

明经通道论坛翻译丛书

VISUAL LISP® DEVELOPER'S BIBLE

Ed. 2011

Visual LISP® 开发者宝典

(2011版)

David M. Stein

张晨南 译

明经极客出版社



VISUAL LISP[®] DEVELOPER'S BIBLE

Visual LISP[®] 开发者宝典

David M. Stein

张晨南 译

献给：明经通道的 Visual LISPer

明经极客出版社

*** 内 容 简 介 ***

本书针对 AutoCAD® 2011 进行了更新,主要讲述如何使用 Visual LISP® 进行编程。Visual LISP® 是用于基于 AUTODESK® AutoCAD® 产品的增强型 AutoLISP 编程环境。本书涵盖初级到中级编程技术,从函数、工程到名称空间、反应器和对话回调的一切,并提供了 Visual LISP® 和 DCL 语言以及 VLIDE 的全面参考。

责任编辑 张晨南

封面设计 张晨南

出版发行 明经极客出版社

网 址 <http://bbs.mjtd.com/thread-187166-1-1.html>

开 本 216 mm×279 mm

版 次 2023年2月17日发行 2023年2月17日印刷

定 价 334.00元

(本书只用于个人学习交流, 严禁用于商业用途)

版本历史

- 2002.05 • 首次公开发布
- 2002.06 • 第二次发布（第一次更新）
- 2002.08 • 第三次发布（第二次更新）
- 2002.09 • 第四次发布（一些小的更新和修正）
- 2002.10 • 第五次发布（一些小的更新）
- 2002.12 • 第六次发布（增加了 ObjectDBX、XRecord 等内容）
- 2003.03 • 第七次发布（为 AutoCAD® 2004 进行更新）
- 2003.10 • 第八次发布（增加了一些附加信息和代码示例）
- 2011.01 • 第九次发布（为 AutoCAD® 2011 进行更新）

关于作者



我出生于美国弗吉尼亚联邦（州），也在那里长大，在 IT 领域工作了 20 多年，担任过各种系统工程师和软件开发人员。我目前在一家支持 Microsoft® 平台服务的 IT 服务公司担任高级顾问。然而，我仍然被轻拍肩膀求助帮忙解决 AUTODESK® 部署，FlexLM/FlexNet 许可证服务器环境、活动目录、组策略、智能打包等问题，编写脚本和构建疯狂的软件，让一些人摸着下巴说“嗯……”。

在从事 IT 工作之前，我沉浸在 AUTODESK® 开发人员网络世界中大约十年。我的大部分工作涉及 AutoLISP、Visual LISP、少量 VBA、少量 .NET 以及各种外部接口，以实现美国国防造船业设计流程的自动化。在业余时间，我喜欢阅读、背包旅行、骑自行车、皮划艇、喝咖啡和啤酒（虽然不是同时）、没完没了喋喋不休地谈论毫无意义的废话，并尽可能多地陪伴家人。我从来没有空闲时间。

关于译者


一个从业桥梁工程多年的设计师，明经通道论坛 ID: VORMITTAG。其他没什么好说的，一个我行我素不折不扣的 Geeker。

译者的话

从 2022 年底产生反应这本书的念头产生，到这本书完全翻译完成，历时大约三个月。其实我本人已经有十几年时间没有再怎么摸过 Auto/Visual LISP 了，翻译这本书完全是为了测试一下我自己用 L^AT_EX3 写的一个 L^AT_EX 文档类——**transbook**。我的兴趣在这些年很多都放在了 L^AT_EX 排版上，所以接下来这里主要说一说本书排版样式的约定。

关于 LISP 代码的格式化问题，我非常同意 PETER SEIBEL 的看法，格式化 LISP 代码的关键在于正确缩进它，这一缩进应当反应代码的结构，而不需要用开闭括号的对齐来保证。一般而言，每一个新的嵌套层次都需要缩进一些，如果必须折行，位于同一个嵌套层次的项应该对齐。这样，一个需要跨越多行的函数调用可能写成这样：

```
(some-function arg-with-a-long-name
               another-arg-with-an-even-longer-name)
```

那些实现控制结构的宏和特殊形式在缩进上有所不同，“主体”元素相对整个形式的开括号缩进一个  或对应的空格（我是空格党，通常使用 2 或 4 个空格，视编辑器的情况而定），就像这样：

```
(defun multi-helloworld (/ i)
  (repeat (setq i 5)
    (princ
      (strcat
        "\n"
        (itoa (setq i (1- i)))
        " Hello World!")))))
```

结尾的)))) 可能看起来令人生畏，但代码一旦缩进正确，那么括号的意义就不那么重要了，没必要通过将他们分散在多行来突出代码结构，有经验的程序员也都会通过编辑器处理自动缩进和括号匹配。说实话，看到连续的闭括号占了十几行的代码真让我头疼，所以，你在本书中看到的代码示例都将是这个格式化风格。

本书中其他的一些排版约定说明如下：

- 除了标题样式外，本书正文中使用**粗体**表示强调。
- 除了特殊样式外，本书正文中西文用 `typewriter` 字体，中文用等距更纱字体表示的内容为一个特定的字符串。同时，这也是本书中程序代码的标准字体。
- 本书正文中提到的英文书名按英文的体例采用 *Italic* 体排印，如 *Practical Common Lisp*，外国人名采用小型大写字母方式排印，如 PETER SEIBEL。
- 书中将以 `LSP` 和 `acad.lsp` 的方式表示文件名或文件扩展名，一般来说，单独的文件扩展名使用大写的方式。如果带有路径，那表现形式是 `C:\Program Files\Autodesk\AutoCAD 2020\acad.exe`。`工具` `环境选项` `常规选项` 用于表达程序的菜单项，单独的菜单项可能是 `下一嵌套表达式`。你如果看到 `Ctrl+Alt+Del` 肯定知道这是组合键，而 `下一步` 就是对话框上的按钮。
- 下面这种格式表示的是 AutoCAD 命令行的操作：

```
Command: CHPROP
```

有时会带上命令或表达式的回显：

```
Command: (itoa 5)
"5"
```

- 操作系统里的命令长这样：

```
$ >sudo rm -rf /
```

• 本书常用的代码格式和样式你在上一页已经见过了，代码对 AutoLISP 函数、`vla-`系列函数、其他 `v1` 系列函数和作者自定义函数采用了语法高亮，你在书中可以看到。如果代码示例在文中需要交叉引用，或者代码本身就是一个完整的文件，亦或者代码是非 LISP 语言代码，则带有一个 Windows® 窗口风格的外框：

```

示例代码
(defun c:helloworld ()
  (princ "\nHello World!")
  (princ))
  
```

- 在非代码示例的环境中：
 - 可在 AutoCAD 命令行提示符下运行的命令是 `CHPROP`，`DIMSCALE` 则用来表示 AutoCAD 系统变量。

– 可在 Windows® 命令行提示符下运行的命令是 `COPY`；`%PROGRAMFILES%` 则用来表示 Windows® 系统变量，`HKEY_CURRENT_USER` 表示 Windows® 注册表项，有时也用 `HKCU` 来简化，而 `CurVer` 则表示注册表键。

– AutoLISP函数在本书中会以 `(princ)` 的形式来表示，`(vl-load-com)` 则用于表示 Visual LISP 函数，第三方程序定义的函数看起来是 `(dos_getstring)` 的样子，而本书作者的自定义函数则是 `(dump)`。

– 本书主要对 Visual LISP 函数进行解释，因此你会看到一些语法解释的内容，如果函数参数不是太多，通常用 `(vl-foo <foo1>[<foo2>])` 的方式进行解释，显然你应该能够看出来 `<foo1>` 是一个必备参数，而 `[<foo2>]` 是一个可选参数。如果函数参数太多，那么会用一个语法解释框来表示，效果如下：

```
(vlax-import-type-library
:tlb-filename <namestring>
:methods-prefix <mpfxstring>
:properties-prefix <ppfxstring>
:constants-prefix <cpfxstring>)
```

– 对于 ActiveX 和 DXF 中的一些特定的词语，译者以不同的样式表现出来：


- * 对象模型中的集合用 `AcadApplication` 表示；
- * 对象模型中的对象用 `Document` 表示；
- * 对象的属性用 `Layer` 表示；
- * 对象的方法用 `Add` 表示；
- * 对象的事件用 `Modified` 表示；
- * 枚举将以 `:vlax-vbarray`、`acRed` 的方式表示；
- * 反应器类型将以 `:VLR-AcDb-Reactor` 的方式表示；
- * DXF相关概念：如组码 0 对应的图元类型用 `INSERT` 表示，类名称用 `AcDbEntity` 表示。

– 本书中作者对一些计算机技术的词汇进行了解释，以词汇表的形式列在附录之后，正文中相应出现的位置都进行了链接，你可以点击跳到词汇表的相关页面查看这些术语解释。词汇表中也补充了一些译者认为有必要补充的词汇，同时，译者还对文中出现的缩略语进行了整理，功能和词汇表差不多。

– 最后，对于本书中所有出现解释的函数，译者对其作了索引，列在本书的最后，通过函数索引，你能很快定位到书中讨论它们的地方。

唠叨完这些你可能根本不关心的东西，再回到 Visual LISP 上来，由于本人的语言能力和技术水平都相当有限，译文中肯定会存在大量的错误，无论是在语言表达上，还是技术内容上。

当然，非常欢迎你向我指出。如果你真要拿这本书作为 Visual LISP 编程参考书，那我建议你还是尽量去仔细研读作者的原著。

最后关于版权问题的一些声明：本书仅用于个人学习，严禁用于商业用途。你在书中看到的丛书、出版社都是虚构的，至于定价，你仔细看看，那实际上是本书编译完成后的  PDF 文件的总页面数——334（包括了为保证每一章起始出现在迎面页而插入的空白页），这些不过是 transbook 文档类默认设置的一个 Feature。😊

引言

本书的目的在于帮助有经验的 AutoLISP 开发者更好地理解和使用 Visual LISP。什么？你的意思是 AutoLISP 和 Visual LISP 不是一回事？是的，他们不是一回事。本书将涉及的主题有 ActiveX、编译代码、调试代码、代码格式化，以及有效利用反应器和名称空间等高级的特性。AutoLISP 的基本内容就留给另外的书去讨论，因为那些内容在其它的书上都有很好的阐述。这本书仅着重讨论 Visual LISP 中扩展了 AutoLISP 的那些内容，以及 Visual LISP 所提供的独特的性能。

对于这本书，你可能需要用到 AutoCAD® 2011^①，或者其它包含了 Visual LISP 工具集的 AUTODESK® 产品，例如 AutoCAD® Map 3D、AutoCAD® Mechanical、或 AutoCAD® Civil 3D 等等。在这里你可能会纳闷：AutoCAD® LT 好像不行啊。^②

什么是 Visual LISP?

Visual LISP® 最初是 Basis Software Inc 开发的一款独立的软件产品，其原名是 *Vital LISP*。*Vital LISP* 拥有一批坚实的核心拥趸，受到了 AUTODESK® 的关注和尊重。在 AutoCAD® R14 的后期，AUTODESK® 公司购买了 *Vital LISP* 的整体版权，并将其更名为 Visual LISP®。接着它被作为一个独立的插件集成到 AutoCAD® R14 版本中。随着 AutoCAD® 2000 版本的发布，Visual LISP® 替换了旧的 Proteus AutoLISP 解释模块，从而成为了 AutoCAD® 整体中的一部分。后来，它做为 LISP 解释器被合并到所有基于 AutoCAD® 的垂直产品中，例如 AutoCAD® Map、AutoCAD® Mechanical、或 AutoCAD® Mechanical Desktop。

Visual LISP 并不是简单的取代了 AutoLISP，实际上，它除了能正常运行旧的 AutoLISP 代码外，还带有一些新的改良。这些改良包括一个内置的语法识别代码编辑器、对话框预览功

^① 译者注：因为重新排版，原书图片不是很清晰，有些程序界面实在找不到他们在 AutoCAD® 2011 中的样子了，这些程序界面本书中用 AutoCAD® 2020 对应的程序界面代替了。

^② 译者注：AUTODESK® 的有些产品是不包含 Visual LISP 的。而且，因为使用到了 ActiveX 技术，macOS 版本的 AutoCAD® 都无法使用 Visual LISP。

能、调试工具、代码格式化工具、在线开发参考、编译器以及编译向导以及工作空间项目管理等等。

不过对于这个语言来说，最有意义的改变应归于增加了 ActiveX 接口功能，这一功能的增加有效地使 Visual LISP 能与其它利用 ActiveX 技术的软件如 Visual Basic for Applications (VBA) 相提并论。虽然 Visual LISP 仍然缺少 VBA 所拥有许多复杂工具，但它确实具备了与 ActiveX 提供方和使用方连接的能力，例如 Microsoft® Office、Microsoft® Windows®，甚至 AutoCAD® 本身，而这些仅靠 AutoLISP 是不可能实现的。

当 AUTODESK® 将最初的 *Vital LISP* 修改为 Visual LISP® 时，大部分 *Vital LISP* 特性和功能并没有过多被修改。虽然 Visual LISP® 应该可以提高到一个更强大的开发平台，但相对于老土的 LISP 语言，AUTODESK® 公司似乎更喜欢其它的技术，如 VBA、ObjectARX 和 XML。^①

AUTODESK® 将 *Vital LISP* 移植成 Visual LISP® 时选择放弃了 *Vital LISP* 中的一些特性。这个不幸的错误导致了以后大多数的开发只能在 LISP 或 Visual LISP 下进行。虽然 *Vital LISP* 的一些特性仍存在于 Visual LISP® 中，但它的文件在 Visual LISP® 下是无法运行的，导致了它的一些特性在 Visual LISP® 中也无法运行。我只能指出这一点了，如果你碰巧是个老的 *Vital LISP* 用户，那你必须清醒地认识到这一点。其中的一些特性我在本书稍后会讲到。

书中使用的提示方式

本书中会使用一些特定的符号来提示某些指定的信息，例如……



这种方式将指一些没有出处或很难在别处找到的信息。



这种方式将表示一个愚蠢的，或者说毫无意义的信息，也可能是我因为一些愚蠢的原因提出来的。



这种方式指的是你为避免编程代码出错而需要特别注意的信息。

VLIDE 是 Visual LISP 集成开发编辑器 (Visual LISP Integrated Development Environment) 的简称 (E 也可以代表环境 (Environment) 或者其他什么词语)^②，它碰巧也是你要开

^① 译者注：现在，AUTODESK® 公司口味又发生了变化了，喜欢上了 .NET。

^② 译者注：我更倾向于 E 代表 Environment，翻译为 Visual LISP 集成开发环境，本书中的缩略语也按此处理，本处尊重原文先翻译为集成开发编辑器。

启 VLIDE 控制台时需要输入的命令 (**VLIDE**)。

“VLISP”将会作为 Visual LISP 的简称。我们很懒，任何以“VL”开头的东西都会让我们觉得有趣。就这样。^①

未来会怎样？

在我看来，Visual LISP 有两种未来：由 AUTODESK® 决定的未来的和潜在的不现实的未来。如果 AUTODESK® 能付出一些努力来弥补 VLIDE 和语言本身所缺少的东西，那么潜在的不现实的未来就是这样。事实上，他们可以扩展 Visual LISP 平台来作为 Powershell、VBScript 和 KiXtart 的替代品。我确实说过“不现实”，从技术上讲，没有什么能阻止这成为现实。现在，放下幻想并专注于现实，面对由 AUTODESK® 决定的 Visual LISP 未来：未老先衰^②。Visual LISP 代码在主机应用程序版本升级之间保持相对一致性和可移植性，不需要去感受 ObjectARX/ObjectDBX 甚至是 VBA 升级代码的痛苦。一些 Visual LISP 的代码更新是需要的，但几乎没有像在 ObjectARX 和 VBA 中那么重要（也没有重新编译的烦恼）。

余下的就是我在这本书的 2003 版里所说的：以我之愚见，Visual LISP 是开发与 CAD 应用程序相关的极端强大、灵活和活力十足的语言。如果加以重视，它可以做到比现在更好，只是现在看着有些营养不良（而且最近看来冰箱还是有点空……）。除非有一天有新的事物可以完全取代它而没有一点局限性，我将一直像用着其它的一些我每天必用的语言一样运用它。

① 译者注：我们并不是对作者的冒犯，作为译者的我们不犯懒，这个简称在全书中都将以 Visual LISP 的形式出现。其实对于使用 L^AT_EX 来排版本书的译者，生成这个词汇只需要在编辑器中输入 `\vlsp`，最多是 `\vlsp*`，后者会带上注册符号。好像我们的击键数量并没有增加，而且更方便了，对了，而且我们也是用 `vl` 开头，原作者会感兴趣的。

② 译者注：作者用的原文是“dying slowly on the vine”，直译是“在藤蔓上慢慢死去”，也就是说还没成熟就死掉了。译者水平有限，在中文中没有找到较好的译法，“夭折”似乎有点过了，于是我用了“未老先衰”。

目录

版本历史	i
关于作者	i
译者的话	iii
引言	vii
什么是 Visual LISP?	vii
书中使用的提示方式	viii
未来会怎样?	ix
1 Visual LISP 的开发环境	1
1.1 VLIDE 工具栏	2
1.2 VLIDE 下拉菜单	4
1.3 VLIDE 设置	4
2 Visual LISP 的基本编程	7
2.1 AutoLISP 与 Visual LISP/ActiveX 的比较	8
2.2 混合编码	10
2.3 嵌套对象引用	10
2.4 浏览对象属性和方法	11
2.5 ActiveX vs. DXF?	15
3 通过 Visual LISP 使用 ActiveX	17
3.1 类	17
3.2 对象	19
3.3 对象模型	19
3.4 类继承	20

3.5	集合与词典	20
3.6	属性、方法和事件	21
3.7	数据类型	25
3.8	名称空间	27
3.9	接口与类型库	28
4	在 Visual LISP 中调试代码	33
4.1	测试和调试工具	33
4.2	Visual LISP 错误捕获函数	41
5	使用工程和多文件	47
6	使用变体和安全数组	49
6.1	Visual LISP 变体函数	49
6.2	Visual LISP 安全数组函数	53
7	对象操作函数	59
8	文件与目录函数	63
9	迭代与字符串函数	69
10	使用名称空间	79
10.1	名称空间范围	80
10.2	名称空间函数	80
11	注册表函数	87
12	反应器与回调	91
12.1	Visual LISP 反应器函数	92
12.2	反应器类型	94
12.3	核实反应器类型	97
12.4	使用对象反应器	99
12.5	将数据附着到反应器对象	101
12.6	通过 VLIDE 检验反应器	101
12.7	查询反应器	102
12.8	临时反应器和永久反应器	102

12.9 打开包含永久反应器的文档	103
12.10 反应器与多重名称空间	103
12.11 反应器使用规则	104
13 创建 Visual LISP 应用程序	107
13.1 为什么要创建  VLX 应用程序	107
13.2 编译一个简单的应用程序	108
13.3  PRV 文件	112
14 在 Visual LISP 中使用 ObjectDBX	115
14.1 什么是 ObjectDBX	115
14.2 Visual LISP 中的 ObjectDBX	116
15 XDATA 和 XRecord	123
15.1 使用 XDATA	123
15.2 使用 XRecord 对象	123
16 AutoCAD 应用程序对象	129
17 AutoCAD 图元	133
17.1 图元通用属性与方法	133
17.2 各种图元的属性与方法	135
18 文档	171
18.1 Documents 集合	171
19 Preferences 对象	177
19.1 AcadPreferences	177
19.2 DatabasePreferences	182
19.3 重新加载配置文件	183
20 菜单和工具条	189
20.1 MenuBar 集合对象	189
20.2 MenuGroups 集合对象	191
20.3 MenuGroup 对象	192
20.4 PopupMenus 集合对象	192

20.5 PopupMenu 集合对象	192
20.6 Toolbars 集合对象	193
20.7 Toolbar 集合对象	193
20.8 功能区菜单	196
21 外部接口	197
21.1 Microsoft Excel	197
21.2 使用遗留的类型库	199
21.3 Windows ScriptControl	202
21.4 Windows Scripting Host	202
21.5 KiXart	204
21.6 FileSystem 对象	204
21.7 WMI and SWbemLocator	206
21.8 使用服务	208
22 在 Visual LISP 中使用 Visual Basic 的 DLL	209
23 使用对话框表单	215
23.1 引用 DCL 定义	215
23.2 动态对话框交互	216
23.3 从 DCL 回调控制图像	219
24 代码示例	221
24.1 示例 1 —获取图层属性列表	221
24.2 示例 2 —设置所有实体属性为 ByLayer	223
24.3 示例 3 —对所有打开的图形文件进行清理、核查并保存	223
24.4 示例 4 —对所有打开的图形文件缩放全景并保存	224
24.5 示例 5 —创建表格	225
25 AutoCAD® 2011 的变化	227
25.1 一般性变化	227
25.2 Visual Lisp 在 AutoCAD® 2011 中的变化	227
结论	229
A VLAX 枚举	231

B	VLA 函数名称	233
C	VLAX 函数名称	293
D	Visual LISP IDE 键盘快捷键	297
E	Visual LISP 中的小贴士和技巧	299
E.1	为 (autoload) 函数增加对 VLX 的支持	299
E.2	保存你的 VLIDE 配置	299
E.3	从 VLX 文件中恢复 DCL 代码	299
E.4	通过 生成应用程序 向导使用工程和 DCL	300
E.5	基于团队的 VLX 开发	300
F	有用的网络资源	301
	致谢	303
	词汇表	305
	缩略语	309
	函数索引	311

Visual LISP 的开发环境

VLIDE (Visual LISP 集成开发环境) 是一套组合工具, 它能使编程、测试、调试、编译输出更容易也更高效, 在 AutoCAD 命令行提示符下键入 **VLIDE** 来打开 Visual LISP 编辑器。此时将加载 Visual LISP ObjectARX 应用程序接口 (即 `vlide.arx`), 加载后在 AutoCAD 使用过程中 VLIDE 就可用。Visual LISP 是 AutoCAD® 的一部分, 因此不能在 AutoCAD® 以外的环境使用 VLIDE。



使用 AutoCAD 经典 工作空间 的用户可以使用 **工具 >> AutoLISP >> Visual LISP 编辑器** 调出 Auto/Visual LISP 编辑器。使用功能区菜单的用户可以在 **管理功能区** 选项卡的 **应用程序** 面板上找到 Visual LISP 编辑器。

注意在图 1.1 上显示的 VLIDE 界面。上部包含了 VLIDE 下拉菜单和工具栏, 中间部分包含了编译输出、Visual LISP 控制台和跟踪窗口。这里同时也是各已打开的程序的显示和编辑的窗口区域 (一个文件一个窗口), 其它没有显示在上面的窗口包括有监视窗口、对象检查和自动匹配窗口。



VLIDE 的一个有趣“特性”是, 每次关闭和重新打开 AutoCAD 时, 工具栏图标的颜色通常会发生变化。但是, 当 VLIDE 关闭并重新打开时则不会。颜色似乎在基本颜色中轮换: 红色、蓝色、绿色、黄色、洋红色、青色、黑色。如果你不喜欢这些颜色, 只需关闭 AutoCAD 并重新打开即可。这是自 VLIDE 多年前添加到 AutoCAD 以来的行为。什么原因, 谁知道呢? 将其视为一种奖励功能吧!

VLIDE 窗口底部边缘有状态栏。这里将显示 VLIDE 中每一动作的信息。右下角显示的是代码编辑器中光标的位置。它显示光标在代码文件中的当前位置, 其中 **L nnnnn** 为行号, 而 **C nnnnn** 是字符偏移位置。在本例中, 光标位于第 20 行的第 12 个字符处。

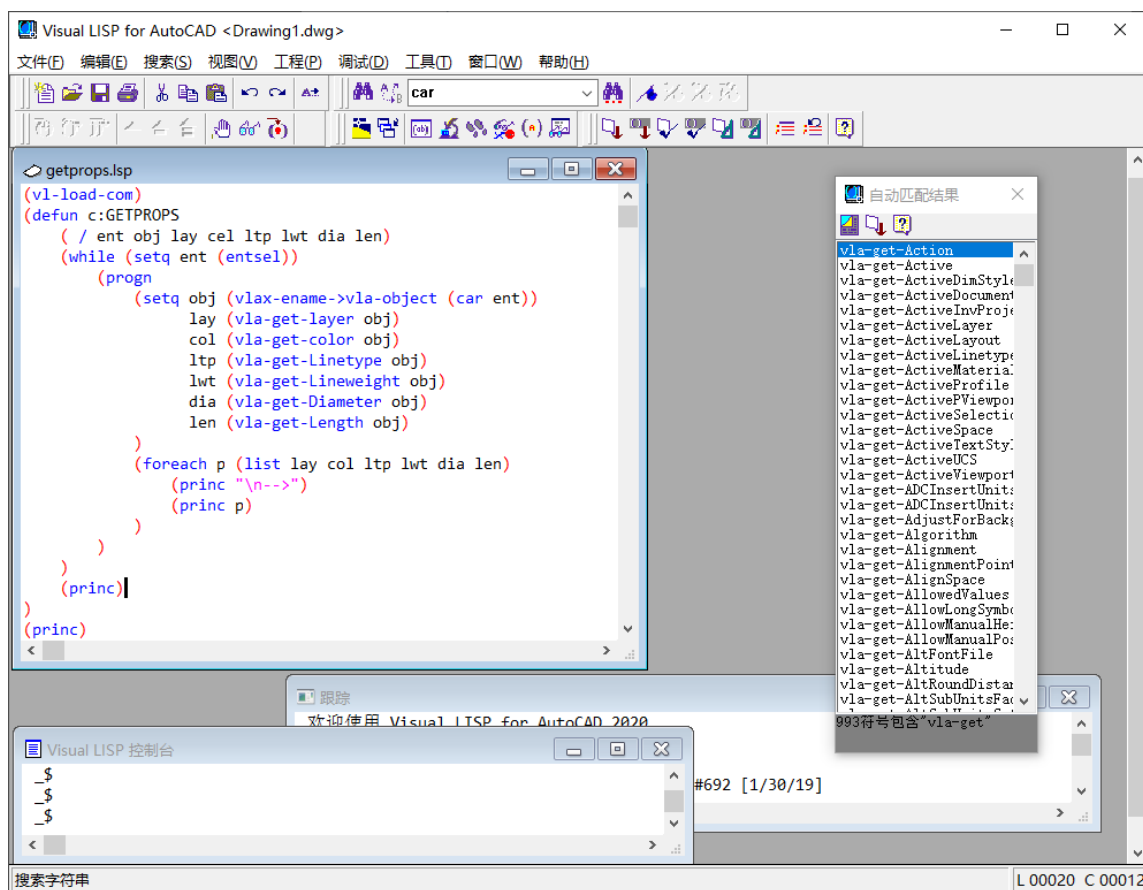


图 1.1 Visual LISP 集成开发环境

1.1 VLIDE 工具栏

这里有四个独立的 VLIDE 工具栏（图 1.2）^①，你可以对它进行移动、停靠或解除停靠（浮动），也可以根据你的喜好隐藏或显示其中的任何一个。工具栏并不完全与它们对应的下拉菜单的项目相对应，因此请注意不要假设工具栏上的所有内容都可以在 VLIDE 窗口中访问。你可能会发现下拉菜单更有效。

^① 译者注：实际上是有 5 个工具栏，**搜索** 工具栏作者并没有介绍。此外，出于排版的需要，译者将工具栏的图合并成为一个编号的插图。



图 1.2 VLIDE 工具栏

工具栏: 标准

标准 工具栏包括一般文件管理功能。这包括从左到右: • 新建文件 • 打开文件 • 保存文件 • 打印 • 剪切 • 复制 • 粘贴 • 放弃 • 重做 • 完成词语

工具栏: 工具

工具 工具栏包含一般编辑器功能。这包括从左到右: • 加载活动编辑窗口 • 加载选定代码 • 检查编辑窗口 • 检查选定代码 • 设置编辑窗口格式 • 设置选定代码格式 • 注释代码 • 取消注释代码 • 帮助

工具栏: 调试

调试 工具栏包含用于在受控执行期间测试和调试代码的工具。这包括从左到右: • 下一嵌套表达式 • 下一表达式 • 跳出 • 继续 • 退出 • 重置 • 切换断点 • 添加监视 • 上一断点代码



调试 工具栏最右边按钮的图标当前状态已被删除, 最后一个按钮仅显示一个视觉队列, 说明当前进程分步执行状态。

工具栏: 视图

视图 工具栏包含选项。这包括从左到右:

• 激活 AutoCAD (切换到 AutoCAD 编辑器) • 选择窗口^① • Visual LISP 控制台 • 检验 • 跟踪 • 符号服务 • 自动匹配 • 监视窗口

^① 译者注: 原文遗漏了这一项。

1.2 VLIDE 下拉菜单

在 VLIDE 窗口的默认状态中，VLIDE 下拉菜单总是可用的，它不象工具栏那样可以移动、隐藏等等。如同在以上部分提到的，在 Visual LISP 编辑器命令下的下拉菜单比工具栏包含了更多功能。因此，你会发现应用下拉菜单命令可以为你日常的编程工作提供很多便利。

菜单：文件

文件 菜单包含标准文件管理选项，例如 **打开文件**、**新建文件**、**保存**、**打印** 和 **退出**。它还提供其他有用的命令，如 **还原**、**全部关闭**、**全部保存** 和 **加载文件**。有关于 **生成应用程序** 的功能将在第 13 章创建 Visual LISP 应用程序稍后讨论。



图 1.3 文件 菜单



这是一个动态菜单，因此某些命令仅在相关时可见。

菜单：编辑

现在，在花时间查看下拉菜单并浏览各种命令后，你可能会有一种奇怪的感觉，即这些命令的位置并不符合逻辑。那是因为他们本身就是不合逻辑的。不过别担心，无论如何你都很少会用到其中的大部分。

