件移到其他目录中，与

ImageFile 特性中保存的目录不同），那么 **AutoCAD** 将删除名称中的相对或绝对路径信息（例如，使

`\\images\\tree.tga` 或 `c:\\my project\\images\\tree.tga` 变为

`tree.tga`），并搜索用 **Preferences** 对象的 **SetProjectFilePath** 方法定义的路径。如果图形不在此路径中，

AutoCAD

会尝试再次在第一个搜索路径中搜索。

可以删除文件名中的路径或通过重置 **ImageFile** 特性来指定相对路径。

更改 **ImageFile** 特性中的路径不会影响工程文件的搜索路径设置。

命名图像

图像名不一定要与图像文件名相同。在将图像附着到图形时，**AutoCAD**

将使用文件名（不带文件扩展名）作为图像名。可以更改图像名，这样做不会影响文件名。

图像文件由 **Raster** 对象的 **ImageFile** 特性表示。更改

ImageFile 特性将更改图形中的图像。图像名由 **Name** 特性表示，更改 **Name** 特性只会更改图像的名称，

而不会更改与其关联的文件。

修改图像和图像边界

所有图像都具有图像边界。将图像附着到图形时，图像边界将继承当前的特性设置，包括颜色、图层、线

型和线型比例。如果图像是两色图像，则图像颜色和边界颜色相同。

与对待其他 AutoCAD 对象相同，用户可以修改图像及其边界特性。例如，用户可以：

显示或隐藏图像边界

修改图像的图层、边界颜色和线型

更改图像位置

缩放、旋转和更改图像的宽度和高度

切换图像的可见性

更改图像的透明度

更改图像的亮度、对比度和褪色度

更改图像显示的质量和速度

本节内容包括：

显示和隐藏图像边界

更改图像的图层、边界颜色和边界线型

更改图像的比例、旋转、位置、宽度和高度

更改图像的可见性

修改两色图像的颜色和透明度

调整图像的亮度、对比度和褪色度

显示和隐藏图像边界

隐藏图像边界可以确保图像不会被意外地移动或修改，并且可以防止打印或显示边界。隐藏图像边界时，被剪裁的图像仍显示在指定的边界界限内，只有边界会受到影响。显示和隐藏图像边界将影响所有附着到图形的图像。

要显示或隐藏图像边界，请使用 **ClippingEnabled** 特性。

注意 此特性只影响图像边界。要在切换此特性时查看图像的变化，请仔细查看图像周围的小边界。

更改图像的图层、边界颜色和边界线型

可以使用以下特性更改图像边界的颜色和线型以及图像的图层：

Layer

指定图像的图层

Color

指定图像边界的颜色

Linetype

指定图像的线型

更改图像的比例、旋转、位置、宽度和高度

可以使用以下方法和特性更改图像的比例、旋转、位置、宽度和高度：

ScaleEntity

缩放图像

Rotate

旋转图像

Origin

指定图像位置

Width

以像素为单位指定图像的宽度

Height

以像素为单位指定图像的高度

ImageWidth

使用数据库单位指定图像的宽度

ImageHeight

使用数据库单位指定图像的高度

ShowRotation

确定是否旋转显示光栅

更改图像的可见性

通过在当前绘图任务中隐藏图像，图像可见性可以影响重画的速度。隐藏的图像既不会显示也不会打印，而只是显示图形边界。要隐藏图像，请将

ImageVisibility 特性设置为 **FALSE**。要重新显示图像，请将 **ImageVisibility** 特性设置为 **TRUE**。

修改两色图像的颜色和透明度

两色光栅图像是仅由前景色和背景色组成的图像。附着两色图像时，图像中的前景像素会继承当前图层的颜色设置。除了可以修改任何附着的图像以外，用户还可以通过更改前景色以及打开或关闭背景透明度来修改两色图像。