1.3 VLIDE 设置

VLIDE 的设置主要通过 **工具** > **环境选项** 进行设置，如图 1.4 所示。

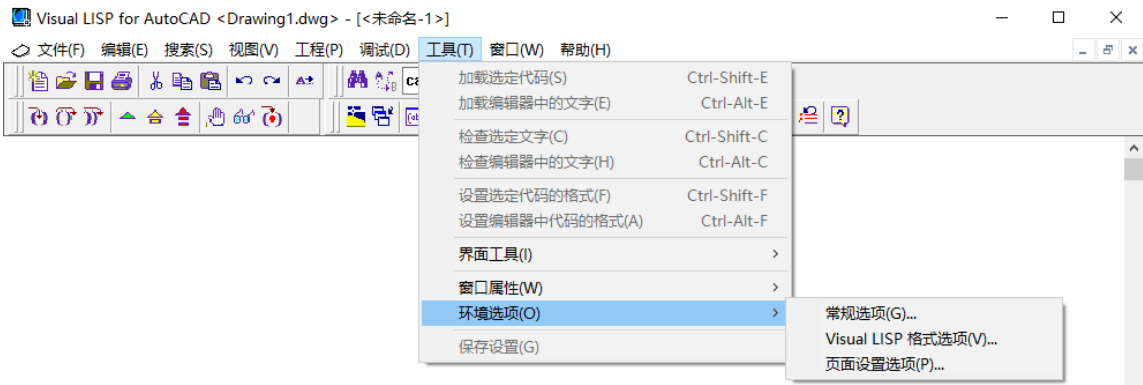


图 1.4 工具 >> 环境选项 菜单

常规选项

要为字体、颜色、布局、制表符和间距等配置 VLIDE 首选项，请使用 工具 >> 环境选项。常规选项对话框（图 1.5）提供了各种编辑器、桌面和符号管理选项。

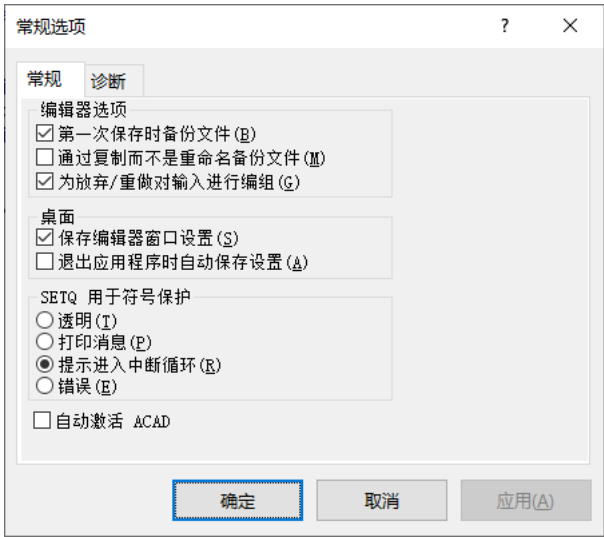


图 1.5 工具 >> 环境选项 >> 常规选项 对话框

Visual LISP 格式选项

图 1.6 中的对话框是你在基本对话框中单击 其他选项... 按钮时所看到的。



图 1.6 工具>环境选项>Visual LISP 格式选项 对话框

页面设置选项

如果你需要打印源代码，打印页面设置通过 工具>环境选项>页面设置选项 进行，如图 1.7 所示。^①

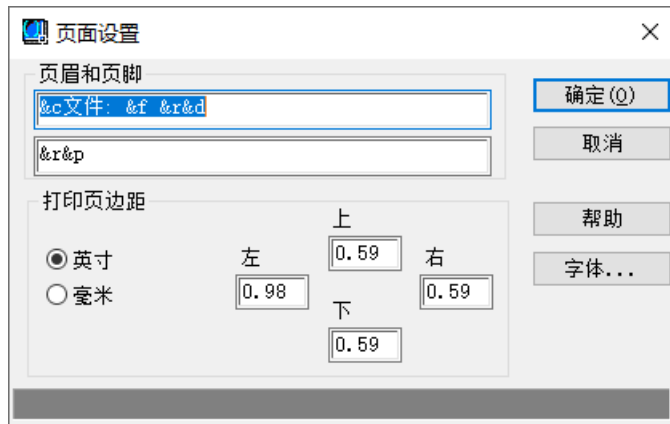


图 1.7 工具>环境选项>页面设置选项 对话框

^① 译者注:实际上原作者是没有这句话的,只是排版时在这一小节内完全没有文字内容,只有一个浮动插图,插图会跑到下一页去……所以,这段废话是我加上去的。

Visual LISP 的基本编程

在本章中我们将开始使用 Visual LISP 来编写一些基本代码，并通过一个例子学会编写、测试、调试和编译代码直到成为一个完整的程序。为了能与 CAD 程序员的期望保持一致，这里就不使用类似“Hello World”这种惯用代码^①来举例了。

(vl-load-com)

为了在 Visual LISP 中使用很酷的 ActiveX 功能，你首先需要用 (vl-load-com) 函数来初始化 ActiveX 接口。它可以被包含在每个文件或者每个函数的定义内，无所谓。一旦它被执行过，后续的重复调用也没有任何影响。

```
(vl-load-com)
(defun C:SHOWLAYER ( / ent)
  (while (setq ent (entsel))
    (princ
      (strcat "\nLayer: "
        (vla-get-Layer
          (vlax-ename->vla-object (car ent))))))
    (princ))
```

上面的代码演示了如何取得选定图元的图层并将其在命令行回显中显示出来，将这段代码加载到 AutoCAD 并在命令行中键入 **SHOWLAYER** 来运行。此时屏幕上会提示 **Select object:**^②，选中对象后该对象的图层名就会显示出来……

^① 译者注：据说，没有“Hello World”的编程书籍是不完整的，我想作者应该不这么看。

^② 译者注：L^AT_EX 排版设计的 **verbatim** 环境中只支持使用英文提示符，因此这里不对这个提示符进行翻译，中文版软件显示的应该是：选择对象：

```
Command: SHOWLAYER
Select object:
Layer: Alpha
```

与传统的通过 DXF 方式访问图层名有点不同，用户不需要知道在 DXF 中的组码 8 用于指定图层。而可以用 `(vla-get-Layer)` 来代替，这样更加直观。这也就是 Visual LISP 中 ActiveX 功能吸引人的关键：清晰。



你有很多种方法可以将 `(vl-load-com)` 添加到启动组中。比如说，你可以将其添加到 `acad.lsp` 或者 `acad.dcl` 文件中；也可以写一个小的 `LSP` 文件并通过 **APLOAD** 添加到启动组；还可以写入到任意一个用户界面 `MNL` 文件。在一个给定的会话中多次重复调用 `(vl-load-com)` 并不会有什么负面的影响。

PETER SEIBEL 说得好，他将 LISP 描述为一种**可编程的编程语言**，因为你始终可以用自己的方式来扩展和增强这种语言。举例来说，如果你习惯了其他语言的函数比如：`to-upper()` 或者 `Ucase()`，并且感到使用 `(strcase "dog")` 或者 `(strcase "dog" t)` 来转换字符串的大小写有些许的奇怪，你可以轻松地写一个自己的函数使用你更熟悉的别名。

```
(defun Ucase (myString)
  (strcase myString))

(defun Lcase (myString)
  (strcase myString t))
```

2.1 AutoLISP 与 Visual LISP/ActiveX 的比较

下面的两个语句在本质上是一样的，第一个语句稍显冗长，需要消耗多一点系统资源去执行，但增加的这些负担在多数情况下是可以忽略不计的。

```
Command: (vla-get-layer (vlax-ename->vla-object (car ent)))
Command: (cdr (assoc 8 (entget (car ent))))
```

不管以哪种方式取得对属性的初始访问 (`(entget)` 或 `(vlax-ename->vla-object)`)，接下来的工作用 ActiveX 方式来写会更简单一些。象这种情况，一般每次只进行一个对象的操作。例如……

```
(defun GETLAYER (entity / elist)
  (cdr (assoc 8 (entget entity))))
```

……和下面的例子在功能上是一样的……

```
(defun GETLAYER (entity / obj)
  (vla-get-Layer (vlax-ename->vla-object entity)))
```

无论如何, 这个比较并不全面, 因为它没有演示 ActiveX 对象模型如何允许在逻辑层面上去进行关联性的操作。例如, 以下的代码就显示了如何从 **PreferencesFiles** 对象中取得属性:

```
(vla-get-SupportPath
  (vla-get-Files
    (vla-get-Preferences (vlax-get-acad-object))))
```

以上的特性不可能仅靠 AutoLISP 完成, 需要通过 ActiveX 和 AutoCAD 的对象模型才能做到, 而实际上, Visual LISP 和 VBA 可以通过 AutoCAD 的 ActiveX 接口来访问这些特性。

让我们来看看另一个获取特定的直线 (**Line**) 图元属性的例子, 这个例子可以看到 ActiveX 接口怎样提供了方便的方法去理解这些名称, 从而使编码更直观:

```
(setq ent (car (entsel "\nSelect line object: ")))
(setq objLine (vlax-ename->vla-object ent))
(vla-get-Layer objLine)
(vla-get-Color objLine)
(vla-get-Lineweight objLine)
(vla-put-Layer objLine "0")
(vla-put-Color objLine acRed)
```

通过这两个例子可以了解到, 相对于神秘的 DXF 组码来说, 使用 ActiveX 能更直观地访问和编辑图元属性。同样, 值得注意的是, DXF 的 62 组码是短暂的, 而图元的 **Color** 属性才是持久的。换句话说: 一个带 **Color=ByLayer** 的图元在 **(entget)** 数据表中是没有 DXF 62 组码的。只有在将颜色添加并覆盖了默认层设置后, 这个图元才有 DXF 62 组码。然而, 如果使用 ActiveX 访问这样的图元, 即使是 **Color=ByLayer**, 返回的值就是 **acByLayer**。

做为展示开发者要如何使用的示例, 我们来看一看下面的函数, 这个函数可以从一个图元向另一个图元复制图层 ()、颜色 () 和线型 () 属性。

```
(defun CopyLayerColor1 (obj1 obj2)
  (vla-put-Layer obj2 (vla-get-Layer obj1)))
```

```
(vla-put-Color obj2 (vla-get-Color obj1)))
```

可以注意到我们并不需要依赖 DXF 组码，也不需要使用 `(subst)` 或 `(entmod)` 函数去更新图元属性。同样的函数用 AutoLISP 写的话就会被写成如下例这样：

```
(defun CopyLayerColor2 (ent1 ent2 / elist1 elist2 lay1 col1)
  (setq elist1 (entget ent1)
        elist2 (entget ent2)
        lay1 (cdr (assoc 8 elist1)))
  (setq elist2 (subst (cons 8 lay1) (assoc 8 elist2) elist2))
  (if (assoc 62 elist1)
      (progn
        (setq col1 (cdr (assoc 62 elist1)))
        (if (assoc 62 elist2)
            (setq elist2 (cons (cons 62 col1) elist2))
            (setq elist2 (subst (cons 62 col1) (assoc 62 elist2) elist2)))))
      (entmod elist2))
```

请注意，需要同时对源图元和目标图元的数据表进行 DXF 组码 62 的额外检查。如你所见，在执行许多常规任务中，Visual LISP 和 ActiveX 可以明显地减少代码的数量，减少了代码同时就减少了潜在的错误的发生，也就减少了相应的测试和调试代码。所有这些都能更快更简单更高效地完成高质量的代码，并且也会令用户更满意。

2.2 混合编码

虽然可以轻松地将传统的 AutoLISP 和较新的 Visual LISP 代码结合到给定的程序甚至是在给定的函数或语句中，但不建议这样做。事实上，在 Visual LISP 中混合诸如 `(entmake)` 和某些对象之类的东西可能会产生一些非常奇怪的问题。除非在极少数情况下，我强烈建议有意识地选择一种方式，而不是两种方式并用。对我来说，一个例外情况是 `(ssget)` 和各种选择集的函数，他们在古老的 AutoLISP 中比在 ActiveX 中使用 `SelectionSet` 对象的属性和方法时更容易实现，后者更像是 `Group` 对象而不是特定的集合。

2.3 嵌套对象引用

虽然在给定语句中嵌套对象引用很容易，而且完全允许，但有充分的理由不隐式地这样做。我的意思是，应该始终将对象引用存储在变量（符号）中，而不是简单地将其沿链向上传递到

下一个属性或方法调用。你会看到我在整本书中使用这样的嵌套代码；然而，我强烈建议不要那样做。请参考以下两个例子：

不好的示例：

```
(setq oFiles
  (vla-get-Files
    (vla-get-Preferences
      (vlax-get-Acad-object))))
```

好的示例：

```
(setq oFiles
  (vla-get-Files
    (setq oPrefs
      (vla-get-Preferences
        (setq oAcad (vlax-get-Acad-object)))))))
```

另一种好的示例：

```
(setq oAcad (vlax-get-Acad-object)
  oPrefs (vla-get-Preferences oAcad)
  oFiles (vla-get-Files oPrefs))
```

为什么这对任何人都很重要？好吧，当隐式传递对象引用时，实际上每次都在调用一个外部接口函数并将其丢弃，如果出于某种原因需要再次访问嵌套对象，就需要再次获取它，这样每次都在为处理增加开销；如果对象被保存为符号，那么就能以更少的处理开销重复访问，此外，这样还提供了对该对象引用生命周期的明确控制，用对象的说法，你可以决定何时丢弃它或“释放”它。

2.4 浏览对象属性和方法

如果你还没有发现 AutoCAD 中的对象提供的大量属性和方法，有一个好方法可以帮你熟悉它们，就是通过 `(vlax-dump-object)` 函数，这个函数需要一个参数，即所要显示属性的对象，还有一个可选的参数，它是一个标识（任何非 `nil` 值），代表了是否显示对象的方法。

```
(vlax-dump-object <object> [<show-methods>])
```

下面是导出 `AcadApplication` 对象内容的示例：

```
#<VLA-OBJECT IAcadApplication 01877c20>
```

```
Command: (vlax-dump-object acad t)

; IAcadApplication: An instance of the AutoCAD application
; Property values:
; ActiveDocument = #<VLA-OBJECT IAcadDocument 0f1e89b0>
; Application (RO) = #<VLA-OBJECT IAcadApplication 01877c20>
; Caption (RO) = "AutoCAD 2011 - [Drawing1.dwg]"
; Documents (RO) = #<VLA-OBJECT IAcadDocuments 11120b08>
; FullName (RO) = "C:\\Program Files\\Autodesk\\AutoCAD 2011\\acad.exe"
; Height = 726
; Hwnd (RO) = 983660
; LocaleId (RO) = 1033
; MenuBar (RO) = #<VLA-OBJECT IAcadMenuBar 111172b4>
; MenuGroups (RO) = #<VLA-OBJECT IAcadMenuGroups 0736a890>
; Name (RO) = "AutoCAD"
; Path (RO) = "C:\\Program Files\\Autodesk\\AutoCAD 2011"
; Preferences (RO) = #<VLA-OBJECT IAcadPreferences 110cd4bc>
; StatusId (RO) = ...Indexed contents not shown...
; VBE (RO) = AutoCAD: Problem in loading VBA
; Version (RO) = "18.1s (LMS Tech)"
; Visible = -1
; Width = 1020
; WindowLeft = 2
; WindowState = 1
; WindowTop = 2
; Methods supported:
; Eval (1)
; GetAcadState ()
; GetInterfaceObject (1)
; ListArx ()
; LoadArx (1)
; LoadDVB (1)
; Quit ()
; RunMacro (1)
; UnloadArx (1)
; UnloadDVB (1)
```



```
; Update ()
; ZoomAll ()
; ZoomCenter (2)
; ZoomExtents ()
; ZoomPickWindow ()
; ZoomPrevious ()
; ZoomScaled (2)
; ZoomWindow (2)
```

下例是在常规的圆（Circle）上使用该函数的结果：

```
Command: (setq ent (entsel)); pick a CIRCLE entity
```

```
Select Object: (<Entity name: 7ef035d8> (16.3954 9.95463 0.0))
```

```
Command: (setq obj (vlax-ename->vla-object (car ent)))
```

```
#<VLA-OBJECT IAcadCircle2 110f4404>
```

```
Command: (vlax-dump-object obj T)
```

```
; IAcadCircle2: AutoCAD Circle Interface
; Property values:
; Application (RO) = #<VLA-OBJECT IAcadApplication 01877c20>
; Area = 372.878
; Center = (12.6177 20.245 0.0)
; Circumference = 68.4524
; Diameter = 21.7891
; Document (RO) = #<VLA-OBJECT IAcadDocument 0f1e89b0>
; EntityTransparency = "ByLayer"
; Handle (RO) = "1B3"
; HasExtensionDictionary (RO) = 0
; Hyperlinks (RO) = #<VLA-OBJECT IAcadHyperlinks 1116f494>
; Layer = "0"
; Linetype = "ByLayer"
; LinetypeScale = 1.0
; Lineweight = -1
; Material = "ByLayer"
; Normal = (0.0 0.0 1.0)
; ObjectID (RO) = 2129671640
; ObjectName (RO) = "AcDbCircle"
```

```

; OwnerID (R0) = 2129665272
; PlotStyleName = "ByLayer"
; Radius = 10.8945
; Thickness = 0.0
; TrueColor = #<VLA-OBJECT IAcadAcCmColor 11159f90>
; Visible = -1
; Methods supported:
; ArrayPolar (3)
; ArrayRectangular (6)
; Copy ()
; Delete ()
; GetBoundingBox (2)
; GetExtensionDictionary ()
; GetXData (3)
; Highlight (1)
; IntersectWith (2)
; Mirror (2)
; Mirror3D (3)
; Move (2)
; Offset (1)
; Rotate (2)
; Rotate3D (3)
; ScaleEntity (2)
; SetXData (2)
; TransformBy (1)
; Update ()

```

如你所见，这是一个用于检查图元属性和方法的非常有用函数，它也同样有助于检查任何其它对象，包括应用程序对象、文档、集合等等。一个非常方便的技巧是定义一个实用函数，它将以更少的输入导出你指定的任何对象：

```

(defun dump (obj)
  (cond
    ( (= (type obj) 'ENAME)
      (vlax-dump-object (vlax-ename->vla-object obj) t))
    ( (= (type obj) 'VLA-OBJECT)
      (vlax-dump-object obj t))))

```

要使用它，只需在命令提示符下施展魔法，定义分配给符号的对象，按照以下一般方式导出它们……

```
Command: (setq e (car (entsel)))  
Select object: <pick something>  
... <blah blah>
```

```
Command: (setq o (vlax-ename->vla-object e))  
... <blah blah>
```

```
Command: (dump o)  
... <bang! Dumps all the goodies here...>
```

2.5 ActiveX vs. DXF?

ActiveX 能否胜任你在 Visual LISP 中处理的所有编程工作？不行。DXF 能完成 ActiveX 可以完成的所有工作吗？也不行。在很多情况下，古老的 AutoLISP 方法是给定问题的唯一解决方案，反之亦然。在某些情况下，你可以使用其中任何一种，但事实证明 AutoLISP 方法是最有效或最易于管理的选项。你可能会说：“哦，当然，这只是你的想法。”所以，那就让我们看看一些例子吧。

选择集

AutoLISP 或 Visual LISP 都可以用来创建和迭代遍历选择集，然而，你很快就会发现，在 AutoLISP 中处理选择集比在 Visual LISP 中要容易得多，也少很多问题。

点表

事实上，在 AutoLISP 中操作任何表结构都要比在 Visual LISP 操作来得容易，虽然两者都很强大和灵活，但用 LISP 构建和修改表结构比使用 Visual LISP 的数组结构更为简单。

实体属性

尽管大部分属性通过 Visual LISP 的 ActiveX 来访问比较容易也比较直接，但还是有些属性没有公开出来而只能通过 AutoLISP 的 DXF 组码值来读取。例如，线性标注对象 (`acDbRotatedDimension`) 的控制点，引线 (`Leader`) 对象的控制点。而象不常用的块定义 (`BLOCKDEF`)

的 `Description` 属性在 AutoLISP 和 Visual LISP 中都完全无法访问，它只能使用特定加载的函数库才能访问。

虽然 Visual LISP 增加了很多又好又强大的功能，但我在接下来的内容中还是会不厌其烦地举详细的例子来说明使用 AutoLISP 是很重要的。

谁才是赢家……

你！你是最大的赢家，因为你有更多的选择可以在更短的时间内实现你的目标，有更强大的选择，以及可预测和更可靠的结果。在 AutoCAD 编程领域中一个最古老的争论是哪种语言或 API 最“适合”使用。争论本身是愚蠢的，没有任何意义。管钳是“最好的”工具吗？打管道的时候也许是，但钉钉子的时候绝对不是。不同的工具适用于不同的工作。掌握你所需要的知识才能构建最佳产品。你会发现没有任何一种工具可以完成所有工作，除非你从一开始就啥也不做。

LISP 本身旨在提供非常严格的灵活性和可扩展性。与所有编程语言一样，你可以使用 LISP 完成在其他语言中无法完成的事情。例如，在 VBA 中定义一个递归的自我重定义函数就不是一件简单的事情，定义一个嵌套的函数结构也一样。LISP 中的表操作比其他语言更容易、更有效，因为这是 LISP 处理的问题的关键所在。一些读到这篇文章的人可能会反驳“你为什么要做那些事情呢？”小指高高扬起，啜饮着一杯十块钱的葡萄酒，窃笑起来。好吧，一旦发现运用这些能力所展示出的力量，你就会睁大了眼睛，大叫着“啊啊啊啊……，太酷了！”

虽然 Auto/Visual LISP 缺少许多像对话框表单工具这样的现代设施，但它确实提供了一种原始的 DCL 表单构建编程语言，用于构建石器时代的表单：蛮力编码。它的优雅是微妙而迟钝的，而不是明显的和艺术的，只有一丝薄荷的清新。

所以，最好的工具是什么？其实它们都是。好吧，可能除了 COBOL，但即使是 COBOL 也有它的用途。

通过 Visual LISP 使用 ActiveX

在这一章，我们将讨论在 Visual LISP 中使用 ActiveX 功能的更多示例。首先，我们将从 ActiveX 的技术环境开始，包括对象、对象模型、集合、属性、方法等。然后我们将深入研究 ActiveX 技术的某些细节。了解 ActiveX 功能对于运用任何使用它的语言都是必不可少的。

ActiveX 基本上是一种面向对象的媒介，这意味着它以使用对象和对象关系的方式运行。我并不打算深入解释面向对象的问题，这最好留给更有针对性的教科书。但是，为了获得基本的理解，我将尝试概述一些面向对象基本层面的问题。

3.1 类

面向对象环境中的一切都是从类开始。类是抽象框架或模板，用于描述对象应该采用什么形式以及它们应该如何表现和交互。类在某种意义上定义了对象类型的类别。例如，汽车可能是一类车辆。车辆可以是父类，而汽车则可以是子类。反过来，你可以更具体地定义额外的子类，例如旅行车、货车和跑车。

类不针对特定实例，在实例使用前就描述了各个方面。当使用一个类时，就称为调用该类的一个实例。调用一个类的结果通常是创建一个对象。对象可以是单个实体，也可以是其中包含更多对象的容器。

示例

类 机械工具

子类 动力辅助型，手动型

派生类 机械工具 / 动力辅助型 / 燃气动力型 / 起重

派生类 机械工具 / 手动型 / 手持型

派生与继承

使用像电脑文件夹这种树结构的类比来解释派生或子类可能是最容易的。当你向下深入到较低级别的“文件夹”时，通常会发现它是一个从一般到特定的虚拟模型。最顶层的“文件夹”很笼统，它下面的每一级子“文件夹”本质上逐渐变得更加具体。这很像推导。每个子类或派生类都是对其派生的更一般类的进一步细化或规范化。

每个派生类如何获取其父类的特性称为继承。这意味着子类包含派生它的父类的方方面面。这包括属性、方法和事件（Visual LISP 中的反应器类型）。这在使用 VB、VBA 和 C/C++/C# 等语言时更为相关和明显，它们使对象关系更易于查看和理解。相比之下，Visual LISP 并没有使这一点变得明显，作为程序员，你应该意识到必须依靠额外的学习和对知识结构的弥补才能完全理解将要处理的对象类关系。

虽然这似乎与本书意图相矛盾，但我强烈建议你在使用 Visual LISP 编程时保持 VBAIDE（图 3.1）为打开状态，从而可以快速访问 VBAIDE 中的对象浏览器。可以通过在 VBAIDE 中按 **[F2]** 打开对象浏览器并立即访问类文档、属性、方法、事件等。要是 Visual LISP® 包含对象浏览器就好了，可是你只能使用 `(vlax-dump-object)` 这把铲子去挖掘了，这些我们稍后再详细介绍。




图 3.1 VBA 对象浏览器



图形数据库浏览器不显示图元的 ActiveX/COM 方面的内容，它只显示遗留的 DXF 结构和内容。

3.2 对象

对象是类的实例。类描述对象必须支持什么才能成为其成员之一的规则。对象具有固有的属性，也可能具有固有的方法和事件。属性是定义对象行为或反应方式的特性；方法是用于访问或修改对象属性或某些行为的内置函数；事件是由对象发送的通知，以响应它们执行的特定操作或对它们执行的操作。

还是使用上面的汽车类示例，一个对象可能是一辆特定的汽车。例如，你的汽车具有独特的配置（品牌、型号、颜色、设定和车架号）。这就可以说你的汽车是汽车类（或者是从汽车类派生出的某个类，比如“垃圾车”类 ）的一个实例。

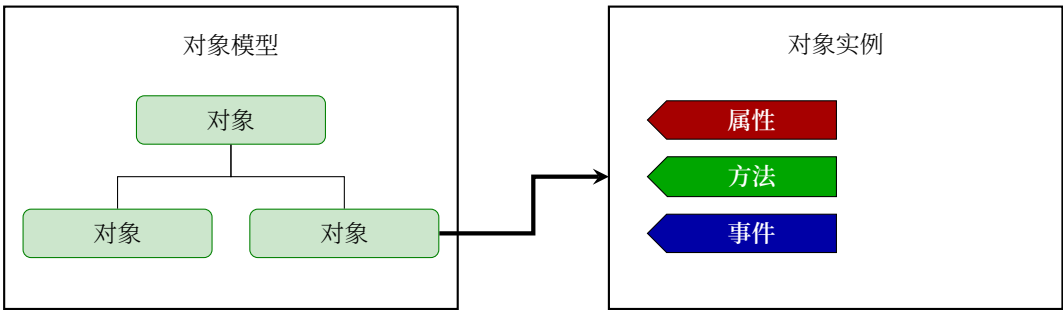


图 3.2 一个简单的对象模型

3.3 对象模型

对象模型是一种任意模式或类关系的排列，它定义了从更高级别的类派生一个对象的层次结构和方法。对象模型独立于用于访问它并在其逻辑框架内工作的语言或工具。无论使用的是 Visual Basic、VBA、Visual LISP、Delphi、Java、C/C++、VB、C# 还是包含或提供 ActiveX 接口的任何其他语言，都存在相同的模型。这并不意味着所有语言都同等支持对象模型的所有功能，一般说来不是的，有些功能只能在某些语言中实现，或者比在其他语言中更容易实现。

作一个类比可能是这样的：对象模型是一所房子及其房间、门窗的布置。进入和使用房子的人都与同一所房子打交道。在这种情况下，房子和房间是对象模型，人是编程语言。希望这样说你能明白一些。

3.4 类继承

对象模型总是以根对象或基础对象开始。对于 AutoCAD，基础对象是 AutoCAD 应用程序对象，也称为 `AcadApplication` 对象。这提供了派生所有其他对象和集合的基本属性、方法、事件和集合。例如，`AcadApplication` 对象有一组 `Documents` 集合，后者又包含一个或多个 `Document` 对象。每个 `Document` 对象都有自己的对象、集合、属性和方法等。

可以在对象模型中向下导航到子级对象和集合，也可以向上导航到父对象和集合。对象模型非常强大，能够使应用程序直接访问和操纵环境，从而来执行几乎无限的任务集。它还使一切保持整洁有序，这在开发软件解决方案时总是有帮助的。

3.5 集合与词典

集合是一组具有共同父容器的相似对象。这个容器有一个唯一的名字，在大多数情况下会提供它自己的方法来访问它包含的对象。词典是一种特殊类型的集合，允许使用名称扩展自己的集合。Visual LISP 不提供创建或处理集合的方法。它允许迭代遍历、修改、添加和删除成员。词典集合允许添加自己的词典对象以及填充、迭代遍历、添加、修改和删除它们的成员以及添加、修改和删除词典本身。

AutoCAD 中的一些常见集合是文档 (`Documents`)、图层 (`Layers`)、标注样式 (`DimStyles`)、线型 (`Linetypes`)、块 (`Blocks`) 等。

AutoCAD 中的另一些常见词典是 `PageSetups`、`Layouts` (是的，它们也存储为词典)、`LayerStates` 和 `XRecord` 对象。AutoCAD 词典函数是 `(dictadd)`、`(dictnext)`、`(dictremove)`、`(dictrename)`、`(dictsearch)` 和 `(namedobjdict)`。在 Visual LISP 中，可以访问 ActiveX 词典对象方法、属性和事件反应器，如下所示：

- 方法：
 - `AddObject`
 - `AddXRecord`
 - `Delete`
 - `GetExtensionDictionary`
 - `GetName`
 - `GetObject`
 - `GetXData`
 - `Item`
 - `Remove`
 - `Rename`

- Replace
- SetXData
- 属性:
 - Application
 - Count
 - Document
 - Handle
 - HasExtensionDictionary
 - Name
 - ObjectID
 - ObjectName
 - OwnerID
- 事件:
 - Modified

3.6 属性、方法和事件

• 属性是与对象或集合相关联的描述性属性。举例子的话就可能包括 `Name`、`Height`、`Width`、`Rotation`、`Scale`、`Color`、`Layer` 和 `Linetype` 等。属性将根据它们关联的对象类型而有所不同，但有些属性是所有对象和集合共有的。集合和词典通常提供 `Count` 和 `Name` 属性，以及 `Item` 和 `Add` 方法。只有词典会提供 `Delete` 方法，因为无法从 Visual LISP 中删除集合。

• 方法是对象提供的内置函数，用于访问或修改特殊属性或对对象本身执行特殊操作。常用方法包括 `Rotate`、`Erase`、`Copy`、`Scale` 和 `Offset`。你可能会注意到这些看起来就像 AutoCAD 的修改命令。好吧，本质上它们就是这样，但略有不同。

尽管 AutoCAD 的修改命令本质上是通用的，但它们必须验证每次执行对象的使用情况，方法由其宿主对象提供，因此每个对象仅提供受支持的方法。有些困惑？

换个说法，我们可以在任何时候使用 `OFFSET` 命令，但是如果尝试偏移一个单行文字 (`Text`) 对象，会从 AutoCAD 收到一条错误消息。文字 (`Text`) 对象本身提供了复制 (`Copy`)、旋转 (`Rotate`)、缩放 (`Scale`) 和移动 (`Move`) 等多种方法，但不提供 `Offset`。因此，可以从对象 `invoke` 方法，这能确保它对该对象的使用是有效的。

• 事件是对象或集合可以从各种活动中生成的动作，这些动作也可以被检测和响应。当事件与对这些事件的反应结合使用时，这称为事件驱动编程。AutoCAD 提供了一组功能强大的事件响应工具，称为反应器，能够在绘图环境中发布响应各种事件的监听设备。

例如，可以创建一个反应器，当活动绘图对象被删除时响应 `Erase` 事件。这只是事件和反应器的一个例子。

属性关联

重要的是要知道：永远不应该假设所有属性都可用于所有对象或集合。在运行中处理属性和方法时，有两个函数对于确保你的代码能够正确地执行是非常有价值的，这其中的一个就是 `(vlax-property-available-p)`。不幸的是，没有直接的方法来获取所有属性的列表，并依此以某种方式迭代给定对象进行编程。不过可以获取一个列表以供参考，这还是有很大帮助。

要查询给定对象具有哪些属性和方法，可以在该对象上使用 `(vlax-dump-object)` 函数。此函数的语法是 `(vlax-dump-object <object> [<show-methods>])`，其中 `[<show-methods>]` 参数为 `nil` 或非 `nil`。如果非 `nil`，则显示该对象支持的方法；否则不会显示方法。

```
Command: (setq acadapp (vlax-get-acad-object))
#<VLA-OBJECT IAcadApplication 00a8a730>
```

```
Command: (vlax-dump-object (vla-get-documents acadapp) T)

; IAcadDocuments: The collection of all AutoCAD drawings open in the
    ↪ current session
; Property values:
; Application (RO) = #<VLA-OBJECT IAcadApplication 00a8a730>
; Count (RO) = 1
; Methods supported:
; Add (1)
; Close ()
; Item (1)
; Open (2)
```

上面的输出显示了 `Documents` 集合对象的属性和方法。注意到输出的第一行显示了内部对象引用 (`IAcadDocuments`) 及其代表内容的描述，然后它列出了可用的属性和方法。



以下命令定义可能会派上用场，可供探索所选实体的属性和方法。虽然没有提供错误处理，但它仍然是一个有用的小工具。

```
(defun C:DUMP ( / ent obj)
  (while (setq ent (entsel "\nSelect entity to get object data: "))
    (setq obj (vlax-ename->vla-object (car ent))))
```

```
(vlax-dump-object obj T)
(vlax-release-object obj))
(princ))
```



某些属性旁边的括号括起来的 (RO) 表示只读，在这种情况下所有属性都是只读的。方法旁边的括号括起来的数字表示使用的方法需要参数的数量。

要访问属性，可以使用 `(vla-get-xxx)` 函数，或者更通用的 `(vlax-get-property)` 函数，二者都可以。第一种形式的语法是 `(vla-get-xxx object)`，其中 `xxx` 是属性名称。函数 `(vlax-get-property)` 的语法为 `(vlax-get-property object propertyname)`，其中 `propertyname` 可以是双引号字符串或单引号引用名称。

`(vlax-get-property object propertyname)` 或

`(vla-get-propertyname object)` 或

`(vlax-get object propertyname)`

返回分配给对象的指定名称属性的值。如果该对象不存在指定的属性，则会生成错误。例如，如果从直线 (Line) 查询 Diameter 属性，则会产生错误。

参数

`object` vla-object 对象；
`propertyname` 与对象相关的有效属性。

示例

```
(vlax-get-property objLine "Length")
(vlax-get-property objLine 'Length)
(vla-get-Length objLine)
(vlax-get objLine "Length")
(vlax-get objLine 'Length)
```

所有这些语句都将完成同样的事情。

属性名称不区分大小写，但为了清晰起见，本书中的示例通常将第一个字母大写。你会发现上面的前两个选项通常最容易使用，但是，有些情况需要使用后两个选项。尤其是在与 Microsoft® Excel 或 Microsoft® Word 等外部应用程序交互方面。

使用方法

使用第 22 页中的示例，可以看到 **Documents** 集合对象支持四种方法：**Add**、**Close**、**Item** 和 **Open**。**Item** 方法需要使用一个参数，因此在示例中该方法旁边显示 (1)，这是要从集合中获取的文档的索引或名称。

Item 方法的一个有趣特性是它可以接受字符串或整数值参数。当给定一个整数参数时，它只返回集合的第 n 项，其中 0 是第一项。当给定一个字符串值时，它会尝试通过其名称属性获取该项目。**Item(<name>)** 方法不区分大小写，这是相当有用的，它无需转换字符串大小写就可直接接受名称。



如果熟悉 Visual Basic 或 VBA 中默认方法和默认属性的使用，你应该注意这项功能在 Visual LISP 中是不存在的。例如，在 Visual Basic 中，可以使用以下三种方式之一访问 **Item** 方法：

Object.Item(12) 或 **Object(12)** 或 **Object("Name")**

这是因为 **Item** 方法是 VB 或 VBA 中大多数对象的默认方法。Visual LISP 不支持此功能，因此要求每次都拼写出所有属性和方法。不过不要难过，Microsoft® 已经放弃了对所有 .NET 语言中默认属性和方法的支持。以下 Visual LISP 示例应该更好地证明这一点：

```
(vlax-invoke-method documents "Item" 12) ;; will work
(vla-item documents "Drawing1.dwg") ;; will work
(vlax-invoke-method documents 12) ;; will not work
```

回头再看看第 22 页中的示例，**Item** 方法可以下列方式使用：

- **(vla-item documents 1)**
- **(vla-item documents "Drawing1.dwg")**
- **(vlax-invoke-method documents "Item" "Drawing1.dwg")**
- **(vlax-invoke-method documents 'Item "Drawing1.dwg")**

(vlax-invoke-method <object> <method> [<arguments>] ...) 或

(vla-<method> <object> <arguments>) 或

(vlax-invoke <object> <method> [<arguments>] ...)

调用与对象关联的方法并向该方法提供任何必需的参数。如果成功，则返回一个结果。如果对象未提供请求的方法，则会生成 ActiveX 错误。例如，从单行文字 (**Text**) 图元请求 **Offset** 方法将产生 ActiveX 错误。

参数

`<object>` vla-object 对象;
`<method>` 对象公开的方法;
`<arguments>` 方法所需的参数。

示例

```
(vlax-invoke-method objLine "Move" point1 point2)
(vla-Move objLine point1 point2)
(vlax-invoke objLine "Move" point1 point2)
```

所有这些例子做的都是同样的事情。这通常适用于大多数 AutoCAD 对象，但不适用于从导入的类型库接口、外部应用程序或 ActiveX 组件创建的对象。应使用第一种形式来处理外部应用程序对象，但是可以使用第二种形式来处理内部对象。



当选择在属性和方法上使用 Get/Put 中的任何一种时，会发现使用较长的形式（例如 `(vlax-put-property)`）比使用较短的形式（例如 `(vla-put-Color)`）更灵活。原因是通过将属性名称与函数分开，就可以自定义接受属性列表及其关联值的函数和迭代语句。例如……

```
(defun MapPropertyList (object proplist)
  (foreach propset proplist
    (if (vlax-property-available-p object (car propset))
      (vlax-put-property
        object
        (car propset)
        (cadr propset))))))
```

尝试将此方式应用于方法时要小心，因为方法的参数列表因对象和方法而异。有些方法不接受任何参数，而另一些方法的参数长度会有所不同。

3.7 数据类型

数据类型是一种描述给定对象或属性可以包含的值类型的方法。数据类型的例子包括整数型 (Integer)、双精度浮点数值型 (Double)、货币 (Currency)、日期 (Date)、字符串 (String) 等。虽然 AutoLISP 多年来一直享有类型无关性，Visual LISP 也基本如此，但并非总是这样。

在 AutoCAD 中，其当然可以像使用 AutoLISP 一样保持类型无关性，但在与其他应用程序（如 Microsoft® Excel）交互时，将不可避免地接受数据类型并明智地使用它们。

类型无关性也不是免费的午餐，其付出的代价是处理效率低下。当提前声明数据类型时，编译器就被告知只需分配足够的资源来满足预期的数据类型。例如，存储整数类型的数据比存储“长”日期值的要求要低得多。

在没有数据类型的工作时，所有内容都会自动分配为尽可能大的数据类型，以确保使用的任何内容都适合可用资源。结果是应用程序无论是在初始负载大小方面还是运行时资源分配方面比实际需要的更加臃肿。AutoLISP 从不强迫程序员在运行前声明数据类型。因此，作为最佳猜测的结果，它总是为给定符号使用最大分配。即使在使用 (getint) 时，变量本身也不会在运行时声明，因此 AutoLISP 在设计时无法有效地预测分配需求。

这就是为什么用 C++、Java 甚至 Visual Basic 等语言开发的应用程序通常更快的原因（与使用无类型语言的类似函数式编码相比）。他们提前确保更精简的执行，以确保在运行时更快的性能。AutoLISP 不这样做，因此是一种处理速度慢得多的语言。Visual LISP 要好得多，但前提是要尽可能充分利用新功能。不过我必须承认，即使是 Visual LISP 也不强制执行设计中的数据类型声明。只是在运行时，它比 AutoLISP 更严格地依赖和尊重数据类型。

对于任何熟悉 VB、VBA 和 VBScript 编程的人来说，Visual LISP 在严格性方面比起 AutoLISP 来更像 VBScript。例如，Visual LISP 中没有与 VB 或 VBA 中的 DIM 语句等效的语句。虽然 VBScript 不强制执行数据类型，但它至少支持 OPTION EXPLICIT 以强制声明变量。

常量与枚举

常量是一种特殊的数据类型。这就是它听起来的样子，一个无法更改的值。这有时被称为静态。通常，为了方便起见，常量由编程语言或托管应用程序本身提供。例如，常量 acByLayer 可以代替属性值 256，因为名称值通常比整数值更容易理解和记忆。例如，下面显示的两个表达式在功能上是相同的：

```
(vla-put-Color object acByLayer)
(vla-put-Color object 256)
```

枚举是常量的逻辑组，用于标识一系列常量值。例如，可以使用颜色 1、2、3、4 和 5，但表示这些颜色的常量（例如 acRed、acYellow、acGreen、acCyan、acBlue、acMagenta 和 acWhite）对于清晰和合理的编码来说非常方便。这种类型的相关常量值的范围称为枚举。有关标准 AutoCAD 枚举的列表，请参见附录 A^①。

^① 译者注：附录 A 给出的只是 VLAX 枚举，并不是完整的标准 AutoCAD 枚举列表。



并非所有 ActiveX 枚举都在 Visual LISP 中提供。例如，标准的 `Decimal` 和 `Short` 数据类型不会镜像为 `vlax-vbDecimal` 或 `vlax-vbShort`。有关数据类型的更多信息，请参阅第 6 章。

变体与安全数组

在上面关于数据类型的部分中，提到了可用于无类型声明的最大分配，例如 AutoLISP 中的 `(setq)` 表达式。实际上，这需要分配变体数据类型。变体是一种包罗万象的数据类型，它提供足够的资源空间来包含任何其他数据类型，无论是数字、日期、字符串还是其他任何类型。变体数据类型实际上是 ActiveX 的产物，但这个概念本质上更通用，并且在 ActiveX 出现之前很久就已经存在了。

Visual LISP 实际上将所有转换后的 ActiveX 数据保存为变体，并带有一个说明符，表示其中包含什么特定类型。这听起来令人困惑，但实际上非常简单。容器是一个变体，其中包含一个货币（`Currency`）数据类型值。为对象分配新值时，必须提供该说明符以确保正确存储数据。当在 AutoCAD 和其他应用程序（例如 Microsoft® Excel）之间传递值时尤其如此。

除了发送数值之外，还可以从变体值查询嵌套数据类型，以及将该值正确转换为相关的 LISP 数据类型。例如，查询其中包含 `Double` 值的变体对象值。然后，可以将该值作为 LISP 中的 `REAL` 数据类型读取。Visual LISP 提供了大量用于在 LISP 环境中创建、读取和修改变体数据值的函数。

安全数组类似于 AutoLISP 中的 `LIST` 对象。主要区别在于它们是静态的，这意味着它们不能被伸缩或改变它们可以存储的成员数量。这可以防止因尝试分配或获取超出数组长度的成员而产生不必要的错误。这就是为什么它们实际上被称为安全的原因。任何传递到 ActiveX 的 `LIST` 结构都必须先转换成安全数组。任何从 ActiveX 对象获取的面向 `LIST` 的数据都应转换为 `LIST` 数据类型，以供 LISP 函数使用，例如 `(car)`、`(nth)`、`(assoc)`、`(mapcar)` 和 `(member)` 等。Visual LISP 提供了大量用于创建、操作和读取安全数组数据值的函数。

有关变体和安全数组的更多信息，请参阅第 6 章。

3.8 名称空间

名称空间是资源分配的虚拟空间，进程在该空间中运行并与其他资源交互。但它有时可以与其他名称空间中的其他进程通信。将名称空间想象成卧室。某个应用程序可能是在一间卧室工作的人，即特定名称空间中的进程；另一个应用程序也可以在相邻的卧室（名称空间）中工作。两者可以保持独立和隔离，也可以使它们在彼此之间传递数据以进行通信。这基本上就

是名称空间的工作方式。

使用名称空间的一些优点是特定名称空间中的进程与其他名称空间中的进程隔离，这可以防止它们相互倾轧（试图在竞争中保留资源）。它还支持通过名称空间直接加载和卸载进程。换句话说，这有点像能够从房子中拔掉其中一间卧室的插头，就好像它是以模块化形式建造的一样。移除一间卧室不会影响其他房间或它们各自激活的进程。

使用名称空间的一个显着缺点可能是它们会在操作系统及其主机应用程序上产生一些开销。为了管理给定的名称空间，必须为其提供自己的内存地址范围和指针分配。这会消耗额外的资源来跟踪和控制名称空间，这反过来又在必要时提供了直接访问、卸载或暂停它的方法。

AutoCAD 在 Visual LISP 以及 ObjectARX 和 VBA 中提供自己的名称空间内部管理。这是 Visual LISP 对 AutoLISP 的又一次强大改进。实际上，每个打开的文档也是它自己的名称空间（如果不在单文档模式下工作）。在一个绘图会话中设置变量并尝试在另一个绘图会话中读取它时，可以看到这种效果。有一些方法可以在绘图会话之间传递此类变量，我们将在第 10 章中讨论这些方法。

3.9 接口与类型库

接口是连接到其他 ActiveX 进程或组件的对象模型的一种方式。当希望能够利用其他应用程序的特定属性、常量或方法时，首先必须定义一个接口来加载该目标应用程序的对象模型。例如，如果希望能够与 Microsoft® Excel 一起在 Visual LISP 中使用 Excel 自己的工具将一些 AutoCAD 信息直接存储到电子表格文件中，这就需要定义接口，然后允许使用类型库。

要使用类型库，必须将其加载到内存中并且必须定义某些接口指针。Visual LISP 提供了一组专门用于加载和配置类型库接口的函数。

(vlax-import-type-library)

将类型库引用导入当前名称空间。其语法格式为：

```
(vlax-import-type-library
  :tlb-filename <namestring>
  :methods-prefix <mpfxstring>
  :properties-prefix <ppfxstring>
  :constants-prefix <cpfxstring>)
```

参数

- <namestring> 类型库文件的路径和文件名；
- <mpfxstring> 任意前缀字符串标识符；

⟨ppfxstring⟩ 任意前缀字符串标识符；
 ⟨cpfxstring⟩ 任意前缀字符串标识符。

示例

```
(vlax-import-type-library
 :tlb-filename "c:\\myfiles\\typelibs\\tlfile.tlb"
 :methods-prefix "dsxm-"
 :properties-prefix "dsxp-"
 :constants-prefix "dsxc-")
```

此示例将类型库接口导入到文件 `tlfile.tlb` 中定义的外部应用程序或控件。其余参数定义从类型库接口公开的方法、属性和常量的前缀。

如果此类型库提供了一个名为 `AddNumbers` 的方法，它将在我们的 Visual LISP 代码中用作 `dsxm-AddNumbers`。有趣的是，一旦实际导入了类型库并且此表达式成功，Visual LISP 将识别来自外部应用程序所有已定义的属性、方法和常量，并像使用任何内置 LISP 函数一样用蓝色对它们进行颜色编码。这是 VLIDE 有助于编码和提高及早发现代码错误能力的另一个原因。

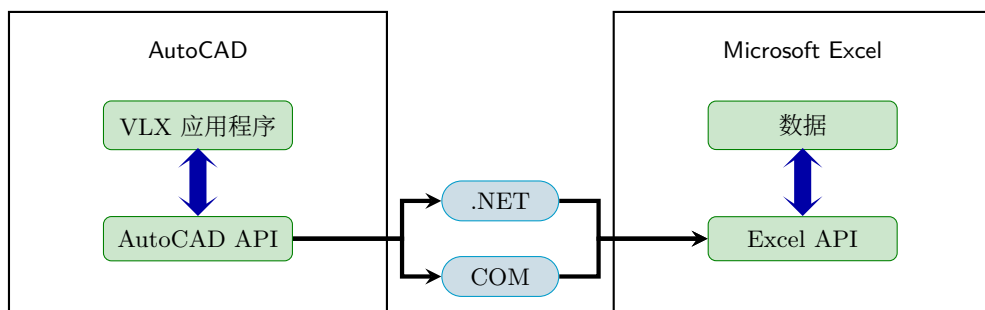


图 3.3 类型库接口

类型库只是一个接口，它将一个提供方的所有对象模型成员公开给请求它的其他应用程序。当加载一个类型库时，它会立即定义和识别提供方应用程序所有公开的属性、常量和工具，并给使用方应用程序使用。

在图 3.3 中，Excel 类型库被加载从而将 Visual LISP 与 Excel 的对象模型连接起来，并且可以使用它公开的工具。这可以通过直接访问 Excel 中内置的工具来节省大量时间和避免麻烦，这些工具将执行需要的操作，而无需尝试在 Visual LISP 中重新发明轮子。下面和例 3.1 显示了如何使用它的示例。

例如，当通过 Visual LISP 调用向 Excel 函数提供常量值作为参数时，可以使用常量枚

举名称而不是实际的值来保持代码清晰易懂。这也让你不必在 Excel 中查找所有枚举并将它们转换为 Visual LISP。如果 Excel 提供常量（例如 `put-cellcolor`），你可以直接从 Excel 中使用它。

Visual LISP 需要类型库信息来确定对象的方法、属性或常量是否可用。某些对象可能没有任何可用的类型库信息，例如 `AcadDocument` 对象。

`(vlax-typeinfo-available-p <object>)`

如果类型库信息可用于对象，则返回 `T`。否则返回 `nil`。

参数

`<object>` `vla-object` 对象。

示例

例 3.1 在 Excel 中使用支持类型库的代码

```
(defun Excel-Get-Cell (rng row column)
  (vlax-variant-value
    (msxl-get-item (msxl-get-cells rng)
      (vlax-make-variant row)
      (vlax-make-variant column))))

(defun Excel-Put-CellColor (row col intcol / rng)
  (setq rng (Excel-Get-Cell (msxl-get-ActiveSheet xlap) row col))
  (msxl-put-ColorIndex (msxl-get-interior rng) intcol))
```

例 3.1 中定义的第二个函数 `(Excel-Put-CellColor)` 提供了一种将颜色填充值应用到 Visual LISP 的 Excel 工作表中给定单元格的方法。这可以通过使用 Excel 中公开的，加载 Excel 类型库后提供的接口方法来实现。上面显示的类型库项目带有 `msxl-` 前缀。

一旦调用类型库接口，引用的函数就会被 Visual LISP 编辑器语法引擎识别。当正确输入它们时，它们会改变颜色以表明它们确实被外部类型库接口识别为有效函数。这是使它成为有用的编码实践的基础：语法识别。



类型库有多种形式，它们最常见的是 `.TLB` 文件，但也可以是 `.OLB`、`.DLL` 甚至 `.EXE` 文件。值得注意的是，Microsoft® Office 97 和 Microsoft® Office 2000 通常使用 `.TLB` 文件，而 Microsoft® Office XP 使用 `.EXE` 文件本身来为其他应用程序提供类型库接口定义。请查阅要使用的任何外部应用程序或服务的文档，以了解有关它如何公开其 ActiveX 类型库信息的信息。

Microsoft® Office 类型库版本

表 3.1:

产品	版本	示例
Office 97	8	Excel.Application.8
Office 2000	9	Outlook.Application.9
Office XP	10	Word.Application.10
Office 2003	11	Powerpoint.Application.11
Office 2007	12	Excel.Application.12
Office 2010	14	Word.Application.14

另一个方便的技巧是查看 `HKEY_CLASSES_ROOT` 下的 Windows® 注册表，找到每个服务和已安装应用程序的所有已注册 API 库。互联网上还有许多免费软件和共享软件工具可以查找、组织、查看和检查计算机上注册的所有类型库。这些通常比在注册表中查找更好，因为它们可以读取类型库中的属性页以显示内部文档。属性页是在 VBA 对象浏览器中选择对象时提取文档信息的地方。

如果省略版本标识符，Windows® 将通过读取 `HKEY_CLASSES_ROOT` 中类 GUID 条目下的 `CurVer` 子键来查询当前版本。例如，在某注册表中会同时看到 `Excel.Application` 和 `Excel.Application.10`。但是，如果在 `Excel.Application` 下查看，它将有一个 `CurVer` 子项，该子项很可能设置 `CurVer` 的值为 `Excel.Application.14`（假设计算机中安装了 Office 2010 或 Excel 2010），它还具有相同的 CLSID 值。这本质上就是更新升级时将类型库接口重定向到所有外部应用程序的方式：使用注册表。

在 Visual LISP 中调试代码

本章将重点介绍如何使用 Visual LISP 中的测试和调试工具及早发现问题并使代码更不易出错。调试是成功代码开发的一部分，就像修剪草坪是为了拥有自己的房子一样。它有时可能是乏味和痛苦的，但忽视或忽略它肯定会带来更大的痛苦。

越早熟悉并熟练使用调试工具，你在各方面就会过得越好。其中最重要的是提高代码质量和提高客户满意度（让支付你薪水的人脸上露出笑容永远不会有坏处）。

4.1 测试和调试工具

断点

断点是一种用于在代码中放置标记以在执行期间自动触发暂停的工具。如果在执行的特定部分遇到代码问题，请在该部分的开头放置一个断点并运行代码，直到它到达该断点。然后，可以使用以下一种或多种工具，有条不紊地深入执行，找出问题的原因并在更短的时间内解决问题。

示例

加载图 4.1 中所示的代码文件 `getprops.lsp`，并在包含一些圆（**Circle**）、圆弧（**Arc**）和直线（**Line**）等图元的图形中运行 `GETPROPS` 命令。

你会注意到，当你选择直线（**Line**）或椭圆（**Ellipse**）实体时，代码会崩溃并显示错误消息：

```
ERROR: ActiveX Server returned the error: unknown name: Diameter
```

如果你选择的是单行文字（**Text**）或点（**Point**）对象，也会发生同样的情况。也许你已经在示例代码中看到此错误的原因，但让我们假设这是一段复杂得多的代码，你无法通过查

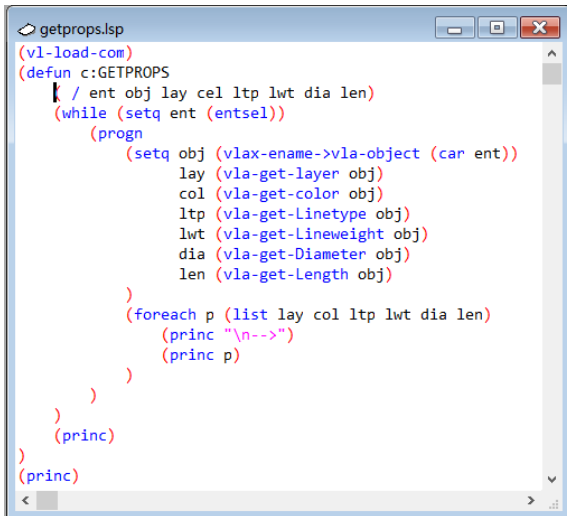


图 4.1 在 getprops.lsp 示例代码

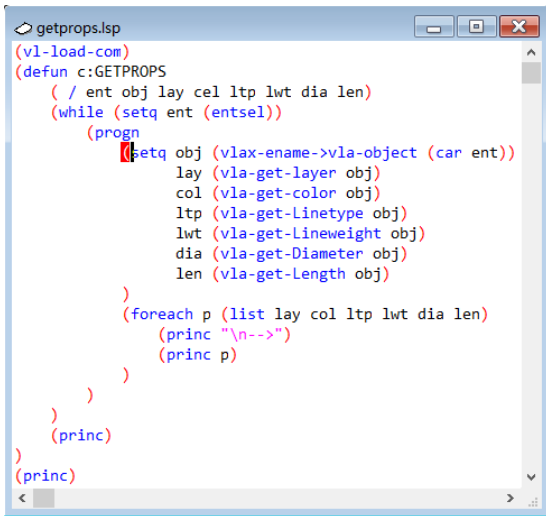




图 4.2 加断点的 getprops.lsp 示例代码

看代码轻松找到此错误的原因。现在要做什么？在代码中放置一个断点，加载它并再次运行它。这一次，当它到达代码中的断点位置时，执行暂停了，你可以使用 VLIDE 中提供的各种工具开始调试代码执行。这些工具之一称为分步执行。


将编辑器光标直接放在包含 `(setq obj ...)` 的行的前面，然后按 **[F9]** 键或选择  按钮，以在该位置打开断点。你会看到开头的括号 `(` 以红色框住（图 4.2）。这是 VLIDE 编辑器提供的视觉辅助工具之一，确实非常有用。开启断点后，使用  按钮再次将代码加载到 AutoCAD 中，按 **[Ctrl]+[Alt]+[E]** 也能达到同样的效果。

现在，运行 **GETPROPS** 命令并选择一个实体时，它将在该断点处停止并跳回 VLIDE 编辑器以等待下一个命令。你会注意到现在这里事情有点不同了。首先，包含在匹配括号内的代码块现在被高亮显示。其次，**调试** 工具栏按钮现在已启用（不再灰显，如图 4.3 所示）。该工具栏现在是继续调试过程的主要工具。



图 4.3 调试 工具栏




左边的前三个按钮是分步执行控制按钮（在下一节中有更详细的描述），后面是继续、退出和重置按钮。接下来的三个按钮是切换断点、添加监视和上一断点代码，最后是断点步进状态按钮。最后一个按钮仅显示一个视觉队列，说明当前进程是在匹配的括号子集之前还是之后停止。查看此内容有助于了解错误是在表达式之前抛出还是在求值之后抛出。

继续并选择 **下一嵌套表达式** 按钮 。持续点击该按钮，观察代码如何继续一次执行一

个表达式。这将一直持续到执行遇到产生错误的表达式为止。此时，执行中止并显示错误消息。

希望你能发现错误的原因是：这段代码假设某些属性可用，而没有首先验证它们确实可用。拾取直线（Line）图元时，Diameter 属性显然不可用。选择圆（Circle）图元时，Length 属性也不可。

分步执行

正如你在上面的示例中所猜测的那样，分步执行只是一种一次一行或一个表达式遍历代码执行的方法。这使你可以暂停执行并在程序进行中提前控制进度，直到你在代码中发现要检查的点或错误或特殊的条件。你可以按 下一嵌套表达式  按钮、下一表达式  按钮或 跳出  按钮，它们是所有编程语言中常用的分步执行方法，而不仅仅是在 Visual LISP 中。

- 下一嵌套表达式—在进行下一个表达式或语句前，按照先从下一表达式最深处的嵌套语句到最外面的语句的求值方式来继续前进执行。
- 下一表达式—跳过当前高亮的语句块并提早执行表达式或语句。
- 跳出—跳过断点块，前进到下一个表达式或语句。如果除了这个点外没有其它断点了，就继续不间断的执行。

自动执行

分步执行的另一种方法是使用自动执行。此功能正常执行代码，但在对每个表达式求值后会暂停以突出显示编辑器窗口中的代码块。每次暂停后，代码自动前进到下一个表达式。暂停是使用定时延迟值处理的，可以根据自己的需要或喜好进行调整。

监视


在调试环境中，监视是放置在特定对象或符号上的标记，用于在程序执行期间持续显示其属性。向特定变量（符号）添加监视使你能够在过程中遇到断点后的步进执行过程中查看其值分配。要添加观察，请通过在编辑器窗口中突出显示代码来选择一个符号，然后选择添加观察按钮  或按 **Ctrl**+**W**。这将打开监视窗口并将一个监视参数添加到监视列表中。你可以根据需要一次查看任意数量的符号，但请记住，添加的符号越多，就越难以清楚地看到问题的所在。

图 4.4 显示了在代码的 (foreach) 部分的符号 p 上放置了一个监视。通过移动断点到 (foreach) 部分的开始处，这样就可以在 (foreach) 循环过程中显示符号 p 每一次的值。

请注意在最初的时候，p=nil，因为这时代码并不在执行过程中，所以也没有值被分配给 p。当 (foreach) 循环开始时，p 就会分别显示列表 (lay col ltp lwt dia len) 中每个符

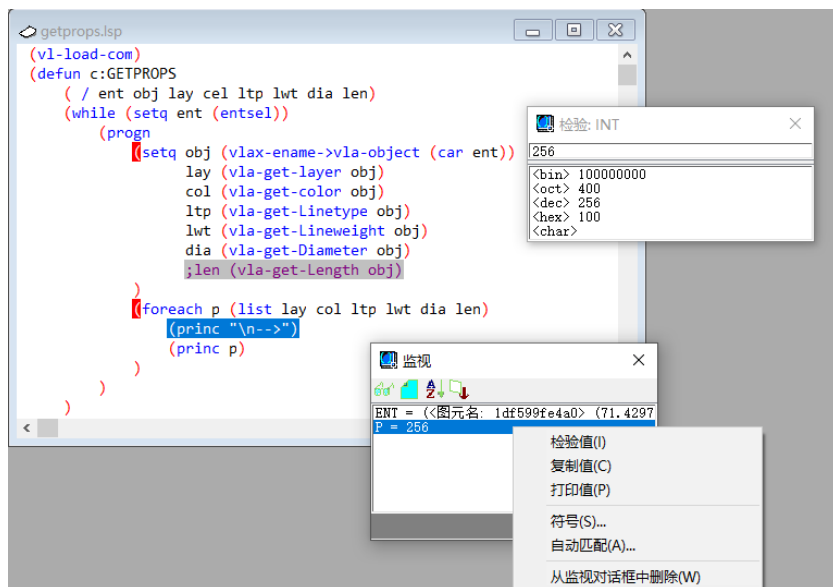


图 4.4 为符号 p 添加监视


号的值，即使它们被设置为 `nil` 也会被显示出来。

跟踪

Visual LISP 提供了多种跟踪功能。一个是命令跟踪，它在给定的命令（或所有命令）上放置了一个记号，当这个命令（或所有命令）被活动代码执行调用时，就会在跟踪日志窗口显示了一个通知。如果 `VLIDE` 处于打开状态，则会显示跟踪日志窗口，并且给出在执行期间发布任何调用。

如果 `VLIDE` 没有被激活，跟踪输出将被发送到 AutoCAD 命令提示窗口。不过，一旦 `VLIDE` 被激活了，就算是在你返回到 AutoCAD 编辑器进程时跟踪窗口仍然保持激活。因此，一旦 Visual LISP 被激活，所有的跟踪输出都被发送到 `VLIDE` 的跟踪窗口。在你关闭并重新打开 AutoCAD 去结束 Visual LISP 的跟踪输出之前，你必须返回到 `VLIDE` 进程去继续查看跟踪输出。

另一种类型的跟踪是堆栈跟踪。

跟踪按钮  不在 **调试** 工具栏上，而是在 **视图** 工具栏上。这是因为跟踪功能实际上是一个窗口显示，而与特定代码段相关的调试命令（如添加监视和断点相关功能）不同。

要显示跟踪窗口日志，你必须选择 **调试》跟踪命令** 并打开命令跟踪，同时检查“跟踪命令”的选中标识。完成后，任何从中的代码所调用的 AutoCAD 命令都被发送给了跟踪日志窗

口, 如图 4.5 所示。

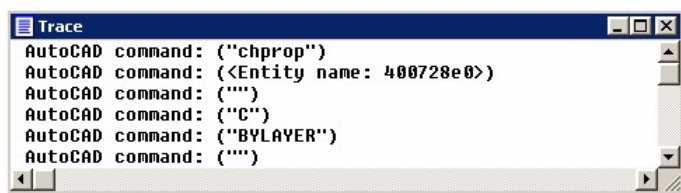


图 4.5 调用 **CHPROP** 命令后的跟踪日志窗口

图 4.5 显示了诸如 **CHPROP** 之类的命令如何连同它使用的任何参数（例如实体名称、命令行选项和提供给它的值）一起报告给“跟踪日志”窗口。你可能会注意到每个组件都表示为一个单成员列表。这是因为 Visual LISP 在内部以列表形式表示命令堆栈。

检验

检验涉及向下深入符号以查看它包含哪些属性以及它被定义成什么形式。例如, 检验函数 **(vla-get-ActiveSpace)** 将显示它被定义为一个 SUBR, 这是 Visual LISP 提供的一个内部函数。声明右侧的数字/字母字符串表示其在当前名称空间中的内存地址。

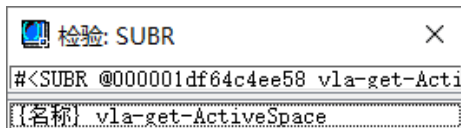



图 4.6 检查窗口

符号服务

符号服务实用程序提供了一种检查符号属性的方法。这包括保护状态、跟踪、入口状态调试以及是否已导出到 AutoCAD 名称空间。从这个弹出式表单中, 你还可以通过选择表单顶部的帮助按钮  来执行在线帮助查找。下面的示例显示了突出显示代码 **(vla-get-Activespace)** 并选择 **符号服务** 按钮的结果。你也可以右键单击突出显示的代码并从弹出菜单中选择 **符号服务**。

自动匹配

自动匹配功能允许你根据通配符匹配搜索函数、属性和方法, 并在 VLIDE 的列表框中返回它们的列表。从此列表中, 你可以复制/粘贴到你的代码窗口或执行在线帮助查找以了解该项目的功能或使用方式。有多种方法可以调用此功能。其中之一是右键单击某些代码并从弹出

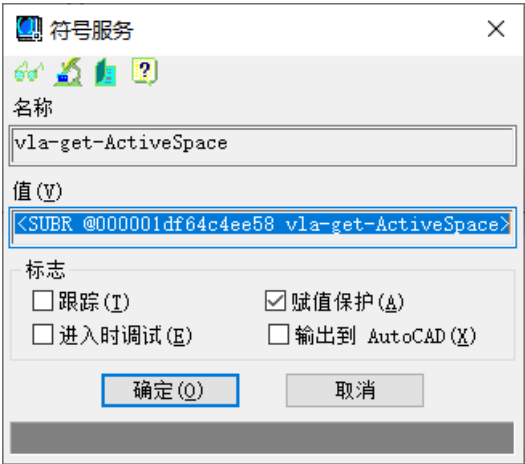



图 4.7 符号服务窗口

菜单中选择 **自动匹配窗口** (如图 4.8 所示)。或者你可以选择 **视图** 工具栏上的自动匹配按钮 .