注意 两色图像和两色图像边界颜色通常相同。

要更改两色图像的前景色，请使用 **Color** 特性。要打开和关闭透明度，请使用

Transparency 特性。

调整图像的亮度、对比度和褪色度

可以在 **AutoCAD** 中将图像的亮度、对比度和褪色度调整为要显示的图像和打印输出，而不影响原始光栅图像文件。

使用以下特性可以调整亮度、对比度和褪色度：

Brightness

指定图像的亮度级别

Contrast

指定图像的对比度级别

Fade

指定图像的褪色度级别

剪裁图像

用户可以通过剪裁图像来定义用于显示和打印的图像的范围。剪裁边界必须是二维多边形或矩形，其顶点限制在图像边界之内。同一图像的多个实例可以具有不同的边界。

本节内容包括：

更改剪裁边界

显示和隐藏剪裁边界

更改剪裁边界

要更改现有的剪裁边界，只需重复前述步骤即可。旧边界将被删除，新边界将取代旧边界。

显示和隐藏剪裁边界

可以使用剪裁边界来显示剪裁的图像，也可以隐藏剪裁边界并显示原始图像边界。要隐藏剪裁边界并显示原始图像，请将

`ClippingEnabled` 特性设置为 `FALSE`。要显示剪裁的图像，请将 `ClippingEnabled` 特性设置为 `TRUE`。

剪裁光栅图像边界

本例在模型空间中添加光栅图像，然后根据剪裁边界来剪裁图像。本例使用 `downtown.jpg`（该文件可以在

`sample` 目录中找到）。如果没有此图像，或者此图像位于其他目录中，请为 `imageName` 变量插入有效的路径和文件名。

```
Sub Ch10_ClippingRasterBoundary()  
Dim insertionPoint(0 To 2) As Double  
Dim scalefactor As Double  
Dim rotationAngle As Double  
Dim imageName As String  
Dim rasterObj As AcadRasterImage  
  
imageName = "C:\AutoCAD\sample\downtown.jpg"  
insertionPoint(0) = 5  
insertionPoint(1) = 5  
insertionPoint(2) = 0  
scalefactor = 2
```

```

rotationAngle = 0

On Error GoTo ERRORHANDLER
' 在模型空间中创建光栅图像
Set rasterObj = ThisDrawing.ModelSpace.AddRaster _
(imageName, insertionPoint, _
scalefactor, rotationAngle)
ZoomAll

' 使用点数组建立剪裁边界
Dim clipPoints(0 To 9) As Double
clipPoints(0) = 6: clipPoints(1) = 6.75
clipPoints(2) = 7: clipPoints(3) = 6
clipPoints(4) = 6: clipPoints(5) = 5
clipPoints(6) = 5: clipPoints(7) = 6
clipPoints(8) = 6: clipPoints(9) = 6.75

' 剪裁图像
rasterObj.ClipBoundary clipPoints

' 启用剪裁显示
rasterObj.ClippingEnabled = True
ThisDrawing.Regen acActiveViewport
Exit Sub

ERRORHANDLER:
MsgBox Err.Description
End Sub

```

使用块和属性

AutoCAD 提供了若干功能，用来帮助用户管理图形中的对象。使用块可以将多个对象作为一个部件进行组织和操作。属性可以将信息项与图形中的块相关联，例如，部件号和价格。

使用 **AutoCAD** 外部参照，可以将整个图形附着或覆盖到当前的图形。打开当前图形时，参照图形中所做的任何更改都会出现在当前图形中。

关于使用块和属性的详细信息，请参见用户手册中的“绘制几何对象”。

本节内容包括：

使用块

使用属性

使用块

块是对象的集合，用户可以将它们关联起来，形成单一的对象或块参照。可以插入、缩放和旋转图形中的块参照。可以将块参照分解为其组成对象、修改块参照以及重定义块。**AutoCAD** 会根据块定义更新该块参照的所有未来实例。

对于使用不同的颜色和线型在不同的图层中绘制的对象，也可以从这些对象定义块。可以保留块中对象的图层、颜色和线型信息。这样，每次插入块时，就可以将块中的每个对象绘制在其原始图层上，并具有原来的颜色和线型。