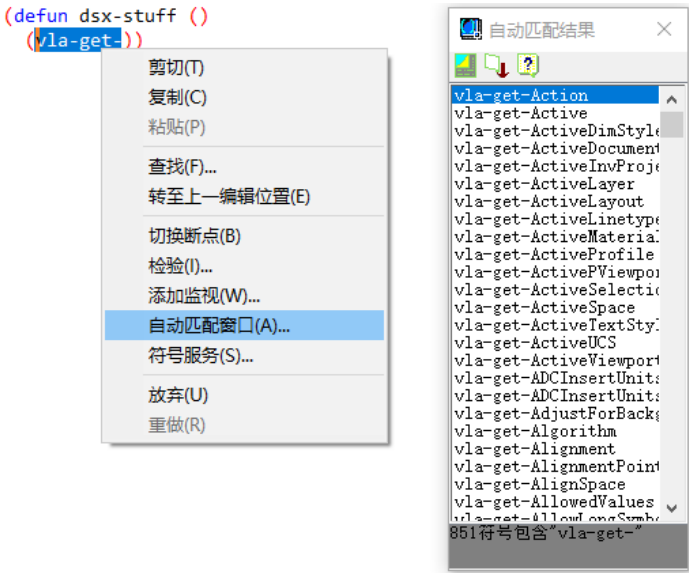


图 4.8 自动匹配结果窗口

如你在图 4.8 的例子中看到的，一个搜索 `vla-get-` 的自动匹配会在结果窗口中出现很多相匹配的项。你可通过在代码窗口中键入更多的字符来缩小搜索范围，例如 `vla-get-Active` 就只是找到那些以相同字符串值开头的项。

你也可以通过选择左上的按钮（工具提示显示 **自动匹配选项**），并在编辑框中键入你对搜索条件的更改来在结果窗口中修改自动匹配搜索。这个表单中的其它选项允许你指定大小写匹配、仅按前缀和小写转换。**过滤值** 按钮在过滤值表单上显示更多的选项（图 4.9）。

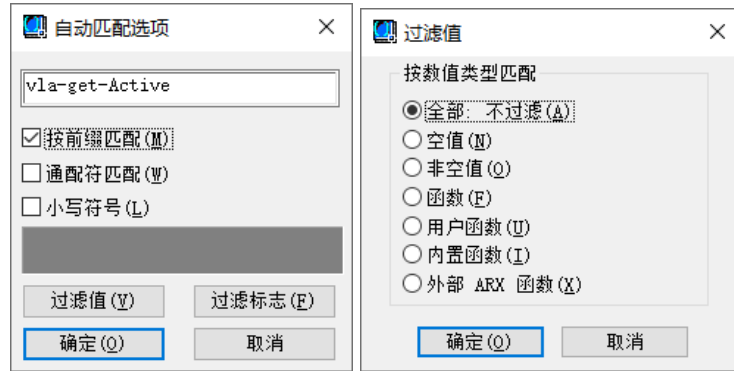


图 4.9 自动匹配过滤器值选项

例如，你可以将搜索限制在诸如内置函数、外部定义的函数（例如由 ObjectARX 应用程序定义的函数）和 Null 或 Non-Null 值等项目。**过滤标志** 按钮显示一个搜索过滤表单（图 4.10），用于将搜索限制为本身具有某些特征的符号，例如受保护的符号或已导出到 AutoCAD 名称空间的符号。

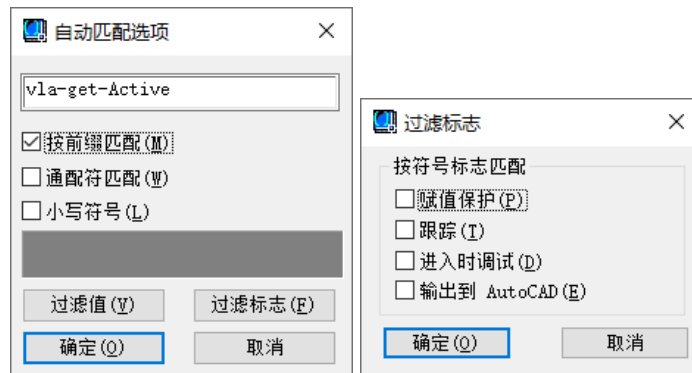


图 4.10 自动匹配过滤标志选项

书签

书签不一定是调试工具，但它们对于快速定位特定代码段很有用。当你在单个文件中处理大量代码并且很难在文件中跳转到代码中的特定点时，尤其如此。书签在插入它们的行的前面

显示为圆角方形实心绿色符号。

要插入书签，请将光标放在所需的代码行上，然后按 **Alt** + **.**（句点）或选择 **搜索 > 书签 > 切换书签**。要删除书签，请将光标放在书签所在的行上，然后再次按 **Alt** + **.** 或选择 **搜索 > 书签 > 切换书签**。要清除给定文件中的所有书签，请选择 **搜索 > 书签 > 清除所有书签**。



虽然 Visual LISP 不允许按名称跳转到书签，但可以以下一个/上一个方式在它们之间移动。要从一个书签跳到下一个书签，请按 **Ctrl** + **.**。要移动到上一个书签，请按 **Ctrl** + **,** 或继续按 **CTRL** + **.** 直到循环浏览所有书签再次回到那里。

转至行

当书签不实用时，你还可以通过行号直接跳转到代码中的一行。只需按 **Ctrl** + **G** 即可显示 **转至行** 对话框（图 4.11），输入行号并按 **Enter** 转到该行。



图 4.11 **转至行** 对话框

错误捕获

最终，没有适当的错误捕获，任何调试方法都无法使你到达目标。什么是错误捕获？它只是捕获错误以诊断错误的性质并因此执行一些纠正措施的过程。与简单地让错误导致代码崩溃并显示丑陋、神秘的消息让用户感到困惑相比，这更有效并产生更好的质量结果。

尤其是 **ActiveX**，在涉及其错误消息的内容时并不以非常友好而著称。例如，在 Visual LISP 中 **ActiveX** 操作抛出的常见错误信息如下：

```
Error: ActiveX error: No description provided.
```

这对用户意味着什么？就此而言，这对任何人意味着什么？不多。但是，在你的代码上下文中，你可能会尝试使用 **ADO** 或 **Jet** 启动与 **Access** 数据库的连接。在你尝试建立连接的地方，你应该在该代码周围放置一个错误捕获并测试它是成功还是失败，如果失败，则确定失败

的原因。然后你可以检查错误情况并显示一条有意义的消息，这可能会帮助用户自己找出原因，从而节省你更多的工作。

如何在代码周围放置错误捕获？你可以使用 Visual LISP 提供的函数来捕获、检查和处理 ActiveX 对象生成的错误。

4.2 Visual LISP 错误捕获函数

Visual LISP 在古老的 AutoLISP `(*error*)` 函数之上提供了一些额外的错误捕获和错误处理函数。这些函数中的每一个都为你提供了一组工具，用于捕获、验证和处理在 Visual LISP 中执行代码时抛出的错误，特别是对于在其自己的名称空间中运行的代码或与外部应用程序对象或过程交互的代码。举个例子来说，除非你使用这些特殊函数，否则有时很难拦截由于 ADO 故障等产生的错误并做出反应。

`(vl-catch-all-apply <function> <list>)`

在函数执行的结果上放置一个错误捕获。与 C++、C#.NET 和 VB.NET 编程语言中提供的 `try-catch` 异常处理类似。此函数返回成功对象或 `Error` 对象。`(vl-catch-all-error-p)` 函数确定返回对象是否为 `Error` 对象。

参数

`<function>` `(defun)` 或 `(lambda)` 形式的函数定义或符号指针；

`<list>` 用于求值的函数所必需的参数表。

函数 `(vl-catch-all-apply)` 用于在一组代码表达式周围放置错误捕获。执行后，任何结果都会直接传递到此函数的输出，可以在其中检查它是否生成了错误，如果是，生成了何种错误。

此函数的语法是 `(vl-catch-all-apply <function> <list>)`，其中 `<function>` 是正在执行的表达式，而 `<list>` 是函数正在执行或通过项。



请注意你打算使用或与之交互的每个 ActiveX 对象。应该小心确定对象在失败时是否会抛出 ActiveX 或 OLE 错误。如果它能够作为失败的结果抛出这样的错误（而不是返回 `nil`），你应该始终将用于与其交互的表达式封装在错误处理程序中，以防止代码在用户身上“爆炸”。

例如，要围绕打开 Microsoft® Excel 的尝试去设置错误捕获，可以使用类似这样的东西……

```
(cond
  ( (vl-catch-all-error-p
```

```
(setq XL
  (vl-catch-all-apply 'vlax-create-object
    '("Excel.Application.14"))))
(vl-exit-with-error
  (strcat "\nError: " (vl-catch-all-error-message XL)))
(T (princ "\nSuccessfully opened Microsoft Excel session object.")))
```

这个小小的例子执行了以下操作（按照处理顺序从内到外）：

- 尝试创建 `Excel.Application.14` 的对象；
- 如果尝试失败，它会将应用程序会话作为 `Error` 对象返回。
- `(vl-catch-all-error-p)` 检查应用程序会话时返回 `T`；
- 求值被 `(vl-exit-with-error)` 函数中止，该函数显示了通过 `Error` 对象应用程序会话传递的错误消息。
- 此错误导致了代码立即中止执行，并同时向用户显示相关消息。否则，如果函数 `(vl-catch-all-error-p)` 返回 `nil`，则返回的应用程序会话对象不是 `Error` 对象，程序可以继续用它做更多的事情。

一个更简单直接的测试是强制创建 "Divide by Zero" 错误并查看 Visual LISP 如何处理它。在 LISP 控制台窗口中，按所示顺序输入以下两行代码。在第一行之后，应该会看到返回的 `Error` 对象 `<%catch-all-applyerror%>`。在第二行之后，应该看到从 `Error` 对象返回的字符串值消息为 "Divide by Zero"。

```
Command: (setq catchit (vl-catch-all-apply '/ '(50 0)))
#<%catch-all-apply-error%>
```

```
Command: (vl-catch-all-error-message catchit)
"divide by zero"
```

使用 `(vl-catch-all-apply)` 的绝佳之处是尝试使用 `(vla-item)` 方法从集合中获取对象时。例如，我们希望以下代码片段在未找到匹配对象时返回 `nil`。但是，它实际会引发 `ActiveX` 错误。

```
(setq layers
  (vla-get-layers
    (vla-get-activedocument
      (vlax-get-acad-object))))
(setq mylayer (vla-item layers "Doors"))
```

执行此操作的正确方法是使用 `(vl-catch-all-apply)` 在请求失败时捕获错误。类似于：

```
(if
  (not
    (vl-catch-all-error-p
      (setq mylayer
        (vl-catch-all-apply
          'vla-item (list layers "Doors")))))
  (princ "\nLayer was found in layers collection!")
  (princ "\nLayer does not exist."))
```



这是我将在这本书中用来代替 `(vla-item)` 的示例函数。函数返回一个对象，如果在提供的集合中找不到项，则返回 `nil`。我强烈建议使用这样的函数代替 `(vla-item)` 以避免代码中的错误。

```
(defun get-item (collection item / result)
  (if
    (not
      (vl-catch-all-error-p
        (setq result
          (vl-catch-all-apply
            'vla-item (list collection item)))))
    result))
```

`(vl-catch-all-error-p <object>)`

视 `<object>` 是否为一个 `Error` 对象而返回 `T` 或 `nil`。

参数

`<object>` 任意 `vla-object` 对象。

示例

```
(vl-catch-all-error-p (vl-catch-all-apply '/ '(50 0)))
```

将返回 `T`，因为表达式 `(/ 50 0)` 是典型的 "divide by zero" 错误。

`(vl-catch-all-error-message <object>)`

从 `Error` 对象返回消息描述。如果 `<object>` 不是 `Error` 对象，则此函数返回 `nil`。

参数

`<object>` 任意 vla-object 对象。

示例

```
(vl-catch-all-error-message (vl-catch-all-apply '/ '(50 0)))
```

将显示一条 "divide by zero" 的错误信息。

```
(vl-exit-with-error <message>)
```

终止 ❷ VLX 的执行并返回字符串型的消息。

参数

`<message>` 含有错误信息结果的字符串。

该函数立即中止执行并返回一个字符串值作为结果。这可以用来向用户提供更清晰的自定义错误消息，非常有用。它与 AutoLISP `(exit)` 函数的工作方式非常相似，且还可以为出现错误的结果返回一个值。例 4.1 显示了如何将 `(vl-catch-all-error-message)` 作为返回消息值传递。

```
(vl-exit-with-value <value>)
```

终止 ❷ VLX 的执行并返回数字或符号结果值。

参数

`<value>` 任意值或符号。

示例

例 4.1 错误测试函数

```
(defun fubar (somevalue / *error*)
  (defun *error* (s)
    (vl-exit-with-value s))
  (/ somevalue 0)); force divide by zero error

(defun errortest ( / try)
  (cond
    ( (vl-catch-all-error-p
      (setq try (vl-catch-all-apply 'fubar (list 12))))
```



```
(princ  
  (strcat "\nError: "  
    (vl-catch-all-error-message try))))))
```

如果加载上面的示例并输入 `(errortest)`，结果将是 "Error: divide by zero"。函数 `(vl-exit-with-value)` 的工作方式与函数 `(vl-exit-with-error)` 相同，只是它返回一个数值作为结果。如果你想使用数值参数处理错误，例如传递 ActiveX 错误编号的返回值，这会很有帮助。

从这些函数中可以看出，可以使用 Visual LISP 执行非常详细的错误捕获和处理，以帮助生成质量更好的代码和软件产品。无论如何，这种做法并不是 Visual LISP 独有的。它与使用其他语言（如 C/C++、Visual Basic、VBA、Java 等等）所做的一般相同。然而，Visual LISP 提供的错误捕获和处理功能绝不像 .NET 和 C++ 提供的那样灵活和健壮（例如 `try/catch/fail` 或 `finally`）。错误捕获是有道理的，但必须努力有效地使用它才能获得它提供的好处。

使用工程和多文件












工程是为共同目的关联在一起的相关  LSP 文件的集合。举例说可以是包含希望在某些方面始终协同工作的单个功能或一组功能的多个文件。这在其他产品中也被称为工作空间，但总体意图是相同的：将相关的代码文件收集在一起并命名，以便于打开和共同处理它们。



图 5.1 工程下拉菜单和工程窗口

虽然工程很棒，但 Visual LISP[®] 还是具有一些局限性，这让它与市场上的其他代码开发工具（如 Microsoft[®] Visual Studio）相比不够理想。这些限制包括不能包含  DCL 或其他类型的代码文件，并且只能将工程编译为  FAS 输出，而不能编译为  VLX。理想情况下，Visual LISP 工程应该允许可以包含  VLX 应用程序中的所有文件类型（ DVB、 txt、 LSP、 DCL）。

尽管工程无非就是将相关的  LSP 文件放在一起，并能够在编辑器中快速打开其中的任何一个或所有文件，它还是非常有用的。有关如何管理工程配置的示例，请参见图 5.2 和 5.3。

一旦创建并打开一个工程，它将在显示所有成员  LSP 文件的 VLIDE 窗口中显示一个可停靠的列表框。要打开特定文件，只需双击它。

可以通过右键单击列表并从弹出菜单中选择 **多个选择** 来一次打开多个文件(请参阅图 5.1b)。

可以随时使用弹出菜单中的 **添加文件** 或 **删除文件** 选项在工程中添加文件和从工程中删除文件。还可以从 **工程特性** 面板中添加或删除文件 (参见图 5.2)。



在工程文件列表中添加文件或对它们进行排序的顺序是在使用工程文件列表作为 **生成应用程序** 向导的输入时它们将被编译的顺序 (在第 13 章中讨论)。在将文件添加到给定工程后，可以返回并修改文件的顺序。

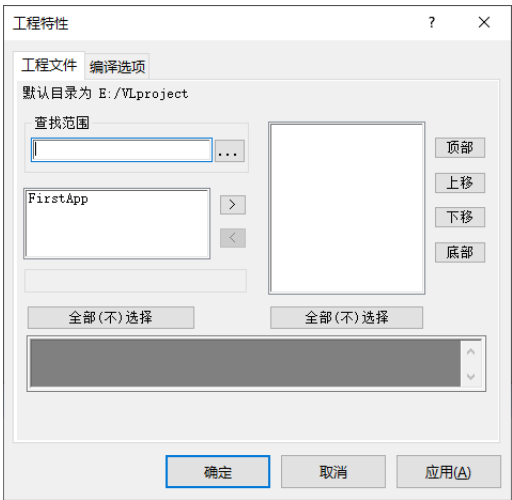


图 5.2 **工程特性** >> **工程文件**



图 5.3 **工程特性** >> **编译选项**








图 5.2 显示了用于制作和修改 Visual LISP 工程的主要属性形式。请注意，有两个选项卡 **工程文件** 和 **编译选项**。图 5.3 显示了 **编译选项** 选项卡面板。**工程文件** 面板是选择要成为工程一部分的  LSP 文件之处。

图 5.3 中显示的选项将在第 13 章 (创建 Visual LISP 应用程序) 中进行更详细的解释。所有这些选项都与  FAS 输出文件的制作有关， FAS 文件是编译的 LISP 代码，可以从一个或多个  LSP 文件创建为单个  FAS 文件，该文件可以与其他  FAS 文件一起编译成  VLX 应用程序文件。

使用变体和安全数组

虽然本书前面（第 27 页）讨论了变体和安全数组数据类型的话题，但它们在 Visual LISP 的 ActiveX 世界中非常重要，足以需要利用一整章专门进行讨论。我们将从简要回顾它们是什么开始，然后开始探索如何使用 Visual LISP 函数来进行处理。

正如我们之前提到的，变体是一种数据类型，旨在成为任何其他类型数据的通用容器。它们消耗所有数据类型中最多的内存和处理资源，因为它们在资源需求方面是最大的。

C/C++、Visual Basic 和 Delphi 等语言都提供了声明语句，提前通知编译器每个变量将包含什么数据类型。这不仅保证了更精简的资源需求，而且还允许在编译期间进行错误检查，从而避免运行中出现的问题。

6.1 Visual LISP 变体函数

```
(vlax-make-variant [<value>] [<type>])
```

使用给定值或符号求值创建变体对象。

参数：

[<value>] 要分配给变体的值。如果省略值，则创建类型为 `vlax-vbEmpty` 的空变体。

[<type>] 变体的数据类型。如果省略类型，则 LISP 数据类型转换为最接近的 ActiveX 数据类型（详见表 6.1）。

示例

```
(vlax-make-variant)
```

或者

```
(vlax-make-variant nil)
```

创建一个未初始化的变体 (`vlax-vbEmpty`)。

```
(vlax-make-variant 10 :vlax-vbInteger)
```

创建一个值为 10 的整数型 (`vlax-vbInteger`) 的变体。

```
(vlax-make-variant "vlisp example")
```

创建一个值为 "vlisp example" 的字符串型 (`vlax-vbString`) 的变体。

```
(setq dblarray (vlax-make-safearray vlax-vbDouble '(0 . 3)))  
(vlax-make-variant dblarray :vlax-vbArray)
```

创建一个包含双精度浮点数值 (Double) 安全数组的变体。



Visual LISP 不支持 `Decimal` 和 `Short` 的 ActiveX 数据类型。但是，可以在从外部源读取值时使用 (`vlax-variant-type`) 指定它们的类型。要以这些类型将数据发送到外部源，必须使用 `vlax-vbDecimal` 和 `vlax-vbShort` 的数字表示，因为它们在 Visual LISP 中未作为枚举提供。例如，`Decimal` 数据类型是枚举值 14。

变体数据类型

如果不为变体指定数据类型怎么办？Visual LISP 将尝试使用默认映射将其转换为适当的变体数据类型。表 6.1 显示了从 LISP 到变体的数据类型的默认映射。

```
(vlax-variant-type <symbol>)
```

返回变体的数据类型。如果符号不是变体，则会生成错误。返回值是数据类型的枚举（有关数据类型枚举，请参阅附录 A）。

参数：

`<symbol>` 包含变体值的符号。

表 6.1 Visual LISP 中 LISP 数据类型和变体数据类型的默认映射关系

LISP 数据类型	变体默认分配的数据类型
nil	vlax-vbEmpty
:vlax-null	vlax-vbNull
INT (integer)	vlax-vbLong
REAL (float)	vlax-vbDouble
STR (string)	vlax-vbString
VLA-OBJECT	vlax-vbObject
:vlax-true 或 :vlax-false	vlax-vbBoolean
VARIANT	与初始值类型相同
SafeArray	vlax-vbArray
N/A	vlax-vbShort
N/A	vlax-vbDecimal
N/A	vlax-vbDate

示例

```
(setq vartest (vlax-make-Variant 6 vlax-vbInteger))
```

```
Command: (vlax-variant-type vartest)
2
```

```
(setq vartest (vlax-make-Variant "dog" vlax-vbString))
```

```
Command: (vlax-variant-type vartest)
8
```

```
(setq vartest 123)
```

```
Command: (vlax-variant-type vartest)
*** ERROR: bad argument type: variantp 123
```

大于 8192 的变体类型值表示该变体包含某种形式的安全数组。要确定安全数组的数据类型, 请从返回值中减去 8192。例如, 如果返回值为 8197, 则安全数组数据类型为 5 (8197–8192), 这是一个 `vla-vbDouble` 的数据类型。

(vla-variant-value <symbol>)

返回包含变体的符号的值。如果符号不是变体, 则会生成错误。返回值是数据类型的枚举 (有关数据类型枚举, 请参阅附录 A)。

参数

<symbol> 包含变体值的符号。

示例

```
(setq vartest (vla-make-variant "testvalue" vla-vbString))
```

```
Command: (vla-variant-value vartest)
"testvalue"
```

```
(setq sa (vla-make-Safearray vla-vbDouble '(0 . 2)))
(setq vartest (vla-make-variant sa vla-vbDouble))
```

```
Command: (vla-variant-value vartest)
#<safearray...>
```

```
Command: (vla-safearray->list (vla-variant-value vartest))
(0.0 0.0 0.0)
```

(vla-variant-change-type <symbol> <type>)

更改变体分配的数据类型。

参数

<symbol> 包含变体值的符号;

<type> 要转换成的数据类型编号或枚举。

示例

```
(setq vartest (vlax-make-variant 5 vlax-vbInteger))
(setq vartest (vlax-variant-change-type vartest vlax-vbString))
```

将 `vartest` 转换为 `String(vlax-vbString)` 类型的变体, 这将导致从 `(vlax-variant-value)` 返回一个为 "5" 的值。

6.2 Visual LISP 安全数组函数

`(vlax-make-safearray <type> <dim1> [<dim2>] ...)`

创建维度边界 `<dim1> ...` 数据类型为 `<type>` 的安全数组, 其中可以指定其他维度。如果操作因任何原因失败, 则表达式返回 `nil`。

`<type>` 数据类型 (整数或枚举);
`<dim1>` 数组的维数 (一维数组);
`[<dim2>]` 数组的第二个维数 (二维数组) 等。

示例

```
(setq sa (vlax-make-Safearray vlax-vbDouble '(0 . 2)))
```

创建双精度的一维数组, 能够存储三个不同的元素 (0, 1, 2)。

```
(setq sa (vlax-make-Safearray vlax-vbString '(0 . 1) '(1 . 3)))
```

创建一个二维字符串数组, 第一个维度包含两个从索引 0 开始的元素。第二个维度包含三个元素并从索引 1 开始。



如果需要填充 安全数组, 那么可以考虑使用函数 `(vlax-safearray-fill)` 或者是使用函数 `(vlax-safearray-put-element)`, 这具体取决于需要一次分配一个元素还是一次分配所有元素。

`(vlax-safearray->list <symbol>)`

如果 `<symbol>` 包含安全数组, 则元素以 LISP 的 `LIST` 数据类型返回。如果 `<symbol>` 不包含安全数组, 则会生成错误。应该将对此函数的调用包装在错误捕获中, 以确保得到正确的错误处理。

参数

`<symbol>` 包含安全数组的符号。

`(vlax-safearray-type <symbol>)`

如果 `<symbol>` 包含安全数组，则元素的数据类型以枚举结果（整数值）返回。这可以通过整数或枚举结果进行匹配（有关数据类型枚举，请参阅附录 A）。如果 `<symbol>` 不包含安全数组，则会生成错误。

参数

`<symbol>` 包含安全数组的符号。

示例

```
(setq sa (vlax-make-Safearray vlax-vbDouble '(0 . 3)))
```

```
Command: (vlax-safearray-type sa)
5
```

返回值 5 (Double) 等价于 `vlax-vbDouble`。

`(vlax-safearray-fill <safearray> <element-values>)`

为安全数组中的多个元素赋值。如果提供的参数不是数组，则返回 ActiveX 错误。应该将此函数的调用包装在错误捕获中，以确保得到正确的错误处理。

`<safearray>` 安全数组型的对象；

`<element-values>` 要存储在安全数组中的数值表。可以指定与安全数组中的元素一样多的值。如果指定的值少于安全数组元素的数量，则其余元素将保留其当前值或保留为空。对于多维安全数组，元素值必须是包含表的表，每个表对应于安全数组的一个维度。

示例

创建双精度 (Double) 值的一维数组：

```
Command: (setq myarray (vlax-make-Safearray vlax-vbDouble '(0 . 2)))
#<safearray...>
```

使用 `(vlax-safearray-fill)` 函数填充数组元素：

```
Command: (vlax-safearray-fill myarray '(1 2 3))
#<safearray...>
```

列出数组的内容以验证元素值:

```
Command: (vlax-safearray->list myarray)
(1.0 2.0 3.0)
```

(vlax-safearray-get-element <safearray> <element> [<element>...])

返回安全数组中指定元素的值, 其中<element> 是整数, 表示要在数组中获取的索引位置。如果 <safearray> 参数不是安全数组对象, 则返回 ActiveX 错误。应该将对此函数的调用包装在错误捕获中, 以确保得到正确的错误处理。

参数

<safearray> 安全数组型的对象;
<element> 要获取的索引位置的整数。

示例

```
Command: (setq sa (vlax-make-Safearray vlax-vbString '(1 . 2) '(1 . 2) ))
#<safearray...>
```

使用 **(vlax-safearray-put-element)** 函数填充数组:

```
Command: (vlax-safearray-put-element sa 1 1 "A")
"a"
```

```
Command: (vlax-safearray-put-element sa 1 2 "B")
"b"
```

```
Command: (vlax-safearray-put-element sa 2 1 "C")
"c"
```

```
Command: (vlax-safearray-put-element sa 2 2 "D")
"d"
```

使用 `(vlax-safearray-get-element)` 函数检索数组第一维中的第二个元素：

```
Command: (vlax-safearray-get-element sa 1 1)
"A"
```

```
Command: (vlax-safearray-get-element a 2 2)
"D"
```

`(vlax-safearray-put-element <safearray> <elmnt> [<elmnt>...] <value>)`

为安全数组中的单个元素分配新值。如果 `<safearray>` 参数不是安全数组对象，则返回 ActiveX 错误。如果提供的元素值无法转换为预期的数组数据类型，也将返回 ActiveX 错误。应该将对此函数的调用包装在错误捕获中，以确保得到正确的错误处理。

参数

- `<safearray>` 安全数组型的对象。
- `<elmnt>` 一组指向要为其赋值的元素的索引值。对于一维数组，指定一个索引值；对于二维数组，指定两个索引值，依此类推。
- `<value>` 分配给每个元素的值。要为数组中的各个元素分配不同的值，请使用唯一值为对应位置的元素进行单独调用。

示例

```
Command: (setq sa (vlax-make-Safearray vlax-vbString '(1 . 2) '(1 . 2)))
#<safearray...>
```

使用 `(vlax-safearray-put-element)` 函数填充数组：

```
Command: (vlax-safearray-put-element sa 1 1 "A")
"A"
```

```
Command: (vlax-safearray-put-element sa 1 2 "B")
"B"
```

```
Command: (vlax-safearray-put-element sa 2 1 "C")
"C"
```

```
Command: (vlax-safearray-put-element sa 2 2 "D")
"D"
```

还可以使用 `(vlax-safearray-fill)` 函数填充数组值。以下函数调用同时完成了另外三个 `(vlax-safearray-put-element)` 调用的任务：

```
(vlax-safearray-fill sa '(("A" "B") ("C" "D")))
```

`(vlax-safearray-get-dim <safearray>)`

返回给定安全数组的维度（数组维度的数量）。如果提供的参数不是安全数组，则返回 ActiveX 错误。应该将对此函数的调用包装在错误捕获中，以确保得到正确的错误处理。

参数

`<safearray>` 安全数组型的对象。

示例

```
Command: (setq myarray (vlax-make-Safearray vlax-vbInteger '(2 . 5)))
#<safearray...>
```

```
Command: (vlax-safearray-get-dim myarray)
2
```

`(vlax-safearray-get-l-bound <safearray> <dim>)`

返回指定数组维度的下界（整数值）。如果提供的参数不是数组，则返回 ActiveX 错误。应该将对此函数的调用包装在错误捕获中，以确保正确的错误处理。

参数

`<safearray>` 安全数组型的对象；
`<dim>` 表示数组中维度位置的整数，其中第一个维度为 1。

示例

下例对按如下方式定义的安全数组进行相关求值：

```
(setq tmatrix (vlax-make-Safearray vlax-vbString '(1 . 2) '(0 . 1) ))
```

获取数组第一维的起始索引值：

```
Command: (vlax-safearray-get-l-bound tmatrix 1)
1
```

(vlax-safearray-get-u-bound <safearray> <dim>)

返回指定数组维度的上界（整数值）。如果提供的参数不是数组，则返回 ActiveX 错误。应该将此函数的调用包装在错误捕获中，以确保正确的错误处理。

参数

<safearray> 安全数组型的对象；
<dim> 表示数组中维度位置的整数，其中第一个维度为 1。

示例

```
(setq sa (vlax-make-Safearray vlax-vbString '(1 . 2) '(0 . 1) ))
```

```
Command: (vlax-safearray-get-u-bound sa 1)
2
```

数组第一维以索引 2 结束。获取数组第二维的结束索引值，从 1 开始：

```
Command: (vlax-safearray-get-u-bound sa 2)
1
```

可以定义自己的速记函数，就像 VB 和 VBA 中常用的 `Lbound()` 和 `Ubound()` 一样。

对象操作函数

Visual LISP 提供了一系列用于创建、操控和关闭 ActiveX 对象的函数。这通常是对外部应用程序会话对象而言的，但它同时也可以适用于任何外部进程对象，如 DLL 或 OCX 接口。

`(vlax-get-object <program-id>)`

尝试连接到既有的对象（进程），与 VBA 函数 `GetObject(<program-id>)` 相同。

参数

`<program-id>` 命名应用程序对象类标识符的字符串，如 "Word.Application.14" 或 "Excel.Application.14"。

示例

```
(setq xlap (vlax-get-object "Excel.Application.14"))
```

成功时返回既有外部 Excel 应用程序进程的 vla-object 对象，否则返回 nil。

`(vlax-create-object <program-id>)`

尝试创建一个新的对象（进程），与 VBA 函数 `CreateObject(<program-id>)` 相同。

参数

`<program-id>` 命名应用程序对象类标识符的字符串，如 "Word.Application.14" 或 "Excel.Application.14"。

示例

```
(setq xlapp (vlax-create-object "Excel.Application.14"))
```

成功时返回一个新的外部 Excel 应用程序进程的 vla-object 对象，否则返回 nil。

```
(vlax-get-or-create-object <program-id>)
```

首先尝试连接到既有的对象（进程），如果没有找到，则尝试创建一个新的对象（进程），在 VBA 中没有与之对应函数，这是 Visual LISP 独有的。

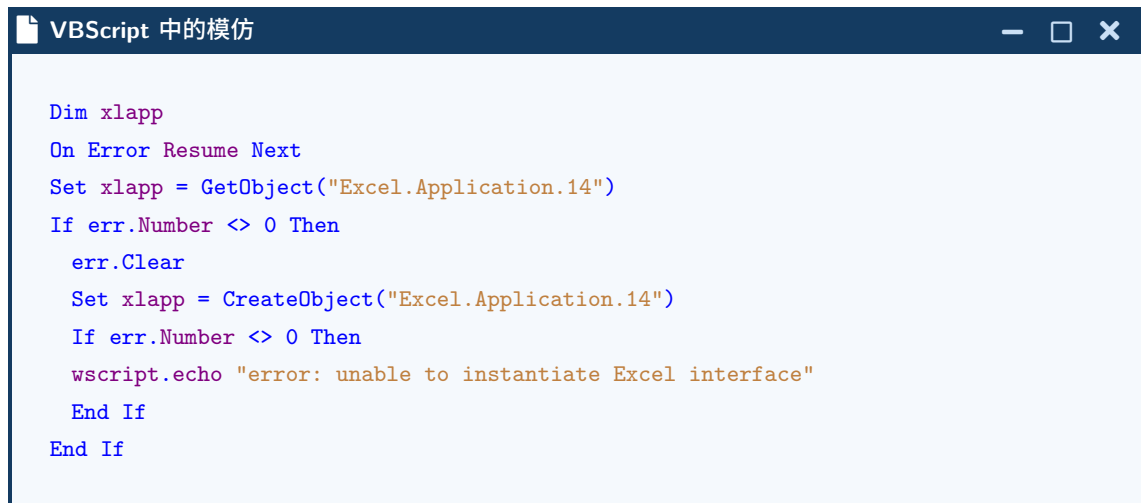
参数

<program-id> 命名应用程序对象类标识符的字符串，如 "Word.Application.11" 或 "Excel.Application.11"。

示例

```
(setq xlapp (vlax-get-or-create-object "Excel.Application.11"))
```

成功时返回一个外部 Excel 应用程序进程的 vla-object 对象，否则返回 nil。要在 VBScript 中模仿此功能，它可能类似于以下内容：



```

Dim xlapp
On Error Resume Next
Set xlapp = GetObject("Excel.Application.14")
If err.Number <> 0 Then
    err.Clear
    Set xlapp = CreateObject("Excel.Application.14")
    If err.Number <> 0 Then
        wscript.echo "error: unable to instantiate Excel interface"
    End If
End If

```

```
(vlax-write-enabled-p <object>)
```

如果 <object> 能够被修改则返回 T，否则返回 nil。



使用这个功能时要小心，当对象实际上已打开以供修改时，它可能经常返回 `nil`。

参数

`<object>` 任意 `vla-object` 对象。

`(vlax-object-erased-p <object>)`

如果对象已从绘图中删除，则返回 `T`，否则返回 `nil`。

参数

`<object>` 任意表示实体对象类型的 `vla-object` 对象。

`(vlax-release-object <object>)`

从内存中释放对象，但不释放内存。释放指向外部应用程序会话的对象时，强烈建议强制调用 `(gc)` 从操作系统资源中释放外部进程。

参数

`<object>` 任意 `vla-object` 对象。



尽管对象符号是局部的，函数用完后不一定要释放对象的资源。但还是建议当已经不需要这个对象时可使用该函数以确保释放对象。然而，需要提醒的是，即使是释放了由外部应用程序驱动的对象，它也可能没有从内存或操作系统进程堆栈中被全部释放。最好是在你完成代码后释放所有不再使用的对象，然后再调用 `(gc)` 函数去强制回收内存堆。

文件与目录函数

Visual LISP 提供的一些最有用的函数是文件和目录函数。这些是使你能够访问和修改文件属性以及列出指定文件夹中的文件和文件夹的功能集合。使用这些的一个例子是在对话框列表框的上下文中。

也许你想在列表框中显示绘图文件列表但不显示它们的扩展名（可能是为了缩短名称）。这可以通过将目录列表和 `(vl-filename-base)` 函数结合在一起来完成，如下所示：

```
(mapcar 'vl-filename-base (vl-directory-files pathname "*.dwg"))
```

这将返回一个形如 ("drawing1" "drawing2" ...) 的表。请谨慎使用此示例，因为它不提供错误检查。如果 `(vl-directory-files)` 函数返回 `nil`，则表达式的其余部分将因错误而崩溃。给出这个示例仅是为了演示如何组合这些功能并用于促进文件和目录信息的使用。

`(vl-file-size <filename>)`

以整数形式返回文件名的字节大小。如果未找到文件名，则返回 `nil`。

参数

`<filename>` 被查询文件的文件名字符串。

示例

```
(vl-file-size "c:\\myfile1.txt"); returns 125523 (roughly 124 Kb)
```

```
(vl-file-copy <source-filename> <target-filename> [<append>])
```

将文件从源位置复制到目标文件。如果 `<append>` 非 `nil` 且目标文件存在，则将源文件附加到既有目标文件；如果目标文件存在且 `<append>` 为 `nil`，则不会复制文件，返回值为 `nil`。如果复制成功，则返回一个整数值。

参数

`<source-filename>`

被复制的源文件的文件名字符串，如果该文件不在默认搜索路径内，文件名必须包含全路径。

`<target-filename>`

复制到的目标文件的文件名字符串。如果未指定目标路径，则使用默认工作目录位置。

`[<append>]`

如果非 `nil`，当目标文件存在时则将源文件内容添加至目标文件。

示例

```
(vl-file-copy "c:\\myfile1.txt" "c:\\mycopy.txt")
(vl-file-copy "c:\\myfile2.txt" "c:\\mycopy.txt" T); appends target file
```

```
(vl-file-delete <filename>)
```

删除文件，成功则返回 `T`，否则返回 `nil`

参数

`<filename>`

被删除文件的文件名字符串。

```
(vl-file-rename <old-name> <new-name>)
```

将文件名由 `<old-name>` 重命名为 `<new-name>`，成功则返回 `T`，否则返回 `nil`。

参数

`<old-name>`

既有文件的文件名字符串。

`<new-name>`

完成重命名后的文件名字符串。

```
(vl-file-directory-p <filename>)
```

如果 `<filename>` 是一个路径文件夹的名称则返回 `T`，如果 `<filename>` 是一个文件名或者干脆不存在则返回 `nil`。

`(vl-file-systime <filename>)`

返回文件名最后修改的日期和时间值列表。返回列表的形式为 (⟨年⟩ ⟨月⟩ ⟨星期⟩ ⟨日⟩ ⟨时⟩ ⟨分⟩ ⟨秒⟩)。

`(vl-filename-base <filename>)`

返回文件的基本文件名，不带路径和扩展名。

参数

⟨filename⟩ 文件的文件名字符串，带或不带路径及扩展名均可。

示例

```
(vl-filename-base "c:\\myfiles\\drawing1.dwg")
```

返回 "drawing1"。

```
(vl-filename-base "drawing1.dwg")
```

返回 "drawing1"。

`(vl-filename-directory <filename>)`

从给定的文件名字符串中返回目录或路径前缀值。

参数

⟨filename⟩ 命名文件的字符串，包括路径名。

示例

```
(vl-filename-directory "c:\\dwgfiles\\working\\drawing1.dwg")
```

返回 "c:\\dwgfiles\\working"。

`(vl-filename-extension <filename>)`

从给定的文件名字符串中返回扩展名。

参数

⟨filename⟩ 命名文件的字符串。

示例

```
(vl-filename-extension "c:\\myfiles\\drawing1.dwg")
```

返回 "dwg"。

```
(vl-filename-mktemp [<pattern>] [<directory>] [<extension>])
```

创建用于临时文件的唯一文件名。返回文件名字符串，其格式为：

```
<directory>\<base><xxx>.<extension>
```

其中 <base> 最多 5 个字符，取自 [<pattern>]，<xxx> 是 3 个字符的唯一组合。

当你退出 Visual LISP 时，在 Visual LISP 会话期间由 (vl-filename-mktemp) 生成的所有文件名都将被删除。

参数

[<pattern>] 包含文件名模式的字符串；如果为 nil 或不存在，vl-filename-mktemp 将使用 "\$VL~~"。

[<directory>]

命名临时文件目录的字符串；如果为 nil 或不存在，vl-filename-mktemp 将按以下顺序选择一个目录：

- [<pattern>] 中指定的目录（如果有）；
- TMP 环境变量中指定的目录；
- TEMP 环境变量中指定的目录；
- 当前目录。

[<extension>]

分配给文件的扩展名字符串，如果为 nil 或不存在，vl-filename-mktemp 将使用 [<pattern>] 的扩展名部分（可能是空字符串）。

示例

```
Command: (vl-filename-mktemp)
```

```
"C:\\TMP\\$VL~~004"
```

```
Command: (vl-filename-mktemp "myapp.del")
```

```
"C:\\TMP\\MYAPP005.DEL"
```

```
Command: (vl-filename-mktemp "c:\\acad2010\\myapp.del")
"C:\\ACAD2010\\MYAPP006.DEL"
```

```
Command: (vl-filename-mktemp "c:\\acad2010\\myapp.del")
"C:\\ACAD2010\\MYAPP007.DEL"
```

```
Command: (vl-filename-mktemp "myapp" "c:\\acad2010")
"C:\\ACAD2010\\MYAPP008"
```

```
Command: (vl-filename-mktemp "myapp" "c:\\acad2010" ".del")
"C:\\ACAD2010\\MYAPP00A.DEL"
```

(vl-directory-files <path> <pattern> [<mode>])

根据选择的模式返回一个目录下文件名和（或）子目录名。

参数

- | | |
|------------------------|---|
| <path> | 查询目录的目录名字符串。 |
| <pattern> | 查询的文件的名字字符串，可能包含通配符。如果未指定或为 <code>nil</code> ，则假定为 <code>"*.*"</code> 。 |
| [<mode>] | 整型数，取值如下： <ul style="list-style-type: none"> • -1 — 仅列出目录名； • 0 — 列出目录和文件名，缺省时的默认设置； • 1 — 仅列出文件名。 |

示例

```
Command: (vl-directory-files "c:\\dwgfiles\\Working" "*.dwg")
("drawing1.dwg" "drawing2.dwg" ...)
```

```
Command: (vl-directory-files "c:\\dwgfiles" nil -1)
("." ".." "Finished" "Working")
```

```
Command: (vl-directory-files "c:\\dwgfiles" nil 1)
nil
```


迭代与字符串函数

AutoLISP 提供了许多强大的映射和迭代函数，例如 `(while)`、`(foreach)`、`(mapcar)` 和 `(apply)` 等等。Visual LISP 还添加了一些更适合处理 ActiveX 集合对象的内容。这其中包括 `(vlax-for)`、`(vl-every)` 和 `(vlax-map-collection)` 等等。

```
(vlax-map-collection <object> <function>)
```

将函数应用于集合对象成员（对象）。如果 `<object>` 不是集合，则会生成错误。

参数

`<object>` 表示集合的 vla-object 对象
`<function>` 应用于对象的符号或 lambda 表达式

示例

```
(setq docs (vla-get-Documents (vlax-get-Acad-object)))  
(vlax-map-collection docs 'vlax-dump-object)
```

这将为当前打开的每个文档列出完整的属性列表……

```
; IAcadDocument: An AutoCAD drawing  
; Property values:  
; Active (RO) = -1  
; ActiveDimStyle = #<VLA-OBJECT IAcadDimStyle 046bb644>  
; ActiveLayer = #<VLA-OBJECT IAcadLayer 046bbd84>  
; ActiveLayout = #<VLA-OBJECT IAcadLayout 046b8a64>  
.....
```

```
(vlax-for <symbol> <collection> [<expression1> [<expression2>] ...])
```

迭代集合的成员对象并对每个成员对象执行表达式。如果第二个参数不是集合对象，则会生成错误。对符号的引用是局部的和临时的，就像 `(foreach)` 一样。

参数

<symbol> 分配给集合中每个 `vla-object` 对象的符号；
 <collection> 表示集合的 `vla-object` 对象；
 [expression(x)] 要求值的一个或多个表达式。

示例

```
(setq acadapp (vlax-get-Acad-object))
(setq layers (vla-get-Layers (vla-get-ActiveDocument acadapp)))
(vlax-for eachLayer layers
  (princ (vla-get-Name eachLayer))
  (terpri))
```

这将在命令提示符下列出活动图形中所有图层的名称。



避免定义名为 `acad` 的符号，因为它通常会产生运行时错误。

```
(vl-position <item> <list>)
```

如果能够找到 <item>，则返回 <item> 在 <list> 中的位置索引序号，否则返回 `nil`。第一项的位置索引为 0。

参数

<item> 任意值或符号；
 <list> 值或符号组成的表。

示例

```
(setq mylist '("A" "B" "C"))
(vl-position "B" mylist); returns 1
(vl-position "b" mylist); returns nil.
```

```
(vl-every <predicate-function> <list> [<list>] ...)
```

将每个提供的 `<list>` 的第一个元素作为参数传递给 `<predicate-function>`，然后是每个 `<list>` 的下一个元素，依此类推。一旦其中某一个 `<list>` 用完，求值就会停止。

参数

`<predicate-function>`

测试函数。这可以是任何函数，它接受与 `(vl-every)` 提供的 `<list>` 一样多的参数，并在任何用户指定的条件下返回 `T`。如果 `<predicate-function>` 为每个元素组合均返回一个非 `nil` 值，则函数返回 `T`，否则返回 `nil`。

`<predicate-function>` 的值可以采用下面的形式：

```
(function (lambda (A1 A2) ...))
```

`<list>` 被测试的表。

示例

检查给定文件夹中是否有大于 1024 字节的文件：

```
(vl-every
  (function
    (lambda (filename)
      (> (vl-file-size filename) 1024)))
  (vl-directory-files nil nil 1))
```

比较两个表……

```
(vl-every '= '(1 2) '(1 3)); Returns nil
(vl-every '= '(1 2) '(1 2 3)); Returns T
```

第一个表达式返回 `nil`，因为 `(vl-every)` 比较到了每个表中的第二个元素，但它们在数值上并不相等。第二个表达式返回 `T`，因为 `(vl-every)` 在处理完较短的表 `(1 2)` 中所有元素后就停止了比较，此时表在数值上相等。如果到达表末尾，`(vl-every)` 返回一个非 `nil` 值。

```
Command: (setq list1 (list 1 2 3 4))

(1 2 3 4)
```

```
Command: (setq list2 nil)

nil
```

```
Command: (vl-every '= list2 list1)
T
```

返回值为 T 是因为 (vl-every) 对 nil 的响应就像它已经到达表的末尾一样（即使 (predicate-function) 尚未应用于任何元素）。由于已经到达表末尾，(vl-every) 返回一个非 nil 值。

```
(vl-remove <element> <list>)
```

得到将 <list> 中的每一个 <element> 项移除后的一个新表。

参数

<element> 任意符号或值；
<list> 值或符号组成的表。

示例

```
Command: (vl-remove 3 (list 1 23 45 62 3 2 3 4 2 3))
(1 23 45 62 2 4 2)
```

```
(vl-string-search <pattern> <string> [<start-position>])
```

在字符串中搜索匹配模式并返回其所在起始位置的整数值。

参数

<pattern> 用于在 <string> 参数中搜索的字符串模式；
<string> 要以 <pattern> 查找模式进行搜索的字符串；
<start-position> <string> 中开始搜索的位置，默认是字符串的开头。

示例^①

```
(setq acadapp (vlax-get-Acad-object))
(setq layers (vla-get-Layers (vla-get-ActiveDocument acadapp)))
(vlax-for eachLayer layers
  (princ (vla-get-Name eachLayer))
  (terpri))
```

① 译者注：这个示例似乎有些问题，并不是对 (vl-string-search) 的应用，但译者没有找到正确的示例。

`(vl-string->list <string>)`

将字符串解析或标记为字符串中每个字符的 ASCII 码的表。

参数

`<string>` 要解析或标记的字符串。

示例

```
Command: (vl-string->list "ABCDEF")
(65 66 67 68 69 70)
```

`(vl-string-elt <string> <position>)`

返回字符串值中特定位置字符的 ASCII 码。

参数





`<string>` 被查找的字符串。

`<position>` 表示要求值字符位置的整数值。


示例

```
Command: (vl-string-elt "ABCDEF" 3)
68
```

`(vl-get-resources <filename>)`

返回打包在  VLX 工程中的  TXT 文件中的文本内容。它旨在从  VLX 上下文中调用，这就是不命名  VLX 本身，而是命名其中的文本文件的原因。

参数

`<filename>` 嵌入的  TXT 文件的名称，不带扩展名。

示例

```
(princ (vl-get-resource "license"))
```

`(vl-string-trim <chars> <string>)`

在删除起始（左侧）和结尾（右侧）字符后返回一个字符串。

参数

- ⟨chars⟩ 要移除的字符^①;
- ⟨string⟩ 要修剪的字符串。

示例

```
Command: (vl-string-trim " " " ABC ")  
"ABC"
```

```
Command: (vl-string-trim "A3" "ABACAB3")  
"BACAB"
```

(vl-string-left-trim ⟨chars⟩ ⟨string⟩)

仅删除起始（左侧）字符后返回字符串。

参数

- ⟨chars⟩ 要移除的字符;
- ⟨string⟩ 要修剪的字符串。

示例

```
Command: (vl-string-left-trim " " " ABC ")  
"ABC "
```

```
Command: (vl-string-left-trim "A3" "ABACAB3")  
"BACAB3"
```

(vl-string-right-trim ⟨chars⟩ ⟨string⟩)

仅删除结尾（右侧）字符后返回字符串。

参数

- ⟨chars⟩ 要移除的字符;
- ⟨string⟩ 要修剪的字符串。

^① 译者注：这里的字符是复数形式的“characters”，中文不容易明确表达。也就是说 ⟨chars⟩ 实际是一个字符串，其中的任意一个字符都会作为被移除的字符在 ⟨string⟩ 的起始和结尾进行搜索删除。其余两个类似函数的参数意义相同，不再赘述。

示例

```
Command: (vl-string-right-trim " " " ABC ")
" ABC"
```

```
Command: (vl-string-right-trim "A3" "ABACAB3")
"ABACAB"
```

(vl-string-mismatch <str1> <str2> [<pos1> <pos2> <ignore-case>])

从指定位置开始，返回两个指定字符串之间最长公共前缀的长度。

参数

- <str1> 要比较的第一个字符串；
- <str2> 要比较的第二个字符串；
- [<pos1>] <str1> 中开始比较的位置，默认为 <str1> 的开头；
- [<pos2>] <str2> 中开始比较的位置，默认为 <str2> 的开头；
- [<ignore-case>]
是否忽略大小写。默认情况下会考虑大小写。

示例

```
Command: (vl-string-mismatch "The dog" "The cat" 1 1 T)
3
```

```
Command: (vl-string-mismatch "He said that was good" "She said that was bad" 4 5)
13
```

(vl-string-position <char-code> <str> <start-pos> [<from-end>])

使用指定字符 ASCII 码返回匹配字符串的索引位置。

参数

- <char-code> ASCII 码数字；
- <str> 要搜索的字符串；
- <start-pos> <str> 中开始搜索的位置，默认是 <str> 的开头；
- [<from-end>] 为 T 时表示从字符串末尾开始反向搜索，为 nil（默认情况）时则是从头开始。

示例

```
Command: (vl-position (ascii "A") "HEY MAN")
```

```
5
```

```
Command: (vl-position 65 "HEY MAN")
```

```
5
```

```
Command: (vl-string-position (ascii "Y") "HEY MAN" nil)
```

```
2
```

```
(vl-string-subst <new-pattern> <old-pattern> <string> [<start>])
```

将字符串中的一种模式替换为另一种模式。很像 AutoLISP 函数 [\(subst\)](#)。

参数

<new-pattern>	替换 <old-pattern> 字符串值的新字符串值；
<old-pattern>	<string> 中要替换的现有字符串；
<string>	要实施替换操作的字符串；
[<start>]	在 <string> 中搜索 <old-pattern> 的起始索引。

示例

```
Command: (vl-string-subst "Help" "Love" "Visual LISP gets no Love")
```

```
"Visual LISP gets no Help"
```

```
(vl-string-translate <source-set> <destination-set> <string>)
```

参数

<source-set>	要匹配的字符模式；
<destination-set>	用于替换 <source-set> 中字符的字符模式；
<string>	要操作的字符串。

示例

```
Command: (vl-string-translate "5" "6" "123456 123456")  
"123466 123466"
```

(ACET-STR-TO-LIST *<delimiter>* *<string>*)

将字符串以定界符为分隔解析为表。虽然这在技术上不是 Visual LISP 函数，但它仍然很有用，并且由于通常会安装 Express Tools，所以它通常可以使用。

参数

- <delimiter>* 用于定界的字符或字符串；
- <string>* 要解析成表的字符串；

示例

```
Command: (acet-str-to-list " " "the dog ate")  
("the" "dog" "ate")
```


使用名称空间

开发 Visual LISP 的 VLX 应用程序可以使用单独的名称空间，以提供更多的性能控制和更好的安全性。但是，必须更改编码来避免问题并提供正确的结果，这会增加一些成本。这包括导入和导出函数和符号，以及将值传入和传出本地名称空间。

因为可以隔离单独的名称空间 的 VLX 应用程序，所以如果需要也可以查询和卸载它，这与加载到文档名称空间中并且无法识别或无法按名称卸载的普通 LISP 函数不同。这与 ObjectARX 应用程序非常相似，它为开发人员提供了在 Visual LISP 出现之前 LISP 无法使用的附加功能。

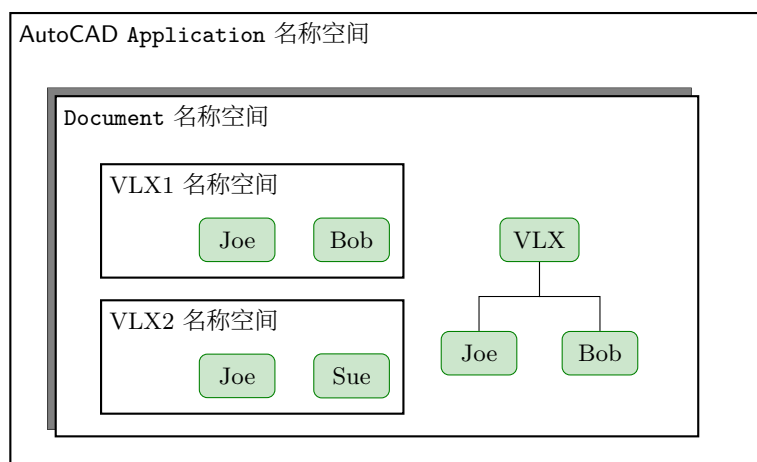


图 10.1 名称空间关系

在 ActiveX 或 COM（组件对象模型）开发的世界中，每个应用程序通常都在 Windows® 中自己的名称空间中运行。这是启用多任务处理的常见部分。由给定应用程序启动的其他进程可能会或可能不会在应用程序的名称空间内运行。它们实际上可能在自己独立的名称空间中运行。优点很多，但也有折衷。

参考图 10.1, 我们可以使用一些人物示例来描述名称空间如何工作以及其中的进程如何运行。在 `Document` 名称空间内运行的两个 `VLX` 应用程序各自在各自独立的名称空间中运行。这有点误导, 因为它们实际上并不是在 `Document` 名称空间内运行, 而是在 `AcadApplication` 名称空间内运行。但是, 因为它们被加载到 `Document` 名称空间中, 所以它们仅在该 `Document` 名称空间内被引用。第三个 `VLX` 不是单独的名称空间应用程序, 而是像任何传统的 AutoLISP 应用程序一样完全在 `Document` 名称空间内运行。

10.1 名称空间范围

请注意, 在此示例中, 存在三个名为 `Bob` 的对象 (函数定义)。虽然每个都被加载到同一个 `Document` 名称空间中, 但它们无法看到或彼此影响。这导致具有三个不同的 `Bob` 对象, 有点像 `Bob Smith`、`Bob Jones` 和 `Bob Doe`。除非合并某些特定的 Visual LISP 函数, 否则它们根本无法相互通信或相互影响。因此, 在本文档中引用 `Bob` 对象的任何对象都只能访问位于同一名称空间中的对象。


如果使用的话, 全局符号也是如此。如果我们在文档会话中从命令行将符号 `G$BOB` 的值设置为 "A", 则在 `VLX1` 或 `VLX2` 中运行的任何 `G$BOB` 符号都不会受到影响。在 `VLX1` 中的函数, 我们可以将 `G$BOB` 分配给值 "B"。如果 `VLX1` 中的函数显示 `(princ G$BOB)`, 它将返回 "B", 但从命令提示符请求 `(princ G$BOB)` 仍将返回 "A"。

这种类型的保护导致通常称为私有函数或私有符号, 因为它们对于该 `VLX` 名称空间是私有的。然而, 在 `Document` 名称空间中定义的函数和符号不是私有的, 因为在该名称空间中运行的所有其他应用程序都可以访问它们。更准确地说, 私有和公共是相对于调用进程所在的位置 (在各自的名称空间内部或外部) 而言的。换句话说, 对象 `Sue` 对在 `VLX2` 名称空间中定义和运行的任何函数都是公共的, 但 `Sue` 在默认情况下 `VLX1` 和其他名称空间无法访问它, 这个意义上它被认为是私有的。