关于使用块的详细信息，请参见用户手册中“绘制几何对象”中的“创建并插入符号（块）”。

本节内容包括：

定义块

插入块

分解块参照

重定义块

定义块

要创建块，请使用 **Add** 方法。这个方法需要输入两个值：图形中要添加块的位置和要创建的块的名称。

创建新块后，可以向其中添加任何几何对象或其他块。然后将块的实例插入到图形中。插入的块是对象，称为块参照。

也可以使用 **WBlock** 方法创建块，将对象编组到单独的图形文件中。这样，该图形文件就可以用作其他图形的块定义。**AutoCAD**

将插入到其他图形中的任何图形都看作块。

关于定义块的详细信息，请参见用户手册中“绘制几何对象”中的“创建块”。

插入块

可以使用 **InsertBlock** 方法将块或整个图形插入到当前图形中。**InsertBlock**

方法需要输入六个值：插入点，要插入的块或图形的名称，**X** 缩放比例，**Y** 缩放比例，**Z** 缩放比例，以及旋转角度。

将整个图形插入到另一个图形中时，**AutoCAD** 会将插入图形当作块参照处理。后续的插入操作会以不同的位置、不同的比例和不同的旋转设置来参照块定义（该块定义包含块的几何说明）。如果在插入后更改原始图形，这些更改将不影响插入的块。如果希望插入的块反映对原始图形所做的更改，则可以通过重新插入原始图形重定义块。此操作可以通过

InsertBlock 方法完成。

如果将图形作为块插入，将自动使用文件名作为块的名称。在创建块之后，可以使用

Name 特性更改块的名称。

默认情况下，AutoCAD 使用坐标 (0,0,0) 作为插入图形的基点。可以打开原始图形并使用

SetVariable 方法为 **INSBASE** 系统变量指定其他插入基点，从而更改图形的基点。下次插入图形时，AutoCAD

将使用新的基点。

如果插入的图形包含 **PaperSpace** 对象，当前图形的块定义中将不包含这些对象。要在其他图形中使用

PaperSpace 对象，可以打开原始图形，并使用 **Add** 方法将 **PaperSpace** 对象定义为块。可以将图形插入到其他图形的图纸空间或模型空间中。

不能通过遍历块参照的方式来查找组成它的原始对象。但是，可以遍历原始块定义，或者将块参照分解为其原始部件。

也可以使用 **AddMInsertBlock** 方法插入块的数组。此方法不会在图形中插入单一块（与

InsertBlock 一样），而是插入指定块的数组。此方法将返回 **MInsertBlock** 对象。

关于插入块的详细信息，请参见用户手册中“绘制几何对象”中的“插入块”。

定义块并将块插入到图形中

本例定义一个块并将一个圆添加到块定义中。然后，将块作为块参照插入图形。

```
Sub Ch10_InsertingABlock()
```

```
’ 定义块
```

```
Dim blockObj As AcadBlock
```

```
Dim insertionPnt(0 To 2) As Double
```

```
insertionPnt(0) = 0
```

```
insertionPnt(1) = 0
```

```
insertionPnt(2) = 0
```

```
Set blockObj = ThisDrawing.Blocks.Add _
```

```
    (insertionPnt, "CircleBlock")
```

```
’ 向块中添加圆
```

```

Dim circleObj As AcadCircle
Dim center(0 To 2) As Double
Dim radius As Double
center(0) = 0
center(1) = 0
center(2) = 0
radius = 1
Set circleObj = blockObj.AddCircle(center, radius)

```

’ 插入块

```

Dim blockRefObj As AcadBlockReference
insertionPnt(0) = 2
insertionPnt(1) = 2
insertionPnt(2) = 0
Set blockRefObj = ThisDrawing.ModelSpace.InsertBlock _
    (insertionPnt, "CircleBlock", 1#, 1#, 1#, 0)
ZoomAll
MsgBox "The circle belongs to " & blockRefObj.ObjectName
End Sub

```

注意

插入之后，外部文件的 **WCS** 会平行于当前图形中当前 **UCS** 的 **XY** 平面。因此，在插入块之前设置 **UCS**，可以按任何方向插入外部文件中的块。

分解块参照

使用 **Explode** 方法可以打断块参照。通过分解块参照，可以修改块，或者添加或删除定义块参照的对象。

显示分解的块参照的结果

本例创建一个块并向块定义添加一个圆，然后将块作为块参照插入到图形中。接着块参照被分解，最后显示分解过程中生成的对象及其对象类型。


```

Sub Ch10_ExplodingABlock()
' 定义块
Dim blockObj As AcadBlock
Dim insertionPnt(0 To 2) As Double
insertionPnt(0) = 0
insertionPnt(1) = 0
insertionPnt(2) = 0
Set blockObj = ThisDrawing.Blocks.Add _
(insertionPnt, "CircleBlock")

' 向块中添加圆
Dim circleObj As AcadCircle
Dim center(0 To 2) As Double
Dim radius As Double
center(0) = 0
center(1) = 0
center(2) = 0
radius = 1
Set circleObj = blockObj.AddCircle(center, radius)

' 插入块
Dim blockRefObj As AcadBlockReference
insertionPnt(0) = 2
insertionPnt(1) = 2
insertionPnt(2) = 0
Set blockRefObj = ThisDrawing.ModelSpace.InsertBlock _
(insertionPnt, "CircleBlock", 1#, 1#, 1#, 0)
ZoomAll
MsgBox "The circle belongs to " & blockRefObj.ObjectName