10.2 名称空间函数

当将 `LSP` 代码编译到单独的名称空间 `VLX` 模块时, 需要使用一些特殊函数来让代码与在新 `VLX` 模块的名称空间之外运行的其他 `VLX` 模块进行通信。无论其他 `VLX` 模块是在文档名称空间内还是分别编译到它们自己的名称空间中, 都是如此。但是, 值得注意的是, 对于编译到其自己的独立名称空间中的每个 `VLX`, 如果需要它们相互通信或与文档会话通信名称空间, 需要在每个文件 (不仅仅是其中某些文件) 中依赖这些函数。


(vl-list-loaded-vlx)

返回所有加载的独立名称空间  VLX 应用程序的列表。如果没有加载，则返回 `nil`。


(vl-unload-vlx <appname>)

按名称卸载独立名称空间  VLX 应用程序 (<appname> 是一个字符串值)。这与 AutoLISP 的 `(arxunload)` 函数对 ObjectARX 应用程序的作用类似。


参数

<appname> 表示要卸载的  VLX 应用程序名称的符号或字符串，例如 `"myapp.vlx"`。






(vl-vlx-loaded-p <appname>)

如果在当前绘图会话中加载了指定的独立名称空间  VLX 应用程序，则返回 `T`。否则，它返回 `nil`。

参数

<appname> 表示要查询的  VLX 应用程序名称的符号或字符串，例如 `"myapp.vlx"`。






(vl-doc-export <function>)

从独立名称空间  VLX 应用程序中公开一个函数，供其名称空间之外的应用程序或函数使用。这必须在编译到独立名称空间  VLX 应用程序之前，在给定  LSP 文件的顶部任何函数定义之前声明。未从给定  VLX 导出的函数是该  VLX 私有的，不能从其名称空间外部访问。

参数

<function> 表示函数名称的引用符号。

(vl-doc-import <filename> [<function>])

从另一个  VLX 应用程序导入函数以在当前独立名称空间  VLX 应用程序中使用。如果不导入从其他  VLX 应用程序公开的此类函数，则在充当该函数的使用方的  VLX 应用程序中无法访问它们。如果指定了 <filename>，但省略了 [<function>]，则会导入  VLX 模块 (<filename>) 中的所有函数。

参数

<filename> 表示  VLX 应用程序名称的符号或字符串
 [<function>] 表示函数名称的引用符号。

如果你想限制被导入的函数，你必须使用 `[(function)]` 参数来命名这些函数，一次一个。`(filename)` 参数不使用文件扩展名，仅使用外部 **VLX** 应用程序文件的基本文件名，并且该文件必须位于默认搜索路径中，或者必须指定完整路径和文件名。

```
(vl-arx-import [(function)| (appname)])
```

从指定加载的 **VLX** 文件中导入一个或一组函数。如果省略 `[(function)]` 和 `[(appname)]`，则导入当前文档名称空间中所有 **VLX** 定义的函数。该函数应该在 `(defun)` 函数定义中使用。必须将 **VLX** 应用程序加载到当前文档会话中才能使用此功能。

参数

- `[(function)]` 表示函数名称的引用符号；
- `[(appname)]` 表示导入的 **VLX** 文件的符号或字符串，当使用 `[(function)]` 时，必须提供此参数。

示例

如果想在独立名称空间 **VLX** 应用程序中使用 DOSLib 的 ARX 函数 `(dos_getstring)`，则必须按如下方式导入：

```
(vl-arx-import 'dos_getstring "doslib2k.arx")
```

如果想从 **doslib2k.arx** 中导入所有函数，可以简单地去掉函数名，如下所示：

```
(vl-arx-import "doslib2k.arx")
```

```
(vl-doc-set (symbol) (value))
```

在名称空间 **VLX** 应用程序中的文档名称空间中设置一个符号。如果在独立名称空间 **VLX** 应用程序之外使用，它的行为类似于 `(set)` 函数。此函数可用于将在独立名称空间 **VLX** 应用程序中定义的符号复制到文档名称空间以供公共访问。该符号是按值复制的，而不是按引用复制的，这意味着 **VLX** 应用程序中的符号不能从文档名称空间修改。要导入文档名称空间符号，必须在独立名称空间 **VLX** 应用程序中使用 `(vl-doc-ref)` 函数。

参数

- `(symbol)` 引用的符号名称；
- `(value)` 分配给符号的值。

示例

将下面定义的示例函数编译到单独的名称空间 **VLX** 中并加载到 AutoCAD 中：




```
(defun DOCSET ()
  (vl-doc-set 'G$NAME1 "Joe"))
```

在文档名称空间，通过命令提示符：

```
Command: (DOCSET)
Command: !G$NAME1
```

```
"Joe"
```


(vl-doc-ref <symbol>)

将符号从文档名称空间导入到  VLX 应用程序的独立名称空间。该符号是按值复制的，而不是按引用复制的，这意味着不能从  VLX 名称空间中直接修改文档名称空间符号。要从  VLX 名称空间中导出或设置文档名称空间符号，必须使用 (vl-doc-set) 函数。

参数

<symbol> 引用的符号名称。

(vl-load-all <filename>)

同时将命名的  VLX 文件加载到所有打开的文档中。它还会加载到随后在同一 AutoCAD 应用程序会话中打开的任何文档中。

参数

<filename> 表示有效文件名的符号或字符串。

(vl-propagate <symbol>)

将符号及其关联值复制到 AutoCAD 应用程序名称空间中所有打开的文档，以及随后在同一 AutoCAD 会话期间打开的所有文档。

参数

<symbol> 引用的符号名称。

(vl-bb-set <symbol>)

将符号及其关联值发布到 blackboard 名称空间。blackboard 名称空间是 AcadApplication 名称空间的一部分，Documents 集合中所有打开的文档都可以访问它。这提供了与 Windows® 剪贴板类似的功能，只是它仅用于发布和检索 LISP 符号。

参数

`<symbol>` 引用的符号名称。

`(vl-bb-ref <symbol>)`

从 `blackboard` 名称空间中检索符号及其关联值。

参数

`<symbol>` 引用的符号名称。

`(vl-list-exported-functions)`

返回已从任何加载的 `VLX` 应用程序公开给文档名称空间的所有函数的列表。

`(vlax-add-cmd <globalname> <function> [<localname>|<flags>])`

将 `VLX` 应用程序中使用 `(defun)` 方式但未定义为 `c:` 形式的函数定义为命令行函数。必须指定 `<globalname>` 和 `<function>` 参数。[`<localname>`] 和 [`<flags>`] 参数则是可选的。不能使用 `(vlax-add-cmd)` 将函数公开为创建反应器对象或用作反应器回调的命令。成功返回 `<globalname>` 值，否则返回 `nil`。

建议在独立名称空间的 `VLX` 中使用 `(vlax-add-cmd)`，并使用 `APLOAD` 命令加载 `VLX`，而不是从 `LISP` 启动例程中加载。

参数

`<globalname>`

表示在命令提示符下使用的命令名称的字符串；

`<function>`

表示函数名称的引用符号。

[`<localname>`]

`VLX` 应用程序名称空间内部的命令名称。如果省略，默认为 `<globalname>`；

[`<flags>`] 修改命令关于透明度、选择集和 `pickfirst` 选项等等的标志。

主要标志选项：

- `ACRX_CMD_MODAL (0)` — 当另一个命令处于活动状态时无法调用此命令。
- `ACRX_CMD_TRANSPARENT (1)` — 可以在另一个命令处于活动状态时调用此命令。

次要标志选项：

- ACRX_CMD_TRANSPARENT (2) —检索 Pickfirst 集时, 它会在 AutoCAD 中被清除。命令将能够检索 Pickfirst 集。命令无法检索或设置夹点。
- ACRX_CMD_TRANSPARENT (4) —检索 Pickfirst 集或夹点集时, 它们都不会在 AutoCAD 中被清除。命令可以检索 Pickfirst 集和 Grip 集。

如果同时设置了 ACRX_CMD_USEPICKSET 和 ACRX_CMD_REDRAW 选项, 则其效果与仅设置 ACRX_CMD_REDRAW 相同。有关这些标志选项的更多信息, 请参阅 ObjectARX 参考手册中的命令堆栈主题。

示例

在 `Transparent.VLX` 中定义下列函数并加载到 AutoCAD 中:

```
(vl-load-com)
(vl-doc-export 'example1)
(defun example1 ()
  (princ "\nThis is an example transparent function.")
  (princ))
(vlax-add-cmd "example1" 'example1 "example1" ACRX_CMD_TRANSPARENT)
(princ)
```

```
Command: LINE
Specify first point: 'EXAMPLE1
This is an example transparent function.
Resuming LINE command.
Specify first point:
```

```
(vlax-remove-cmd <globalname>)
```

删除以前使用 `(vlax-add-cmd)` 定义的命令。函数本身并不受影响, 删除的是命令组中的命令提示符接口。

参数

`<globalname>` 表示要删除命令的字符串。

示例

```
Command: (vlax-remove-cmd "example1")  
T
```

```
Command: (vlax-remove-cmd "example2")  
nil
```

(vl-acad-defun *<function>*)

使 **(defun)** 定义的 LISP 函数可以用作 ObjectARX 应用程序中的 c: 形式的函数。这使得 ObjectARX 应用程序可以访问该函数。

参数

<function> 表示函数名称的引用符号。

示例

```
(vl-acad-defun 'example1)
```

(vl-acad-undefun *<function>*)

取消定义先前使用 **(vl-acad-defun)** 函数公开的命令。如果成功则返回 T, 否则返回 nil。

参数

<function> 表示函数名称的引用符号。

示例

```
Command: (vl-acad-undefun 'example1)  
T
```

注册表函数

Visual LISP 提供了访问和修改 Windows® 注册表的特殊函数。可以使用这些函数查询和修改本地注册表的 `HKEY_LOCAL_MACHINE` 和 `HKEY_CURRENT_USER` 配置单元中的键。使用 Visual LISP 注册表函数无法获得远程注册表访问。也不能从 Visual LISP 访问 `HKEY_USERS`、`HKEY_CLASSES_ROOT` 或 `HKEY_CURRENT_CONFIG` 注册表配置单元。

请注意，即使在 Visual LISP 可以访问的注册表配置单元中，你仍然受限于进程所有者的安全环境从而限制了你的访问。换句话说，如果 Visual LISP 应用程序由对该计算机具有有限权限的用户执行，则某些注册表项可能无法访问或可能无法由 Visual LISP 修改。在使用组策略修改注册表访问权限的网络环境中，这个问题需要重点考虑。

```
(vl-registry-read <regkey> [<value-name>])
```

如果注册表项在注册表中有定义，则返回分配给显式注册表项或注册表值名称（符号）的值。如果没有找到这样的注册表项或值名称，则结果为 `nil`。

参数

`<regkey>` 在 `HKEY_LOCAL_MACHINE` 或 `HKEY_CURRENT_USER` 配置单元中的注册表项名称。
`[<value-name>]` 指定注册表项下的从属值符号的名称。

示例

```
Command: (vl-registry-write "HKEY_CURRENT_USER\\Example1" "FOO" "123")  
"123"
```

```
Command: (vl-registry-read "HKEY_CURRENT_USER\\Example1" "FOO")  
"123"
```

```
Command: (vl-registry-read "HKEY_CURRENT_USER\\Example1")  
nil
```

```
Command: (vl-registry-write "HKEY_CURRENT_USER\\Example2" "" "ABCDEF")  
"ABCDEF"
```

```
Command: (vl-registry-read "HKEY_CURRENT_USER\\Example2")  
"ABCDEF"
```

(vl-registry-write regkey <regkey> [*<value-name>*] <value>)

将值 *<value>* 写入注册表项或注册表项值名称，如果成功则返回值 *<value>*。如果不成功则返回 *nil*。

参数

<i><regkey></i>	注册表项名称；
[<i><value-name></i>]	指定注册表项下的从属值符号的名称；
<i><value></i>	要写入指定注册表项或值名称的值。

示例

```
Command: (vl-registry-write "HKEY_CURRENT_USER\\Example1" "TEST1" "123")  
"123"
```

```
Command: (vl-registry-write "HKEY_CURRENT_USER\\Example1" "" "456")  
"456"
```

(vl-registry-delete <regkey> [*<value-name>*])

从注册表中的指定位置删除注册表项及其关联值。成功返回 *T*，失败返回 *nil*。如果提供了 [*<value-name>*] 并且不是 *nil*，则指定的值将从注册表中清除。如果 [*<value-name>*] 不存在或为 *nil*，该函数将删除指定的键及其所有值。如果存在任何子项，则无法删除注册表项。要删除具有子键的键，必须使用 **(vl-registry-descendents)** 收集子键并先删除它们。

参数

⟨regkey⟩ 注册表项名称;
 [⟨value-name⟩] 指定注册表项下的从属值名(符号)的名称。

示例

```
Command: (vl-registry-write "HKEY_CURRENT_USER\\Example1" "TEST1" "123")
"123"
```

```
Command: (vl-registry-delete "HKEY_CURRENT_USER\\Example1")
T
```

(vl-registry-descendents ⟨regkey⟩ [⟨value-names⟩])

参数

⟨regkey⟩ 注册表项名称;
 [⟨value-names⟩] 包含注册表项 ⟨regkey⟩ 条目值的字符串。

示例

```
Command: (vl-registry-descendents "HKEY_LOCAL_MACHINE\\SOFTWARE")
("WexTech Systems" "Voice" "Synaptics" "Symantec" "Secure" "Program Groups
  ↳ " "Policies" "ODBC" "Microsoft" "MetaStream" "McNeel" "Intel
  ↳ Corporation" "INTEL" "InstalledOptions" "Helios" "DOSLib" "Dell
  ↳ Computers" "Dell Computer Corporation" "Dell Computer" "Clients" "
  ↳ Classes" "BVRP Software" "Autodesk" "ATI Technologies" "Apple
  ↳ Computer, Inc." "Adobe")
```

打开位于 AutoCAD 安装位置即 `%PROGRAMFILES%\Autodesk\AutoCAD 20xx\Sample\VisualLISP\regdump.lsp`, 可以看到更多注册表函数示例。在此文件中, 可以找到一个名为 `(registry-tree-dump)` 的有用函数, 它对指定注册表项下的所有子项和值名称执行递归搜索。



你可以创建一对 `(*Get)` 和 `(*Set)` 函数来存储和检索注册表值, 并对标准化位置和错误捕获进行一些控制。你可能会发现以下两个函数很有用:

```
(setq G$REGROOT "HKEY_CURRENT_USER\\Software\\MyApplication\\")

(defun RegGet (key default / val)
```

```
(if (= nil (setq val (vl-registry-read (strcat G$REGROOT key))))
  (progn
    (regset key default)
    (setq val (vl-registry-read (strcat G$REGROOT key)))
  )
)
(if val val default)
)

(defun RegSet (key val)
  (vl-registry-write (strcat G$REGROOT key) "" val)
)
```

反应器与回调



本部分的部分内容源自 AutoCAD® 2004 联机帮助文档，并进行了一些修改以提供更多示例或清晰说明。



虽然反应器确实非常强大且对开发人员有用，但在使用时却要特别谨慎。根据在给定情况下定义的反应器的类型和体积，系统资源可以轻松快速地被耗尽并导致 AutoCAD 变得无响应甚至不稳定或崩溃。在选择如何将反应器应用于应用程序开发时要小心。

反应器只是 AutoCAD 和应用程序之间的链接，允许创建响应 AutoCAD 中发生的事件的函数。例如，可以创建一个反应器来通知应用程序某个实体已被删除。然后应用程序可以执行一些操作来响应此事件。表单上的按钮是事件驱动编程的一个简单示例，它使用事件和响应来执行操作。当选择按钮时，它会触发一个事件，很像信号或广播。此事件由某种反应器检测到，该反应器通过使用所谓的回调过程执行某些操作作为结果。

在 AutoCAD 中，可考虑这种方案，使用 `CommandWillStart` 事件触发命令反应器回调，根据执行过的命令执行某些操作。比如也许用户启动了 `HATCH` 命令，希望通过触发一个回调函数来对此作出反应，该函数在将 `HATCH` 放置在图形中之前将特殊层设置为活动状态，然后在命令完成时恢复之前的层状态^①。如果希望它在命令因错误而失败时也恢复之前的图层状态，或者如果用户只是在中途取消命令怎么办？这可以使用反应器和 Visual LISP 编程。

首先需要做的是定义将在回调中使用的函数。如果它作为命令反应器的结果被调用，切记不要在该函数的任何位置使用 `(command)` 或 `(vl-cmdf)`，因为这可能会开始无限循环并使 AutoCAD 崩溃。听起来像是常识，是吧？有时这样的事情并不那么明显，可能会导致大问题。这只是开发人员在考虑反应器时要非常小心的原因之一。

^① 译者注：现在这个问题在 AutoCAD® 2016 以及更高的版本中可以通过系统变量 `HPLAYER` 来解决，`HATCH` 命令创建的填充都会被直接创建在由 `HPLAYER` 指定的图层上。当然，这里举这个例子是对反应器的使用场景进行说明，注意领会精神。

下一步要做的就是定义反应器，并且构建这个反应器，当事件检测遇到合适的条件时，它能调用回调函数。（考虑命令是 `HATCH` 或 `BHATCH`，先忽略所有其它命令）。

例 12.1^① 演示了如何使用命令反应器来响应 `HATCH` 命令或 `BHATCH` 命令，其方法是定义 "HATCHING" 图层并将其设置为活动状态，直到命令完成（发生 `CommandEnded` 事件）或由于错误而中止（发生 `CommandFailed` 事件），或用户取消（发生 `CommandCancelled` 事件）。

12.1 Visual LISP 反应器函数

函数	说明
<code>(vl-load-com)</code>	加载 Visual LISP 反应器支持函数和其它 Visual LISP 扩展
<code>(vlr-acdb-reactor <data> <callbacks>)</code>	构造一个全局的数据库反应器对象
<code>(vlr-add <obj>)</code>	启动一个被禁用的反应器对象
<code>(vlr-added-p <obj>)</code>	测试以确定指定的反应器对象是否已启用
<code>(vlr-beep-reaction [(args)])</code>	产生声音提示的回调函数
<code>(vlr-current-reaction-name)</code>	如果在反应器回调函数中调用该函数，它返回当前事件的名称（符号）
<code>(vlr-data <obj>)</code>	返回与反应器相关的应用程序特定数据
<code>(vlr-data-set <obj> <data>)</code>	覆盖与反应器相关的应用程序特定数据
<code>(vlr-deepclone-reactor <obj> <data>)</code>	构造编辑器反应器对象，在发生深度克隆事件时给出通知
<code>(vlr-docmanager-reactor <obj> <data>)</code>	构造反应器对象，通知与图形文档相关的事件
<code>(vlr-dwg-reactor <obj> <data>)</code>	构造编辑器反应器对象，通知图形事件（如打开或关闭图形文件）
<code>(vlr-dxf-reactor <obj> <data>)</code>	构造编辑器反应器对象，通知与读写 DXF 文件相关的事件
<code>(vlr-editor-reactor <data> <callbacks>)</code>	构造全局的编辑器反应器对象

^① 译者注：译者手中的文件并没有给出这样一个示例，因此这里无法给出示例的源代码。

函数	说明
<code>(vlr-linker-reactor <data> <callbacks>)</code>	构造全局的链接反应器对象，在每次应用程序加载或卸载 ObjectARX 应用程序时通知应用程序
<code>(vlr-miscellaneous-reactor <data> <callbacks>)</code>	构造编辑器反应器对象，它不属于任何其他编辑器反应器类型
<code>(vlr-mouse-reactor <data> <callbacks>)</code>	构造编辑器反应器对象，通知鼠标事件（如双击）
<code>(vlr-notification <reactor>)</code>	确定当反应器相关联的名称空间不在活动状态时是否激发反应器
<code>(vlr-object-reactor <owners> <data> <callbacks>)</code>	构造对象反应器对象
<code>(vlr-owner-add <reactor> <owner>)</code>	将对象添加到对象反应器的所有者列表
<code>(vlr-owner-remove <reactor> <owner>)</code>	从对象反应器的所有者列表中删除对象
<code>(vlr-owners <reactor>)</code>	返回对象反应器的所有者列表
<code>(vlr-pers <reactor>)</code>	使反应器成为永久反应器
<code>(vlr-pers-list [(reactor)])</code>	返回由当前图形文档中永久反应器组成的表
<code>(vlr-pers-p <reactor>)</code>	确定反应器是否是永久反应器
<code>(vlr-pers-release <reactor>)</code>	使反应器成为临时反应器
<code>(vlr-reaction-name <reactor-type>)</code>	返回由该类型反应器所有可能回调条件组成的表
<code>(vlr-reaction-set <reactor> <event> <function>)</code>	添加或替换反应器中的一个回调函数
<code>(vlr-reactions <reactor>)</code>	返回反应器形如 <code>((<evt-name> . <cb_func>))</code> 的点对表
<code>(vlr-reactors [(reactor)...])</code>	返回由现有反应器组成的表
<code>(vlr-remove <reactor>)</code>	禁用反应器
<code>(vlr-remove-all <reactor-type>)</code>	禁用指定类型的所有反应器
<code>(vlr-set-notification <reactor> <range>)</code>	确定当相关联的名称空间不在活动状态时是否执行反应器回调函数

函数	说明
<code>(vlr-sysvar-reactor <data> <callbacks>)</code>	构造编辑器反应器对象，在修改系统变量时通知应用程序
<code>(vlr-toolbar-reactor <data> <callbacks>)</code>	构造编辑器反应器对象，在工具栏中的位图改变时通知应用程序
<code>(vlr-trace-reaction)</code>	预定义回调函数，在 跟踪 窗口显示一个或多个回调参数
<code>(vlr-type <reactor>)</code>	返回代表反应器类型的符号
<code>(vlr-types)</code>	返回由所有反应器类型组成的表（见 12.2）
<code>(vlr-undo-reactor <data> <callbacks>)</code>	构造通知放弃操作的编辑器反应器
<code>(vlr-wblock-reactor <data> <callbacks>)</code>	构造编辑器反应器对象，在发生与写块相关的事件时通知应用程序
<code>(vlr-window-reactor <data> <callbacks>)</code>	构造编辑器反应器对象，在发生与移动或缩放 AutoCAD 窗口相关的事件时通知应用程序
<code>(vlr-xref-reactor <data> <callbacks>)</code>	构造编辑器反应器对象，在发生附着或修改外部参照事件时通知应用程序

12.2 反应器类型

AutoCAD 反应器有多种类型，每种反应器对应一个或多个 AutoCAD 事件。不同类型的反应器可以分为如下五大类：

- **数据库反应器**

当图形数据库发生特定类型事件，如当对象被添加到图形数据库时，数据库反应器将通知应用程序。

- **文档反应器**

如果当前图形文档发生改变（如打开新的图形文档、激活其他文档窗口、改变文档的锁定状态等），文档反应器将通知应用程序。这并不包括所有包含于编辑器反应器中的事件。

- **编辑器反应器**

在调用 AutoCAD 命令（如打开图形、关闭图形、保存图形、输入输出 DXF 文件、改变系统变量的值等）时，编辑器反应器将通知应用程序。

- **链接反应器**
当加载和卸载 应用程序时，链接反应器将通知应用程序。
- **对象反应器**
当特定对象被修改、复制或删除时，对象反应器将通知应用程序。

除编辑器反应器外，每个反应器类别都有一种反应器。表 12.2 列出了在 Visual LISP 环境中识别每种反应器类型的名称：

表 12.2 反应器类型

反应器类型	说明
:VLR-AcDb-Reactor	数据库反应器
:VLR-DocManager-Reactor	文档管理反应器
:VLR-Editor-Reactor	通用编辑器反应器，为向后兼容而保留
:VLR-Linker-Reactor	链接反应器
:VLR-Object-Reactor	对象反应器

从 AutoCAD® 2000 版本开始，编辑器反应器被分为更明确的几种反应器类型。在这其中，:VLR-Editor-Reactor 类型是为考虑向后兼容才保留的，但在 AutoCAD® 2000 中引入的新编辑器反应器不能通过 :VLR-Editor-Reactor 引用。表 12.3 列出了从 AutoCAD® 2000 开始可用的编辑器反应器类型：

表 12.3: 编辑器反应器类型

反应器类型	说明
:VLR-Command-Reactor	通知命令事件
:VLR-DeepClone-Reactor	通知 DeepClone 事件
:VLR-DWG-Reactor	通知图形事件（例如，打开或关闭图形文件）
:VLR-DXF-Reactor	通知和读写 DXF 文件相关的事件
:VLR-Insert-Reactor	通知和插入块有关的事件
:VLR-Lisp-Reactor	通知 LISP 事件
:VLR-Miscellaneous-Reactor	（表中未列出的）其他编辑器反应器类型
:VLR-Mouse-Reactor	通知鼠标事件（例如双击）

表 12.3: 编辑器反应器类型 (续)

反应器类型	说明
:VLR-SysVar-Reactor	通知对系统变量的修改
:VLR-Toolbar-Reactor	通知对工具栏上位图的修改
:VLR-Undo-Reactor	通知 Undo 事件
:VLR-Wblock-Reactor	通知和写块有关的事件
:VLR-Window-Reactor	通知和移动 AutoCAD 窗口或改变 AutoCAD 窗口大小有关的事件
:VLR-XREF-Reactor	通知和附着或修改外部参照有关的事件



用 `(vlr-types)` 函数可返回反应器类型的完整列表。

对每种反应器，都有一些事件可使它通知应用程序，这些事件被称为回调事件，因为它们将触发反应器调用与该事件相关的函数。例如，当发出 `Save` 或 `QSAVE` 命令保存图形时，会发生 `:vlr-beginSave` 事件。当保存过程结束时，会发生 `:vlr-saveComplete` 事件。在设计基于反应器的应用程序时，要确定所感兴趣的事件，并编写这些事件发生时所要激活的回调函数。

`(vlr-reaction-names <reactor> <type>)`

返回与给定反应器类型相关的所有事件组成的表，例如：下述命令返回和对象反应器相关的所有事件组成的表：

```
Command: (vlr-reaction-names :VLR-Object-Reactor)

(:VLR-cancelled :VLR-copied :VLR-erased :VLR-unerased :VLR-goodbye :VLR-
  ↪ openedForModify :VLR-modified :VLR-subObjModified :VLR-modifyUndone
  ↪ :VLR-modifiedXData :VLR-unappended :VLR-reappended :VLR-
  ↪ objectClosed)
```



如果该命令或其他任何 `(vlr-*)` 命令失败，并出现 "no function definition" 消息，那么有可能是忘了调用用于加载 Visual LISP 反应器支持函数的 `(vl-load-com)` 函数。

在 Visual LISP 加载并运行如下代码，可以打印出所有相关反应器事件列表（按反应器类

型排序):

```
(defun print-reactors-and-events ()
  (foreach rtype (vlr-types)
    (princ (strcat "\n" (vl-princ-to-string rtype)))
    (foreach rname (vlr-reaction-names rtype)
      (princ (strcat "\n\t" (vl-princ-to-string rname))))))
  (princ))
```

12.3 核实反应器类型

《AutoLISP 参考手册》列出了与各种反应器类型相关的所有事件。对每种反应器类型，这些信息包括在定义该类型反应器的函数说明之中。这些函数的名称和反应器类型相同，只是没有前面的冒号。例如，`(vlr-acdb-reactor)` 创建数据库反应器，`(vlr-toolbar-reactor)` 创建工具栏反应器，依此类推。

`(vlr-type <reactor>)`

迭代集合的成员对象并对每一成员对象执行语句。如果参数不是集合对象则产生错误。和 `(foreach)` 一样，引用的符号是局部的和临时的。

参数

`<reactor>` 反应器对象

返回值

表示反应器类型的符号。表 12.4 列出了 `(vlr-type)` 可能返回的类型：

表 12.4: `(vlr-type)` 返回的反应器类型

反应器类型	说明
<code>:VLR-AcDb-Reactor</code>	数据库反应器
<code>:VLR-Command-Reactor</code>	通知命令事件的编辑器反应器
<code>:VLR-DeepClone-Reactor</code>	通知 <code>DeepClone</code> 事件的编辑器反应器
<code>:VLR-DocManager-Reactor</code>	文档管理反应器

表 12.4: (vlr-type) 返回的反应器类型 (续)

反应器类型	说明
:VLR-DWG-Reactor	通知图形事件（如打开或关闭图形）的编辑器反应器
:VLR-DXF-Reactor	通知 DXF 文件读写操作相关事件的编辑器反应器
:VLR-Editor-Reactor	一般的编辑器反应器。用于向后兼容
:VLR-Insert-Reactor	通知块插入相关事件的编辑器反应器
:VLR-Linker-Reactor	链接反应器
:VLR-Lisp-Reactor	通知 LISP 事件的编辑器反应器
:VLR-Miscellaneous-Reactor	不属于其他反应器类型的编辑器反应器
:VLR-Mouse-Reactor	通知鼠标事件（如双击）的编辑器反应器
:VLR-Object-Reactor	对象反应器
:VLR-SysVar-Reactor	通知系统变量变化的编辑器反应器
:VLR-Toolbar-Reactor	通知工具栏位图变化的编辑器反应器
:VLR-Undo-Reactor	通知放弃操作的编辑器反应器
:VLR-Wblock-Reactor	通知和写块相关的操作的编辑器反应器
:VLR-Window-Reactor	通知移动或缩放 AutoCAD 窗口事件的编辑器反应器
:VLR-XREF-Reactor	通知与附着或修改外部参照相关的事件的编辑器反应器

示例

```
Command: (vlr-type circleReactor)
:VLR-Object-Reactor
```

获取反应器的信息有很多种方法。Visual LISP 提供了 AutoLISP 函数来查询反应器，可以用基本的 Visual LISP 数据检查反应器的信息。

使用 AutoLISP 列出图形文件中的所有反应器，可调用 (vlr-reactors) 函数。该函数返回反应器列表的表。每个反应器列表都是由反应器类型的标识符开始的，其后跟着的是该类型的每个反应器的指针。例如：

```
Command: (vlr-reactors)
((:VLR-Object-Reactor #<VLR-Object-Reactor>) (:VLR-Editor-Reactor #<VLR-
  ↪ Editor-Reactor>))
```

在该例中, (vlr-reactors) 返回了包含两个表的表, 一个是单一对象反应器标识, 另一个是单一编辑器反应器标识。

要列出给定类型的所有反应器, 就需要提供给 (vlr-reactors) 该反应器类型标识的参数。指定由 (vlr-types) 函数返回的哪一类型的值; 至于不同的类型可在 12.2 节中查到。例如, 以下列出了所有 DWG 反应器的表:

```
Command: (vlr-reactors :vlr-dwg-reactor)
((:VLR-DWG-Reactor #<VLR-DWG-Reactor> #<VLR-DWG-Reactor>))
```