```

’ 分解块参照

```
Dim explodedObjects As Variant  
explodedObjects = blockRefObj.Explode
```

’ 遍历分解的对象

```
Dim I As Integer  
For I = 0 To UBound(explodedObjects)  
    explodedObjects(I).Color = acRed  
    explodedObjects(I).Update  
    MsgBox "Exploded Object " & I & ": " _  
        & explodedObjects(I).ObjectName  
    explodedObjects(I).Color = acByLayer  
    explodedObjects(I).Update  
Next  
End Sub
```

重定义块

要重定义块，请使用 **Block** 对象的任何方法和特性。重定义块时，图形中对该块的所有参照也随即更新，以便反映新的定义。

重定义会影响以前和以后的块插入。固定属性会丢失，并且被新的固定属性所替换。即使新块不包含属性，可变属性也保持不变。

在块定义中重定义对象

本例创建一个块并向块定义中添加一个圆，然后将块作为块参照插入到图形中。块定义中的圆将被更新，块参照也会自动更新。

```
Sub Ch10_RedefiningABlock()
```

’ 定义块

```

Dim blockObj As AcadBlock
Dim insertionPnt(0 To 2) As Double
insertionPnt(0) = 0
insertionPnt(1) = 0
insertionPnt(2) = 0
Set blockObj = ThisDrawing.Blocks.Add _
    (insertionPnt, "CircleBlock")

' 向块中添加圆
Dim circleObj As AcadCircle
Dim center(0 To 2) As Double
Dim radius As Double
center(0) = 0
center(1) = 0
center(2) = 0
radius = 1
Set circleObj = blockObj.AddCircle(center, radius)

```

```

' 插入块
Dim blockRefObj As AcadBlockReference
insertionPnt(0) = 2
insertionPnt(1) = 2
insertionPnt(2) = 0
Set blockRefObj = ThisDrawing.ModelSpace.InsertBlock _
    (insertionPnt, "CircleBlock", 1#, 1#, 1#, 0)
ZoomAll

```

```

' 重定义块中的圆，
' 并更新块参照

```

```
circleObj.radius = 3  
blockRefObj.Update  
End Sub
```

使用属性

属性参照为用户提供交互式标记或标签，以便将文字附着到块中。数据的样例包括部件号、价格、注释和所有者姓名等。

用户可以从图形中提取属性参照信息，并在电子表格或数据库中使用该信息以生成各种项（例如，部件列表或明细表）。如果每个属性参照都包含不同的标记，则可以将多个属性参照与块关联。也可以定义固定属性。由于对每个块引用，这些固定属性都具有相同的值，因此在插入块时，**AutoCAD**不提示用户输入值。

属性可以不可见，这意味着将不显示或打印属性参照。但是，关于属性参照的信息可以存储在图形文件中。

关于使用属性的详细信息，请参见用户手册中“绘制几何对象”中的“块属性概述”。

本节内容包括：

创建属性定义和属性参照

编辑属性定义

提取属性信息

创建属性定义和属性参照

要创建属性参照，首先必须使用 **AddAttribute** 方法在块上创建属性定义。这个方法需要输入六个值：属性文字的高度，属性模式，提示字符串，插入点，标记字符串，以及默认属性值。