在这种情况下, 返回值是包含有一个表的表。这个表列出了两个 DWG 反应器的标识符指针。

12.4 使用对象反应器

和其他 AutoCAD 反应器不同, 对象反应器是附着在特定的 AutoCAD 图元(对象)上。定义对象反应器时, 必须指定反应器所要附着的图元。创建对象反应器的函数 (vlr-object-reactor) 要求的参数如下所示:

- vla-object 对象表, 指定通知反应器的图形对象, 这些对象也被称为反应器的所有者。
- 反应器对象关联的 AutoLISP 数据。
- 指明事件和与该事件相关联的回调函数的点对表 ((evt-name) . (cb_func))。



如果对象被包括在对象反应器的所有者列表之中, 那么不能在回调函数中修改该对象。这样做会导致错误消息, 并使 AutoCAD 崩溃。这是使用反应器的一个非常深奥的问题: 仔细规划实施策略, 以避免循环引用的可能性, 其中反应器回调影响反应器本身的来源之一。

例如, 下列语句定义了只有一个所有者(由 myCircle 指定的对象)的对象反应器, 然后将字符串 "Circle Reactor" 附着到反应器上, 并告诉 AutoCAD 当用户修改 myCircle 时调用 (print-radius) 函数:

```
(setq
  circleReactor
```

```
(vlr-object-reactor (list myCircle)
  "Circle Reactor" '(:vlr-modified . print-radius)))
```

反应器对象存储在变量 `circleReactor` 中; 可以使用此变量引用反应器。定义所有者列表时, 必须仅指定 `vla-object` 对象, 不允许使用 `ENAME` 对象, 因为回调函数只能使用需要 `vla-object` 对象的 `ActiveX` 方法修改 AutoCAD 对象。

注意尽管不能在回调反应器中使用由 `(entlast)` 和 `(entget)` 等函数获取的对象, 但可以用 `(vlax-ename->vla-object)` 函数将 `ENAME` 对象转换成 `vla-object` 对象, 关于该函数的详细信息, 请参见《AutoLISP 参考手册》。

以下代码示例绘制一个圆 (`Circle`) 并将反应器应用于该圆以通知此后对该图元所做的任何更改。加载代码, 绘制圆, 然后返回并使用 `SCALE` 或夹点编辑移动或调整圆的大小, 以查看其工作原理。

```
(vl-load-com)
(setq oAcad (vlax-get-acad-object)
  oDoc (vla-get-activedocument oAcad) )
(cond
  ( (and
    (setq ctrPt (getpoint "\nCenter point: "))
    (setq rad (distance ctrPt (getpoint ctrPt "\nRadius: ")))
    (setq CircleObject
      (vla-addCircle
        (vla-get-ModelSpace oDoc)
        (vlax-3d-point ctrPt)
        rad)))
    (if CircleObject
      (setq circleReactor
        (vlr-object-reactor (list CircleObject) "Circle Reactor"
          '(:vlr-modified . rShowRadius))))
    (defun rShowRadius
      (notifier-object reactor parameter-list)
      (cond
        ( (vlax-property-available-p notifier-object "Radius")
          (princ "*** The radius is ")
          (princ (vla-get-radius notifier-object))))))
```


12.5 将数据附着到反应器对象

在 12.4 中，创建对象反应器的示例在对 `(vlr-object-reactor)` 的调用中包含一个字符串 "Circle Reactor"。指定要包含在反应器中的数据并不是必须的，可以改为指定 `nil`。然而，一个对象上可能附着有好几个反应器，如果给反应器指定文本字符串或应用程序能用的其他数据，就可以区分附着在同一对象上的不同反应器了。

12.6 通过 VLIDE 检验反应器

可以使用 VLIDE 的 **检验** 工具检验反应器。例如，使用 12.4 中定义的对象反应器将返回到变量 `circleReactor` 中。如果打开该变量的检验窗口（图 12.1），Visual LISP 将显示如下信息：

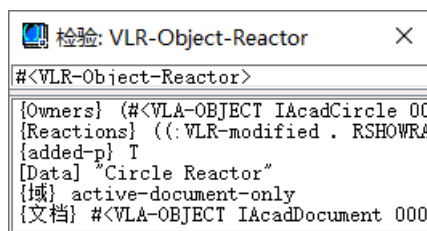


图 12.1 变量的检验窗口

检验列表中显示以下信息：

- 该反应器的所有者对象
 - 事件和与之相关联的回调反应器
 - 该反应器是否是活动的
 - * 如果 `added-p` 为 T，反应器为活动的；
 - * 如果 `added-p` 是 nil，反应器为非活动的。
- 附着到反应器上的用户数据
 - 触发该反应器的文档范围
 - * 0 - 反应器仅在创建它的图形文档中触发；
 - * 1 - 反应器可响应任何文档中的相关事件。
 - 详细信息请参见 12.10 反应器与多重名称空间。
- 附加到对象反应器的 AutoCAD 文档。

双击以 `{Owners}` 开头的条目，可查看所有者对象列表（图 12.2）：

双击检验列表框中的列表项向下深入以查找所有者。

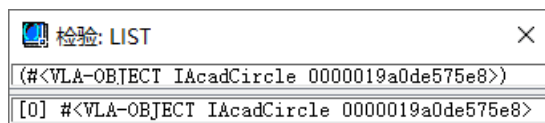


图 12.2 查看所有对象列表

12.7 查询反应器

Visual LISP 还提供了从应用程序中或在控制台提示符下检查反应器定义的函数：

- `(vlr-type)` 返回指定反应器的类型，例如：

```
Command: (vlr-type circleReactor)
:VLR-Object-Reactor
```

- `(vlr-current-reaction-name)` 返回触发回调函数的事件名称。
- `(vlr-data)` 返回附着到反应器的特定应用程序数据，如：

```
Command: (vlr-data circleReactor)
"Circle Reactor"
```

可用该数据区分触发同一个回调函数的多个反应器。

- `(vlr-owners)` 返回向某对象反应器发出通知的 AutoCAD 图形中的对象（反应器所有者）列表，下述函数调用将列出 `circleReactor` 的所有者：

```
Command: (vlr-owners circleReactor)
(#<VLA-OBJECT IAcadCircle 03ad077c>)
```

- `(vlr-reactions)` 函数将返回指定反应器的回调条件与回调函数构成的点对表，下例将返回 `circleReactor` 的相关信息：

```
Command: (vlr-reactions circleReactor)
((:vlr-modified . PRINT-RADIUS))
```

12.8 临时反应器和永久反应器

反应器有临时和永久之分。当绘图文档关闭时，临时反应器会丢失；这是默认的反应器模式。永久反应器与绘图文档一起保存，在下次打开绘图文档时依然存在。如果使用通过回调调

用自定义应用程序的永久反应器，则必须加载自定义应用程序才能使回调正常工作。有关永久反应器的更多信息，请参见 12.9。

用函数 `(vlr-pers)` 可使反应器成为永久反应器。如果要去掉反应器的永久性而使它成为临时反应器，可使用函数 `(vlr-pers-release)`。这两个函数都将反应器对象作为其唯一参数。例如，如下命令将使反应器成为永久反应器：

```
Command: (vlr-pers circleReactor)
#<VLR-Object-Reactor>
```

如果成功，`(vlr-pers)` 返回指定的反应器对象。

如果要确定某反应器对象是临时反应器还是永久反应器，可使用函数 `(vlr-pers-p)`，例如：

```
Command: (vlr-pers-p circleReactor)
#<VLR-Object-Reactor>
```

如果该反应器是永久反应器，`(vlr-pers-p)` 函数返回该反应器对象，否则返回 `nil`。

12.9 打开包含永久反应器的文档

由于反应器只是事件和回调函数之间的链接。虽然此链接仍然存在，但回调函数本身不是反应器的一部分，通常也不是绘图文件的一部分。保存在图形中的反应器只有在其关联的回调函数加载到 AutoCAD 中时才可用。如果将反应器和回调函数定义在独立名称空间 `VLX` 中，则可以在打开图形文件时自动加载回调函数。

如果打开包含 Visual LISP 反应器信息的绘图文档并且未加载关联的回调函数，AutoCAD 会显示一条错误消息。可以使用 `(vlr-pers-list)` 函数返回绘图文档中所有永久反应器的列表。

12.10 反应器与多重名称空间

当前的 AutoLISP 实现支持一次在一个绘图文档中工作。某些 AutoCAD 的 API（例如 ObjectARX 和 VBA）确实支持应用程序在多个文档中同时工作。因此，应用程序可能会修改当前未激活的已被打开的绘图文档。但在 AutoCAD 中还不支持该功能。

12.11 反应器使用规则

正如我在本章开头提到的，需要仔细规划和考虑反应器的性能和稳定性。以下准则是在 AutoCAD 联机帮助文档中提供的，非常值得考虑。

使用反应器时，请尽量遵守以下准则。如果反应器的内部实现发生变化，违反这些准则可能会给应用程序带来不可预测的结果。

- 不要依赖反应器通知的顺序

除了少数特例外，建议不要依赖反应器通知的顺序。例如，运行 `OPEN` 命令能够触发 `BeginCommand`、`BeginOpen`、`EndOpen` 和 `EndCommand` 事件，但是，它们可能不会按该顺序出现。唯一可以安全依赖的事件序列是 `Begin*` 事件总是会在相应的 `End*` 事件之前发生。例如，`commandWillStart()` 总是出现在 `commandEnded()` 之前，`beginInsert()` 总是出现在 `endInsert()` 之前。如果由于将来引入新通知导致重新排列现有通知更改序列，则依赖更复杂的序列可能会导致应用程序出现问题。

- 不要依赖通知之间的函数调用顺序

通知之间调用函数的顺序也是不能保证的。例如，当在对象 A 上收到 `:vlr-erased` 通知时，这意味着对象 A 已被删除。如果在 A 上收到 `:vlr-erased` 通知，然后在 B 上收到 `:vlr-erased` 通知，这意味着对象 A 和 B 都被删除了；但这不能确保 B 在 A 之后被删除。如果将应用程序绑定到这种详细程度，那么它很可能在未来的版本中崩溃。所以不应依赖序列，而是应该依赖反应器来指示系统的状态。

- 不要在反应器回调函数中使用任何交互函数（例如 `(getPoint)`、`(entsel)`）

尝试从反应器回调函数中执行交互函数可能会导致严重问题，因为 AutoCAD 可能在触发事件时仍在处理命令。因此，避免使用 `(getPoint)`、`(entsel)`、`(getkeyword)` 等输入获取方式，以及选择集操作和 `(command)` 函数。

- 不要从事件处理程序中启动对话框

对话框被视为交互式功能，可能会干扰 AutoCAD 的当前操作。但是，消息框和警告框不被认为是交互式的，可以安全地发出。

- 不要更新发出事件通知的对象

不要尝试从同一对象的回调函数更新对象，因为导致对象触发回调函数的事件此时可能仍在进行中，并且调用回调函数时 AutoCAD 仍在使用该对象。但是，可以安全地从触发事件的对象中读取信息。举个例子，假设有一个铺满瓷砖的地板，并将反应器连接到地板的边界。如果你改变地板的大小，反应器回调函数会自动添加或减去瓷砖来填充新区域。该函数将能够读取边框的新区域，但不能尝试对边框本身进行任何更改。

- 不要从会触发相同事件的回调函数中执行任何操作

如果在反应器回调函数中执行触发相同事件的操作，将创建一个死循环。例如，如果

尝试从 `BeginOpen` 事件中打开一个图形，AutoCAD 将继续打开更多图形，直到达到打开图形的最大数量。

- 在设置之前确认反应器尚未设置，否则可能最终会在同一事件上进行多个回调
- 请记住，当 **AutoCAD 显示模式对话框**时不会触发任何事件



虽然 **VLX** 可以在与加载它的文档不同的名称空间中运行，但它仍然与原始文档相关联，并且不能操作其他文档中的对象。

Visual LISP 对在非活动文档中执行的反应器回调函数提供的支持是有限的。默认情况下，只有当定义反应器的文档处于激活状态的时候，才会在相关事件发生时执行回调函数，但可以使用 `(vlr-set-notification)` 函数更改此行为。

要指定反应器即使定义其的文档未处于活动状态时（例如，如果另一个名称空间中的应用程序触发事件）也能执行其回调函数，请调用以下函数：

```
(vlr-set-notification reactor-object 'all-documents)
```

对于应用程序的所有实例（即，如果不是一个传播到所有会话的单独的命名空间 **VLX** 应用程序）通知事件在其中一个会话中发生，这是很有用的。

要修改反应器，使其仅在定义它的文档处于活动状态时发生事件时才执行其回调函数，可以使用以下命令：

```
(vlr-set-notification reactor-object 'active-document-only)
```

`(vlr-set-notification)` 函数返回指定的反应器对象。例如，无论其关联文档是否处于活动状态，以下命令都定义了一个反应器并将其设置为响应事件：

```
(setq circleReactor
  (vlr-object-reactor
    (list myCircle)
    "Circle Reactor" '(:vlr-modified . print-radius)))
```

```
#<VLR-Object-Reactor>
```

```
Command: (vlr-set-notification circleReactor 'all-documents)
```

```
#<VLR-Object-Reactor>
```

要确定反应器的通知设置，请使用 `(vlr-notification)` 函数。例如：

```
Command: (vlr-notification circleReactor)
all-documents
```

`(vlr-set-notification)` 函数只影响指定的反应器。所有反应器在被创建时，其通知设置都是缺省的 `active-document-only`。



如果选择将反应器设置为即使在其文档未激活为活动文档时也触发执行其回调函数，则回调函数除了设置和读取 AutoLISP 系统变量外什么都不做。执行其他类型的操作可能会导致 AutoCAD 变得不稳定或崩溃！

请注意，VBA 不提供与 `(vlr-CommandCancelled)` 等效的功能，这意味着在由命令反应器管理期间，只有通过 Visual LISP 才能处理用户按下 `[ESC]` 的行为。

创建 Visual LISP 应用程序

Visual LISP 提供的最重要的功能可能是构建和管理应用程序的能力。在这种情况下，我们实际上是在谈论 **VLX** 应用程序，但也可以将 **FAS** 输出视为应用程序。Visual LISP 应用程序的构建与工程的使用密切相关，但它们并非不可分割。

13.1 为什么要创建 **VLX** 应用程序










要制作 **VLX** 应用程序的主要原因是提高安全性和改善性能。安全性之所以得到提高，因为 Visual LISP 编译器将 **LSP** 源代码加密并编译为人眼无法读取的二进制输出。它还可以将多个 **LSP** 文件组合成单个 **VLX** 输出，进一步提高安全性并提供单个输出以交付给用户。性能能够得到改善，是因为编译后的代码实际上在运行时执行起来更高效。




与许多其他编程语言不同，Visual LISP 并不真正“编译”其输出，而是更准确地执行加密和部分编译。这有点像 Java 编译器生成“p 代码”输出，然后在运行时由客户端上的 JVM 编译器编译。Visual LISP **VLX** 代码被编译为二进制输出而不是机器级语言，这意味着它必须在运行时由客户端解释。尽管如此，它仍然提供了一些相对于原始 **LSP** 源代码的性能改进措施。

FAS 文件是 **VLX** 编译期间的中间输出，是 **LSP** 文件编译的产物。**VLX** 模块结合了 **FAS** 文件和任何其他文件类型，将其全部打包为客户端上的单个可加载模块。**VLX** 应用程序可以包括其他文件类型，例如 LISP 代码 (**LSP**)、对话控制语言文件 (**DCL**)、编译 LISP 代码 (**FAS**)、VBA 编译文件 (**DVB**)、ASCII 文本 (**TXT**) 甚至其他 Visual LISP 工程 (**PRV**)。

制作 **VLX** 应用程序最有用的功能之一是可以将多个文件合并到单个 **VLX** 输出中。这使得加载和管理变得容易，并使交付产品保持整洁和紧凑。我们来看看例子。

13.2 编译一个简单的应用程序

在 Visual LISP® 中打开本书 CD 中的  FirstApp.LSP 文件，再打开  FirstApp.DCL 文件。^①现在，从下拉菜单中选择 **文件** > **生成应用程序** > **新建应用程序向导** 菜单项。编译  VLX 应用程序有两种模式：简单模式和专家模式。当你只打算编译  LSP 文件而不打算编译独立名称空间  VLX 时使用简单模式。专家模式允许在  VLX 中包含额外的资源文件，例如  DCL、 DVB、 VLX 和其他文件，并使其成为一个单独的名称空间应用程序。

由于在本练习中我们将使用  DCL 文件将  LSP 文件编译到单个单独的名称空间  VLX 应用程序中，因此必须从向导模式面板中选择**专家模式**（图 13.1）。选择 **下一步** 按钮。

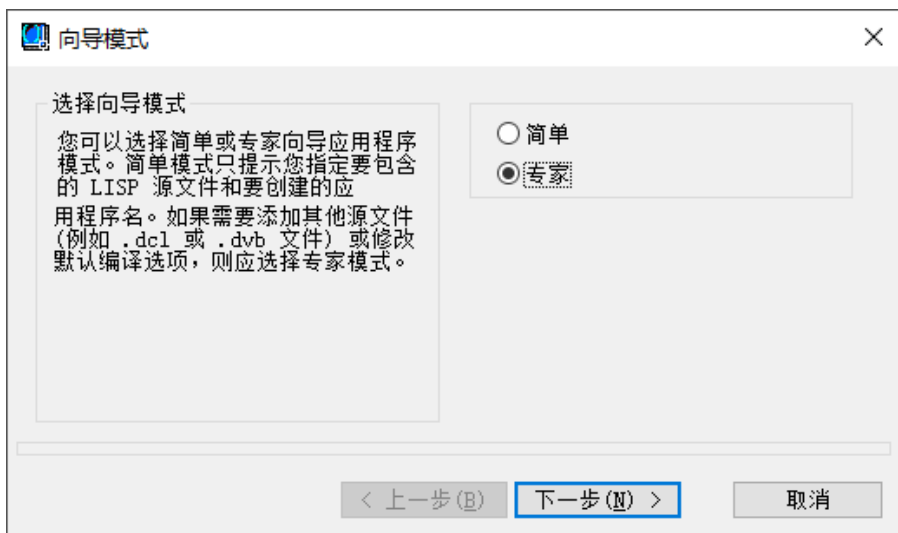









图 13.1 生成应用程序向导

应用程序目录面板（图 13.2）是指定  VLX 文件名和目标输出位置的地方。应用程序位置是流程结束时创建  VLX 文件的位置。应用程序名称是目标  VLX 文件的名称（仅包括基本文件名，不包括扩展名）。在应用程序名称框中键入时，目标文件窗口会显示实际的  VLX 文件名结果。指定应用程序位置并输入应用程序名称 FirstApp 后，选择 **下一步** 按钮继续。

应用程序选项面板（图 13.3）将其设为独立名称空间应用程序，并使用 ActiveX 支持。对于此示例，选中两个选项，然后选择 **下一步** 继续。

要包含的 LISP 文件面板（图 13.4）是选择要包含在  VLX 应用程序中的 LISP 代码文件（ *.LSP）的地方。点击 **添加** 进行浏览，然后选择  FirstApp.LSP 文件。再选择 **下一步** 按钮继续。

^① 译者注：译者没有相关文件，并且出于排版考虑，缩短了这两个文件的文件名。

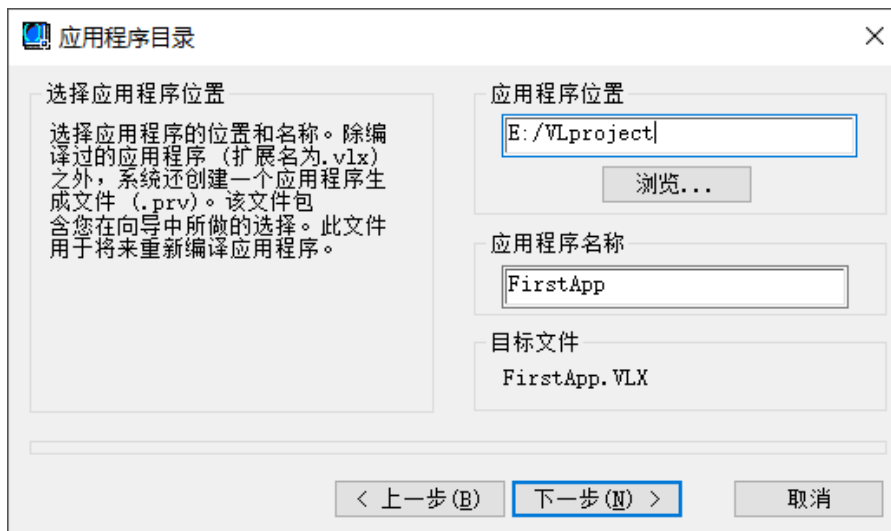


图 13.2 应用程序目录面板

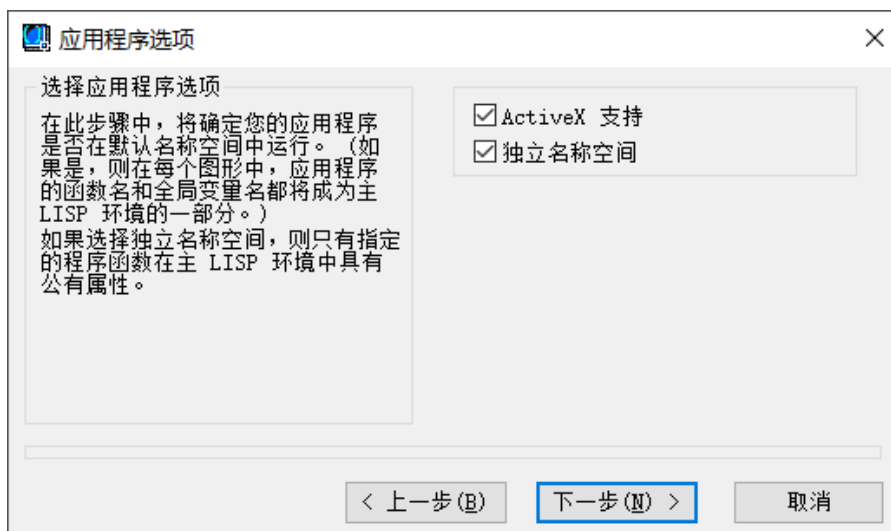


图 13.3 应用程序选项面板（勾选独立名称空间）

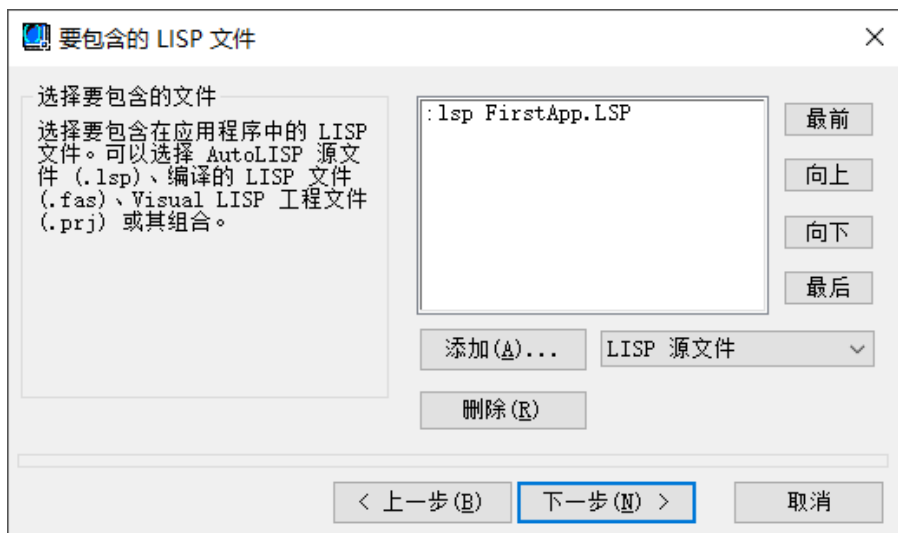






图 13.4 要包含的 LISP 文件面板

要包含的资源文件面板（图 13.5）是选择其他资源文件的地方，例如  DCL 对话框表单文件、 DVB（VBA）文件和其他类型的文件。将文件类型选择更改为  DCL 文件，然后选择 **添加** 按钮以找到并选择  FirstApp.DCL 文件。然后选择 **下一步** 按钮继续。

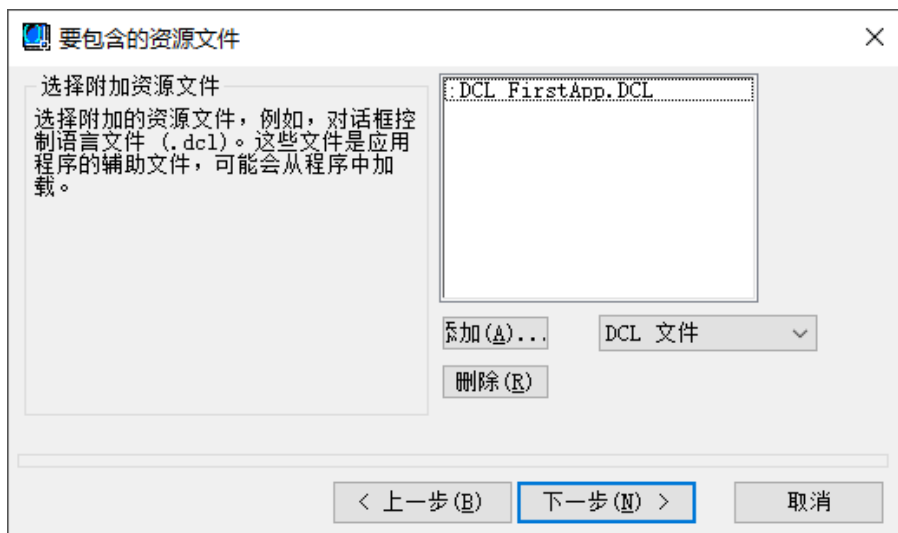


图 13.5 要包含的资源文件面板

下一个面板会提示你选择**标准**或**优化并链接**编译。对于这个例子，使用**标准**选项并选择 **下一步** 按钮继续。

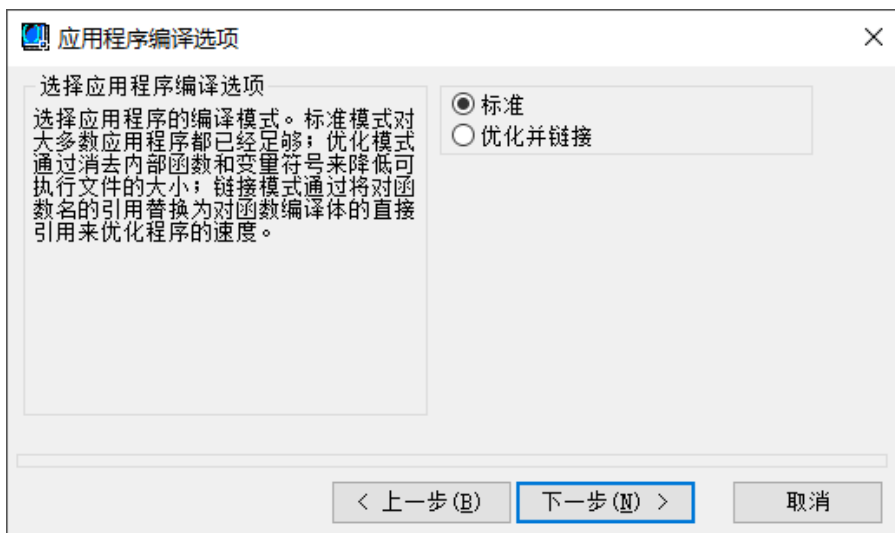


图 13.6 应用程序编译选项面板

最后一个表单询问你是否要保存生成应用程序设置并继续编译 **VLX** 应用程序。如果你选择不编译，你刚刚配置的设置将保存到使用 **PRV** 文件扩展名的文件中。你可以随时使用存储的设置重新编译并节省大量时间。对于此示例，继续并通过选择 **完成** 按钮来编译你的 **VLX** 应用程序。

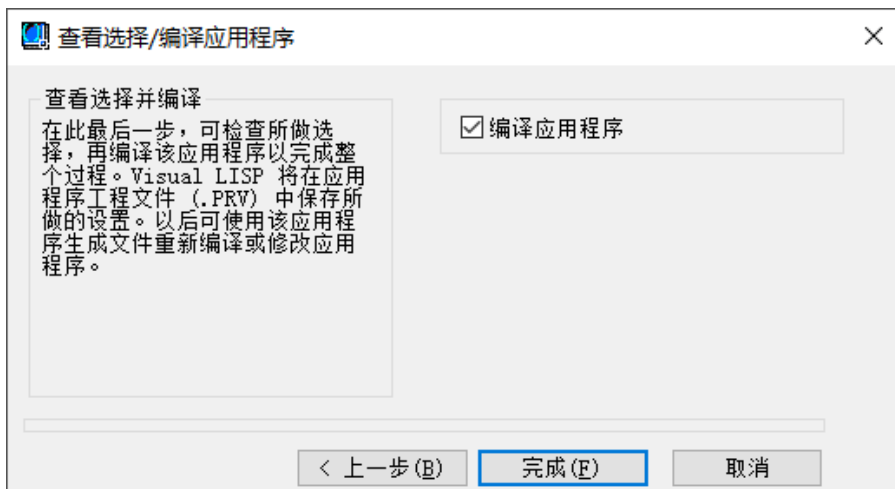


图 13.7 应用程序编译选项面板

现在 **FirstApp.vlx** 已经编译成功，将其加载到 AutoCAD 中并尝试使用 **FIRSTAPP** 命令来查看它是如何工作的。你应该会看到一个带有一个 **确定** 按钮的对话框，并在中间显示一