模式值是可选的。可以输入五个常量来指定属性模式：

acAttributeModeNormal

指定保留每个属性的当前模式。

acAttributeModeInvisible

指定在用户插入块时不显示属性值。**ATTDISP** 命令将替代“不可见”模式。

acAttributeModeConstant

在插入块时赋予属性固定值。

acAttributeModeVerify

在插入块时，提示验证属性值是否正确。

acAttributeModePreset

当用户插入包含当前属性的块时，将属性设置为其默认值。在此模式下不能编辑该值。

可以输入无、任意组合或全部选项。要指定选项组合，请将各个常量相加。例如，可以输入

acAttributeModeInvisible + acAttributeModeConstant。

插入包含属性的块时，将显示提示字符串。此字符串的默认值是 **Tag** 字符串。在模式中输入

acAttributeModeConstant 可以禁用提示。

标记字符串可以标识每个属性引用。用户可以使用任何字符，但空格与叹号除外。**AutoCAD**

会将小写字符转换为大写。

在块中定义属性定义之后，每当用户使用 **InsertBlock** 方法插入块时，都可以为属性参照指定不同的值。

属性定义将与创建此属性定义的块关联。在模型空间或图纸空间中创建的属性定义不会附着到任何给定的块。

定义属性定义

本例创建一个块，然后将属性添加到块中。接着将块插入到图形中。

```
Sub Ch10_CreatingAnAttribute()
```

```
’ 定义块
```

```
Dim blockObj As AcadBlock
```

```
Dim insertionPnt(0 To 2) As Double
```

```
insertionPnt(0) = 0
```

```
insertionPnt(1) = 0
```

```
insertionPnt(2) = 0
```

```
Set blockObj = ThisDrawing.Blocks.Add _  
    (insertionPnt, "BlockWithAttribute")
```

```
’ 向块添加属性
```

```
Dim attributeObj As AcadAttribute
```

```
Dim height As Double
```

```
Dim mode As Long
```

```
Dim prompt As String
```

```
Dim insertionPoint(0 To 2) As Double
```

```
Dim tag As String
```

```
Dim value As String
```

```
height = 1
```

```
mode = acAttributeModeVerify
```

```
prompt = "New Prompt"
```

```
insertionPoint(0) = 5
```

```
insertionPoint(1) = 5
```

```
insertionPoint(2) = 0
```

```
tag = "New Tag"
```

```
value = "New Value"
```

```
Set attributeObj = blockObj.AddAttribute(height, mode, _  
    prompt, insertionPoint, tag, value)
```

’ 插入块，创建块参照

’ 和属性参照

```
Dim blockRefObj As AcadBlockReference
```

```
insertionPnt(0) = 2
```

```
insertionPnt(1) = 2
```

```
insertionPnt(2) = 0
```

```
Set blockRefObj = ThisDrawing.ModelSpace.InsertBlock _
```

```
(insertionPnt, "BlockWithAttribute", 1#, 1#, 1#, 0)
```

```
End Sub
```

编辑属性定义

可以使用 **Attribute** 对象特性和方法来编辑属性。

下面是属性的部分特性：

Alignment

指定属性的水平对齐和垂直对齐方式

Backward

指定属性文字的方向

FieldLength

指定属性的字段长度

Height

指定属性的高度

InsertionPoint

指定属性的插入点

Mode

指定属性的模式

PromptString

指定属性的提示字符串

Rotation

指定属性的旋转角度

ScaleFactor

指定属性的缩放比例

TagString

指定属性的标记字符串

以下方法可用来编辑属性：

ArrayPolar

创建环形阵列

ArrayRectangular

创建矩形阵列

Copy

复制属性

Erase

删除属性

Mirror

镜像属性

Move

移动属性

Rotate

旋转属性

ScaleEntity

缩放属性

重定义属性定义

本例创建一个块，然后向块中添加属性。接着将块插入到图形中。最后更新属性文字，使其反向显示。

```
Sub Ch10_RedefiningAnAttribute()
```

```
’ 定义块
```

```
Dim blockObj As AcadBlock
```

```
Dim insertionPnt(0 To 2) As Double
```

```
insertionPnt(0) = 0
```

```
insertionPnt(1) = 0
```

```
insertionPnt(2) = 0
```

```
Set blockObj = ThisDrawing.Blocks.Add _
```

```
(insertionPnt, "BlockWithAttribute")
```

’ 向块添加属性

```
Dim attributeObj As AcadAttribute
Dim height As Double
Dim mode As Long
Dim prompt As String
Dim insertionPoint(0 To 2) As Double
Dim tag As String
Dim value As String

height = 1
mode = acAttributeModeVerify
prompt = "New Prompt"
insertionPoint(0) = 5
insertionPoint(1) = 5
insertionPoint(2) = 0
tag = "New Tag"
value = "New Value"

Set attributeObj = blockObj.AddAttribute(height, mode, _
prompt, insertionPoint, tag, value)
```

’ 插入块，创建块参照

’ 和属性参照

```
Dim blockRefObj As AcadBlockReference

insertionPnt(0) = 2
insertionPnt(1) = 2
insertionPnt(2) = 0

Set blockRefObj = ThisDrawing.ModelSpace.InsertBlock _
(insertionPnt, "BlockWithAttribute", 1#, 1#, 1#, 0)
```

’ 重定义属性文字，使其反向显示。

```
attributeObj.Backward = True  
attributeObj.Update  
End Sub
```

提取属性信息

可以使用 **GetAttributes** 和 **GetConstantAttributes**

方法从图形中提取属性信息。**GetAttributes** 方法返回附着到块的属性参照数组及其当前值。**GetConstantAttributes**

方法返回附着到块或外部参照的固定属性数组。由此方法返回的属性是固定属性定义，而不是属性参照。

不需要样板文件来提取属性信息，而且也不会创建属性信息文件。要迭代属性参照数组，只需要使用属性参照的

TagString 和 **TextString** 特性来检查属性信息。

TagString 特性代表属性参照的单个标记。**TextString**

特性包含属性参照的值。

关于提取属性信息的详细信息，请参见用户手册中“绘制几何对象”中的“从块属性提取数据”。

获取属性参照信息

本例创建一个块，然后向块中添加属性。接着将块插入到图形中。然后返回属性数据，并在消息框中显示。

块参照中的属性数据将被更新，并再次返回和显示属性数据。

```
Sub Ch10_GettingAttributes()  
' 创建块  
Dim blockObj As AcadBlock  
Dim insertionPnt(0 To 2) As Double  
insertionPnt(0) = 0  
insertionPnt(1) = 0  
insertionPnt(2) = 0  
Set blockObj = ThisDrawing.Blocks.Add _
```

```
(insertionPnt, "TESTBLOCK")
```

’ 定义属性定义

```
Dim attributeObj As AcadAttribute
```

```
Dim height As Double
```

```
Dim mode As Long
```

```
Dim prompt As String
```

```
Dim insertionPoint(0 To 2) As Double
```

```
Dim tag As String
```

```
Dim value As String
```

```
height = 1#
```

```
mode = acAttributeModeVerify
```

```
prompt = "Attribute Prompt"
```

```
insertionPoint(0) = 5
```

```
insertionPoint(1) = 5
```

```
insertionPoint(2) = 0
```

```
tag = "Attribute Tag"
```

```
value = "Attribute Value"
```

’ 在块上创建属性定义对象

```
Set attributeObj = blockObj.AddAttribute _
```

```
(height, mode, prompt, _
```

```
insertionPoint, tag, value)
```

’ 插入块

```
Dim blockRefObj As AcadBlockReference
```

```
insertionPnt(0) = 2
```

```
insertionPnt(1) = 2
```

```
insertionPnt(2) = 0
Set blockRefObj = ThisDrawing.ModelSpace.InsertBlock _
    (insertionPnt, "TESTBLOCK", 1, 1, 1, 0)
ZoomAll
```

’ 获取块参照的属性

```
Dim varAttributes As Variant
varAttributes = blockRefObj.GetAttributes
```

’ 将属性标记和值移至

’ 要在 MsgBox 中显示的字符串中

```
Dim strAttributes As String
strAttributes = ""
Dim I As Integer
For I = LBound(varAttributes) To UBound(varAttributes)
    strAttributes = strAttributes + "Tag: " + _
        varAttributes(I).TagString + vbCrLf + _
        " Value: " + varAttributes(I).textString
Next
MsgBox "The attributes for blockReference " + _
    blockRefObj.Name & " are: " & vbCrLf _
    & strAttributes
```

’ 更改属性值

’ 注意：没有 SetAttributes。一旦包含

’ 变量数组，就拥有了对象。

’ 更改这些对象就会改变图形中的对象。

```
varAttributes(0).textString = "NEW VALUE!"
```

' 再次获取属性

```
Dim newvarAttributes As Variant
```

```
newvarAttributes = blockRefObj.GetAttributes
```

' 再次显示标记和值

```
strAttributes = ""
```

```
For I = LBound(varAttributes) To UBound(varAttributes)
```

```
strAttributes = strAttributes + "Tag: " + _
```

```
newvarAttributes(I).TagString + vbCrLf + _
```

```
" Value: " + newvarAttributes(I).textString
```

```
Next
```

```
MsgBox "The attributes for blockReference " & _
```

```
blockRefObj.Name & " are: " & vbCrLf _
```

```
& strAttributes
```

```
End Sub
```