条消息 “Congratulations!” (图 13.8)。

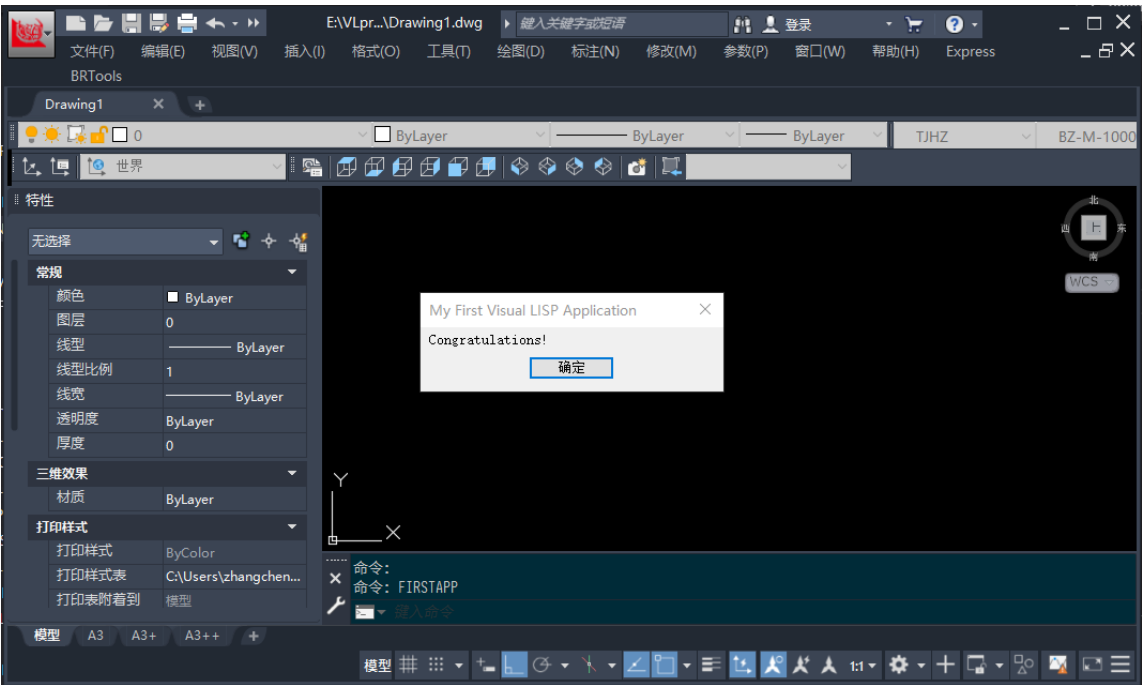


图 13.8 FIRSTAPP 的运行结果

如果没有发生这种情况，请查看本章以确保正确地遵循了所有步骤并再次编译和加载它。要重新加载单独的名称空间 **VLX**，你首先必须使用 `(vl-unload-vlx)` 函数卸载现有定义。要卸载 **FIRSTAPP**，你可以在命令提示符下使用 `(vl-unload-vlx "FirstApp.vlx")`。

13.3 创建 PRV 文件

生成应用程序 向导创建一个 **PRV** 文件来存储你的应用程序的设置。在记事本中打开一个 **PRV** 文件，可以看到它实际上是一个 LISP 格式的文件，其中属性以点对列表的形式存储。下面的示例显示了将 **LSP** 和 **DCL** 文件编译到 **asw_pm.vlx** 输出中的 **PRV**。

```
asw_pm.PRV

;;; Visual LISP make file [V1.0] asw_pm saved to:[C:/] at:[3/05/10]
(PRV-DEF (:target . "FirstApp.VLX")
  (:active-x . T)
  (:separate-namespace))
```

```
(:protected . T)
(:load-file-list (:lsp "source/FirstApp.lsp"))
(:require-file-list (:DCL "source/FirstApplication.DCL"))
(:ob-directory)
(:tmp-directory)
(:optimization . st)
)
;; EOF
```




尽管你可能想在文本编辑器中“调整”图标 PRV 文件, 但你应该改为使用 [文件》生成应用程序》](#) [现有应用程序特性](#) 功能来修改图标 PRV 配置设置。手动编辑图标 PRV 文件可能会损坏文件并在你尝试重新编译时导致错误。

在 Visual LISP 中使用 ObjectDBX

Visual LISP 可以与用户任何支持 ActiveX 的可用资源进行交互。ObjectDBX 是 AutoCAD 中提供的另一种资源，Visual LISP 可以利用它来执行任何其他技术无法完成的特殊任务。首先，我们需要从解释 ObjectDBX 的真正含义开始。

14.1 什么是 ObjectDBX

ObjectDBX 是 ObjectARX 的一个子集，好吧，有一点像。它是一个 C++ 面向对象的 API，用于操作 AutoCAD 及其相关对象、集合、属性、方法和事件。虽然 ObjectDBX 能够实现许多大胆壮举，但与 ObjectARX 和 Visual LISP 相比，它确实有一些局限性。除了限制之外，它还提供了一些优势。有点迷惑？我知道我一开始是的。但是，ObjectDBX 真正大放异彩的一个地方是远程文档访问领域，特别是挖掘非公开图纸。

最近，AUTODESK® 发布了 ObjectDBX SDK，供开发人员在未安装 AutoCAD® 的情况下用于挖掘和操作绘图数据。免费的？当然不是。事实上，如果你想追求这个宝贝，就需要面对高昂的价格标签和许可使用费。你可以选择 OpenDWG 作为替代品，但 ObjectDBX 是由 AUTODESK® 构建的，因此你可以相当确定它在识别  DWG 文件中的所有细微之处时将是可靠的。

然而，为了本章的缘故，我将把重点放在 ObjectDBX 作为 AutoCAD® 中的一个整体服务，以及如何从 Visual LISP® 中使用它来执行 Visual LISP 单独无法完成的某些任务。听起来很有趣？让我们看看这是如何工作的。

14.2 Visual LISP 中的 ObjectDBX

为了在 Visual LISP 中使用 ObjectDBX, 必须首先加载 ObjectDBX 类型库接口, 如例 14.1 所示。然后, 必须使用特殊函数 `(vla-getInterfaceObject)` 调用接口, 如例 14.3 所示。例 14.1 显示了几个示例函数, 用于在 Visual LISP 中加载 ObjectDBX 类型库接口。

假设你希望能够搜索绘图文件目录以找到包含特定块插入的文件。虽然你可以打开每个图形并获取 **Blocks** 集合或进行 `(ssget)` 搜索, 但还有另一种无需在 AutoCAD 编辑器中打开图形即可执行此操作的方法: ObjectDBX。

例 14.1 调用 ObjectDBX接口的 Visual LISP 函数

```
;; Calls REGSVR32 to register a DLL silently via the /S option
(defun DLLRegister (dll)
  (startapp "regsvr32.exe" (strcat "/s \" dll \"")))
;; Calls REGSVR32 to un-register a DLL silently via the /U /S options
(defun DLLUnRegister (dll)
  (startapp "regsvr32.exe" (strcat "/u /s \" dll \"")))
;; Returns the ProgID for a given ClassID if found in registry
(defun ProgID->ClassID (ProgID)
  (vl-registry-read (strcat "HKEY_CLASSES_ROOT\\" progid "\\CLSID")))
;; Registers ObjectDBX (if not already), Returns ProgID if successful
(defun DBX-Register ( / classname)
  (setq classname "ObjectDBX.AxDbDocument.18")
  (cond
    ((ProgID->ClassID classname) )
    (and
      (setq server (findfile "AxDb.dll"))
      (DLLRegister server)
      (ProgID->ClassID classname))
    (not (setq server (findfile "AxDb.dll")))
      (alert "Error: Cannot locate AxDb.dll..."))
    ( T
      (DLLRegister "ObjectDBX.AxDbDocument.18")
      (or
        (ProgID->ClassID "ObjectDBX.AxDbDocument.18")
        (alert "Error: Failed to register ObjectDBX..."))))))
```


`(DLLRegister)` 函数是一种通用工具, 可用于通过外壳操作使用 `REGSVR32` 命令在客户端上执行 Windows® DLL 注册。/s 参数表示静默注册, 它会在注册过程中抑制任何通知。

`(DLLUnRegister)` 函数执行与 `(DLLRegister)` 相反的操作, 它从本地计算机上删除 DLL 的注册。当你需要注册同一 DLL 的更新版本时, 这对于删除 DLL 通常很有用。

`(ProgID->ClassID)` 函数在 Windows® 注册表中执行给定类注册的查找并返回 GUID, 它是给定 ActiveX 组件的冗长编码唯一标识符。GUID 的值是在编译期间由复杂的哈希算法生成的, 任何两个 GUID 的值都是不同的。此特定函数通过检查注册表中的 GUID 来验证给定的 DLL 是否已注册。如果未找到 GUID, 则 DLL 尚未注册, 返回 `nil`。然后你可以使用 `(DLLRegister)` 在客户端计算机上注册 DLL。

例 14.2 中给出的函数打开一个远程绘图文档, 如果成功则返回 DBX 文档对象, 否则返回 `nil`。你可以使用这个函数来处理这些乱七八糟的事情, 只需使用返回的文档对象来执行你想要的任何操作。

例 14.2 打开 ObjectDBX 文件的函数

— □ ×

```

(defun DBX-Doc-Open (filename / dbxdoc)
  (cond
    ( (findfile filename)
      (if (not (DBX-Register))
          (exit))
      (setq dbxdoc
        (vla-getinterfaceobject
          (vlax-get-acad-object) "ObjectDBX.AxDbDocument.18"))
      (cond
        ( (vl-catch-all-error-p
          (vl-catch-all-apply
            'vla-Open (list dbxdoc (findfile filename))))
          (princ "\nUnable to open drawing.")
          (exit))
        ( T dbxdoc )))))

```

现在你有了一个漂亮的小黑盒函数来远程打开绘图文件, 这样你就可以继续包装更大更好的东西, 比如返回表列等等, 还可以通过 DBX 修改远程图形的某些属性。

例 14.3 使用 ObjectDBX 检查图形表的 Visual LISP 函数

```

(defun DBX-GetTableList
  (filename tblname / dbxdoc out name)
  (cond
    ((setq dbxdoc (DBX-doc-open filename))
      (vlax-for tblItem (DBX-TableGet tblName dbxdoc)
        (setq name (vla-get-Name tblItem))
        (if (/= (substr name 1 1) "*")
          (setq out (cons name out))))
      (vlax-release-object dbxdoc))
    (T
      (strcat (princ "\nUnable to open file: " filename))))
  (if out (reverse out)))

(defun DBX-TableGet (tName object)
  (cond
    ((= (strcase tName) "BLOCKS") (vla-get-Blocks object) )
    ((= (strcase tName) "LAYERS") (vla-get-Layers object) )
    ((= (strcase tName) "TEXTSTYLES") (vla-get-textstyles object) )
    ((= (strcase tName) "DIMSTYLES") (vla-get-dimstyles object) )
    ((= (strcase tName) "LINETYPES") (vla-get-linetypes object) )
    ((or
      (= (strcase tName) "PLOTCONFIGURATIONS")
      (= (strcase tName) "PAGESETUPS"))
      (vla-get-plotconfigurations object))
    ((= (strcase tName) "LAYOUTS") (vla-get-Layouts object) )
    ((= (strcase tName) "GROUPS") (vla-get-Groups object) )
    (T
      (vl-exit-with-error
        "\n(dbx-dwgscan error): Invalid table name specified.))))

```

使用例 14.3 中代码的一个示例是查询另一幅图中的图层 (LAYERS) 表。这样做的语法如下：

```
(dbx-gettablelist "c:\\users\\dstein\\drawings\\sample1.dwg" "LAYERS")
```

返回值为：

```
("0" "Alpha" "Beta" "Gamma" "Delta" "Epsilon")
```

例 14.3 中所示的函数使用 ObjectDBX 的 `Open` 方法来访问给定的绘图文件及其中给定的表集合。使用 ObjectDBX 的限制之一是你无法访问在 AutoCAD 文档集合中打开的任何绘图文件的表, 因为这会产生错误。即使绘图文件已被其他用户打开, 只要请求通过 ObjectDBX 接口访问绘图文件的用户未打开该绘图文件, ObjectDBX 就可以进行访问。

例 14.4 dbx-dwgscan.lsp

```
(defun DWGSCAN
  ($table $name $dwgfiles / $files $dwgs $path $collection n out)
  (cond
    ( (and $table $name $dwgfiles)
      (princ
        (strcat
          "\nScanning "
          (itoa (length $dwgfiles))
          " drawings for "
          (strcase (substr $table 1 (1- (strlen $table)))) t)
          " [" $name "]" ..."))
      (foreach n $dwgfiles
        (cond
          ( (setq $collection (DBX-GetTableList n $table))
            (cond
              ( (member (strcase $name) (mapcar 'strcase $collection))
                (setq out (cons n out))))
            (setq $collection nil))
          ( T
            (princ
              "\nUnable to query table collection in drawing."))))
      ( T (princ "\nUsage: (DWGSCAN tablename itemname drawingfiles)"))
      (if out (reverse out)))
```

例 14.4 显示了一个函数, 该函数使用例 14.1 和 14.3 中的函数为指定的表项执行绘图列表的搜索。如果将示例文件 `dbx-dwgscan.lsp` 加载到 AutoCAD 会话中, 则可以使用 `(DWGSCAN)` 函数在其他图形中搜索项目。下面的示例演示了如何使用 `(DWGSCAN)` 在绘图列表中搜索名为 "Chair123" 的块。

```
Command: (dwgscan "Blocks" "Chair123" dwgfiles)

Scanning 51 drawings for block [Chair123]...
("c:\\drawings\\plan003.DWG" "c:\\drawings\\plan004.DWG" "c:\\drawings\\
  ↳ plan005.DWG")
```

关于从 Visual LISP 使用 ObjectARX 服务的一些注意事项：

- 不能通过 DBX 对图形执行选择集操作。只能使用表操作。
- 无法打开执行 DBX 操作中的在 AutoCAD 会话的文档集合中打开的任何文档。
- ObjectARX 不支持对文档使用任何“命令”操作。
- 确保在使用完 DBX 对象后释放它，并在外部进程（AutoCAD 名称空间外部）的任何对象释放之后使用 (gc)。

通过 ObjectARX 公开的文档接口比给定 AutoCAD 编辑会话内部的文档对象的接口要严格得多。下面列出了 ObjectARX 文档对象公开的属性和方法。可以通过对活动的 DBX 文档对象执行 (vlax-dump-object) 看到这一点，例如上面显示的 (DBX-Doc-Open) 函数返回的对象。



一些有趣的注意事项：以文档为中心的系统变量并未公开；因为文档实际上并未以类似于打开绘图进行编辑的方式在 Application 名称空间中打开，Application 对象因此也不存在；Name 属性不是只读的；注意可用的方法。

```
; IAcDbDocument: IAcDbDocument Interface
; Property values:
; Application (RO) = Exception occurred
; Blocks (RO) = #<VLA-OBJECT IAcadBlocks 037aad64>
; Database (RO) = #<VLA-OBJECT IAcadDatabase 037ac634>
; Dictionaries (RO) = #<VLA-OBJECT IAcadDictionaries 037a8a34>
; DimStyles (RO) = #<VLA-OBJECT IAcadDimStyles 037a8954>
; ElevationModelSpace = 0.0
; ElevationPaperSpace = 0.0
; Groups (RO) = #<VLA-OBJECT IAcadGroups 037acd24>
; Layers (RO) = #<VLA-OBJECT IAcadLayers 037acc44>
; Layouts (RO) = #<VLA-OBJECT IAcadLayouts 037acba4>
; Limits = (0.0 0.0 12.0 9.0)
; Linetypes (RO) = #<VLA-OBJECT IAcadLineTypes 037a8e84>
; ModelSpace (RO) = #<VLA-OBJECT IAcadModelSpace 037a8dd4>
; Name = "C:\\Documents and Settings\\dave\\My Documents\\DRAWING3.dwg"
```

```

; PaperSpace (R0) = #<VLA-OBJECT IAcadPaperSpace 037a8d24>
; PlotConfigurations (R0) = #<VLA-OBJECT IAcadPlotConfigurations 037a8bf4>
; Preferences (R0) = #<VLA-OBJECT IAcadDatabasePreferences 037ac694>
; RegisteredApplications (R0) = #<VLA-OBJECT IAcadRegisteredApplications
    ↪ 037a8b34>
; TextStyles (R0) = #<VLA-OBJECT IAcadTextStyles 037a93a4>
; UserCoordinateSystems (R0) = #<VLA-OBJECT IAcadUCSs 037a92f4>
; Viewports (R0) = #<VLA-OBJECT IAcadViewports 037a91e4>
; Views (R0) = #<VLA-OBJECT IAcadViews 037a9124>
; Methods supported:
; CopyObjects (3)
; DxfIn (2)
; DxfOut (3)
; HandleToObject (1)
; ObjectIdToObject (1)
; Open (1)
; RegisterLicenseObject (2)
; RevokeLicenseObject (1)
; Save ()
; SaveAs (1)

```

ObjectDBX 确实是一只酷狗。当大多数程序员发现并了解它的功能时，他们会立即询问它是否可以在 AutoCAD 之外使用。嗯，是的，可以，但是从 AUTODESK® 获得给个人开发者的授权许可是非常昂贵的。

XDATA 和 XRecord

AutoCAD 提供了几种在绘图中存储非图形信息的方式。这其中可以包括各种类型的数据，例如数字、文本等。其中，最常见的两种方式是实体扩展数据（EED）和词典。EED 最常见的形式是 XDATA，它是所有图形实体以及许多表对象（例如图层和线型）的扩展。这允许在这些实体或表对象中存储隐藏（非图形）信息，并在需要时检索信息。

另一种存储非图形信息的形式是通过使用 XRecord 对象。XRecord 对象是 Document 对象的一部分，允许将字符串信息存储在 Dictionary 对象中。

使用 XDATA 的优点是信息附加到特定的实体或表成员。使用 XRecord 对象的优点是它们附加到 Document 本身，而不是任何特定的实体或表对象。此外，XDATA 对可以存储在给定实体或表成员上的数据大小有一定的限制。XRecord 对象不会对数据存储施加任何大小限制，但会影响 DWG 文件大小和打开图形时的内存要求。

15.1 使用 XDATA

XDATA 可以附加到绘图图中的任何图形实体以及许多表对象（例如图层、布局 and 线型）并从中检索。XDATA 按数据类型划分信息存储，因此无论何时将信息附加到什么实体，以及何时尝试检索它，都必须知道要存储的信息类型。例如，如果将一个整数值附加到一个实体并尝试将其作为字符串值进行检索，将不会获得所需的结果。

15.2 使用 XRecord 对象

XRecord 对象作为词典进行维护，这意味着它们具有唯一的名称并且可以通过该名称进行访问。它们附加到 Document 对象本身，而不是像 XDATA 那样附加到任何图形对象或表对

象。如果将图形文件保存回 AutoCAD® R12 格式，或者在转换为不支持它们的另一种 CAD 格式时，XRecord 会从绘图中删除。

因为 XRecord 对象附加到文档，所以它们不会被用户随意删除。例如，如果将 XDATA 附加到一个层，并且该层被清除，那么 XDATA 也会被释放。当然也可以将 XDATA 附加到 0 层以防止这种情况发生，但是，XDATA 仍然对可以存储的数据类型和数据大小施加限制。

XRecord 只能被创建、重命名或删除。没有修改它们的直接方法。修改 XRecord 的唯一方法是检索其内容，从 Dictionary 对象中删除 XRecord，然后用新数据重新创建一个新的 XRecord。以下函数演示了如何使用标准 AutoLISP 执行此操作。

```
(defun Xrecord-Rebuild (name dat)
  (Xrecord-Delete name)
  (Xrecord-Add name dat))
(defun Xrecord-Get (name / xlist)
  (if (setq xlist (dictsearch (namedobjdict) name))
      (cdr (assoc 1 xlist))))
(defun Xrecord-Delete (name)
  (dictremove (namedobjdict) name)); remove from dictionary
(defun Xrecord-Add
  (name sdata / xrec xname)
  (setq xrec
    (list
      (cons 0 "XRECORD")
      (cons 100 "AcDbXrecord")
      (cons 1 sdata)
      (cons 62 1)))
  (setq xname (entmakex xrec)); rebuild xrecord
  (dictadd (namedobjdict) name xname); return to dictionary
  (princ))
```

上述形式的问题在于使用 (entmake) 并且在与某些其他 ActiveX 函数混合使用时有时会在 AutoCAD 中引起问题。更合适的形式是 ActiveX 方法，如下例所示。

```
(vl-load-com)
(defun Xrecord-Rebuild (dict name data)
  (Xrecord-Delete dict name)
  (Xrecord-Add dict name data))
(defun Xrecord-Get
  (dict name / acadapp doc dcs odc xrec #typ #dat out)
```



```

(setq acadapp (vlax-get-acad-object))
doc (vla-get-ActiveDocument acadapp)
dcs (vla-get-Dictionaries doc)
(cond
  ( (setq odc (dsx-item dcs dict))
    (cond
      ( (setq xrec (dsx-item odc name))
        (vla-getXrecordData xrec '#typ '#dat)
        (setq #typ (vlax-Safearray->list #typ)
              #dat (vlax-Safearray->list #dat))
        (setq out (mapcar 'vlax-variant-value #dat))
        (vlax-release-object odc)))
      (vlax-release-object dcs)))
  out)
(defun Xrecord-Delete (dict name / dcs odc xr)
  (setq dcs (vla-get-dictionaries active-doc))
  (cond
    ( (setq odc (dsx-item dcs dict))
      (cond
        ( (setq xr (dsx-item odc name))
          (vla-delete xr)
          (vlax-release-object xr)))
        (vlax-release-object odc)))
    (vlax-release-object dcs))
(defun Xrecord-Add
  (dict name data / acadapp doc dicts dict xrec #typ #dat)
  (setq acadapp (vlax-get-acad-object))
  doc (vla-get-ActiveDocument acadapp)
  dicts (vla-get-Dictionaries doc)
  dict (vlax-invoke-method dicts "Add" dict)
  xrec (vla-AddXrecord dict name))
(if (not (listp data)) (setq data (list data))); ensure list!
(vla-setXrecordData xrec
  (List->VariantArray
    (List->IntList data)
    'vlax-vbInteger)
  (List->VariantArray data 'vlax-vbVariant))
(vla-getXrecordData xrec '#typ '#dat)
(setq #typ (vlax-Safearray->list #typ))

```

```
#dat (vlax-Safearray->list #dat))
(mapcar 'vlax-variant-value #dat))
```

`(List->VariantArray)` 和 `(List->IntList)` 这两个函数分别用于定义安全数组的内容和维度。显然，它们的用途远不止于此。`(List->VariantArray)` 的第二个参数必须是以单引号引用的 ActiveX 数据类型声明，例如 `'vlax-vbString`。

```
(defun List->VariantArray (lst datatype / arraySpace sArray)
  (setq arraySpace
    (vlax-make-safearray
      (eval datatype)
      (cons 0 (1- (length lst))))))
  (setq sArray (vlax-Safearray-fill arraySpace lst))
  (vlax-make-variant sArray))
(defun List->IntList (lst / n)
  (setq n 0)
  (mapcar (function (lambda (x) (setq n (1+ n)))) lst))
```

另一个函数 `(DSX-Item)` 用于使用集合的 `Item` 方法抓取（或尝试抓取）对象项。这个函数包括错误捕获以防抓取失败，它返回一个 ActiveX 错误而不是仅仅返回类似 `nil` 的错误。在这种情况下，如果抓取失败，我们会捕获一个错误并返回 `nil`。否则，返回集合中的对象。

```
(defun DSX-Item (collection item / out)
  (if
    (not
      (vl-catch-all-error-p
        (setq out
          (vl-catch-all-apply 'vla-item (list collection item))))))
    out)); return object or nil
```

为了演示如何使用这些东西，我们将在当前绘图中保存一个 `XRecord`，其中包含一些信息，例如当前用户名（假设我们使用的是 Windows® NT/2000/XP^①）和其他一些信息。

```
(setq username (getenv "username")); logged on user ID
      machine (getenv "computername")); NETBIOS computer name
(Xrecord-Rebuild "PERSONAL" "UserData" (list username machine))
```

① 译者注：现在看来，应该换成 Windows® NT/2000/XP/Vista/7/8/10/11...

```
("DSTEIN1234" "W2K-1234")
```

```
Command: (Xrecord-Get "PERSONAL" "UserData")  
("DSTEIN1234" "W2K-1234")
```

```
Command: (Xrecord-Rebuild "PERSONAL" "UserData" "1234")  
("1234")
```

那么，究竟可以使用 **XRecord** 做什么？你想要的都行。它们对于在不直接绑定到任何特定实体或表格的绘图中存储信息非常有用。如果习惯于在 XDATA 中存储信息，你可能知道如果实体或表项被删除，XDATA 就会丢失。当然，你可以将 XDATA 附加到 0 层之类的东西，这样它就永远不会被删除。但是，XDATA 对内容施加了限制，而这些限制可能会通过切换到 **XRecord** 来得到缓解。

AutoCAD 应用程序对象

AcadApplication 对象是 Visual LISP 可以处理的与 ActiveX 相关的所有内容的根。包括从该基础对象继承的集合、属性、方法、事件和派生对象。请记住，ActiveX 对象模型是类的层次结构，允许派生较低级别的类，这些类包含其父类的属性和方法。为了访问 AutoCAD 中的任何内容，必须首先访问 AutoCAD 本身，然后逐步深入到需要的内容。

例如，如果要访问 **Layers** 集合并获取特定层，则必须首先获取 **AcadApplication** 对象，然后是 **ActiveDocument** 对象，然后从该对象获取 **Layers** 集合。开始了解 **AcadApplication** 对象的一种方法是使用 `(vlax-dump-object)` 函数导出对象。此函数采用一个必需的参数（对象）来请求其属性列表，以及一个可选的标志，如果提供（并且非 `nil`）则也请求方法列表。

```
Command: (vlax-dump-object (vlax-get-acad-object) T)

; IAcadApplication: An instance of the AutoCAD application
; Property values:
; ActiveDocument = #<VLA-OBJECT IAcadDocument 0f1e89b0>
; Application (RO) = #<VLA-OBJECT IAcadApplication 01877c20>
; Caption (RO) = "AutoCAD 2011 - UNREGISTERED VERSION - [Drawing1.dwg]"
; Documents (RO) = #<VLA-OBJECT IAcadDocuments 11120b08>
; FullName (RO) = "C:\\Program Files\\Autodesk\\AutoCAD 2011\\acad.exe"
; Height = 726
; HWND (RO) = 983660
; LocaleId (RO) = 1033
; MenuBar (RO) = #<VLA-OBJECT IAcadMenuBar 111172b4>
; MenuGroups (RO) = #<VLA-OBJECT IAcadMenuGroups 0736a890>
; Name (RO) = "AutoCAD"
; Path (RO) = "C:\\Program Files\\Autodesk\\AutoCAD 2011"
; Preferences (RO) = #<VLA-OBJECT IAcadPreferences 110ef5bc>
```

```

; StatusId (R0) = ...Indexed contents not shown...
; VBE (R0) = AutoCAD: Problem in loading VBA
; Version (R0) = "18.1s (LMS Tech)"
; Visible = -1
; Width = 1020
; WindowLeft = 2
; WindowState = 1
; WindowTop = 2
; Methods supported:
; Eval (1)
; GetAcadState ()
; GetInterfaceObject (1)
; ListArx ()
; LoadArx (1)
; LoadDVB (1)
; Quit ()
; RunMacro (1)
; UnloadArx (1)
; UnloadDVB (1)
; Update ()
; ZoomAll ()
; ZoomCenter (2)
; ZoomExtents ()
; ZoomPickWindow ()
; ZoomPrevious ()
; ZoomScaled (2)
; ZoomWindow (2)

```

虽然你可能希望 **AcadApplication** 对象的属性、集合和方法列表更大, 但请记住对象模型是树结构。这意味着大部分复杂性在多个级别上进行了分配, 例如下放到 **Documents** 集合、**Pref-erences** 集合等。上例所示的项目仅适用于 **AcadApplication** 对象, 仅此而已。

要演示可以使用 **AcadApplication** 对象执行的操作, 请加载并运行以下代码示例。这将最小化 AutoCAD 会话窗口, 然后在短暂的暂停后将其最大化。

```

(defun MinMax ( / acadapp)
  (vl-load-com)
  (setq acadapp (vlax-get-acad-object))
  (vla-put-windowstate acadapp acMin)

```

```
(vl-cmdf "DELAY" 1000)
(vla-put-windowstate acadapp acMax)
(vlax-release-object acadapp))
```

当你打算编写启动另一个应用程序的程序时，最小化 AutoCAD 会派上用场。AutoCAD 经常会跳回前面并隐藏其他应用程序窗口，因为它试图从 Windows® 应用程序堆栈中重新获得“焦点”。虽然不总这样，但它还是经常发生的。避免这种情况的一种方法是在启动其他应用程序后隐藏 AutoCAD。这将防止它弹出到其他应用程序窗口的前面。

```
(defun ShowNotepad (filename / acad fn)
  (vl-load-com)
  (cond
    ( (setq fn (findfile filename)); make sure file exists first
      (setq acadapp (vlax-get-acad-object))
      (vla-put-windowstate acadapp acMin)
      (vlax-release-object acadapp)
      (startapp "notepad.exe" fn))
    ( T (princ (strcat "\nFile not found: " filename))))))
```

另一种解决方案是使用第三方函数，例如 DOSLib 函数 `(dos_exewait)`，它会启动另一个应用程序并暂停 AutoCAD，直到另一个应用程序会话终止（关闭）。



AcadApplication 对象的 Path 属性显示 `acad.exe` 在本地计算机上的路径。这可用于在对 AcadPreferences 文件集中的支持文件路径列表进行修改时获取实际安装路径。

由于 AUTODESK® 决定效仿 Microsoft® 并放弃对 VBA 支持，一个有趣的现象是 VBE 对象在 AcadApplication 对象中虽然仍然可见，但它已损坏。例如：

```
Command: (vlax-dump-object (vla-get-vbe (vlax-get-acad-object)) t)
; ERROR: Automation Error. Problem in loading VBA
```


AutoCAD 图元

AutoCAD 图元是图形对象，例如 `Arc`、`Circle` 和 `Line` 对象。在 AutoCAD 中的 ActiveX 编程上下文中，术语图元^①与图形对象同义。所有图元都派生自名为 `AcDbEntity` 的基类，它为所有图元提供某些默认属性和方法。一些默认属性是 `Layer` 和 `Color`，而一些默认方法是 `Copy` 和 `Delete`。

有些属性是所有对象共有的。一些属性对对象组是通用的，但也不是对所有对象通用。某些属性特定于给定的对象类型。要使用 Visual LISP 访问或修改对象属性，请使用以下示例：

```
(vla-get-<propertyname> <object>)
```

其中 `<propertyname>` 可能是 `Color`（颜色）或 `Linetype`（线型）。如果有需要，也可以使用替代形式 `(vlax-get-property <object> <propertyname>)`。在使用 AutoCAD 对象时，这些是可以互换的。

```
(vla-put-<propertyname> <object> <value>)
```

其中 `<propertyname>` 可能是 `color` 或 `linetype`，`<value>` 可以是 `acRed` 或 `"dashed"`。如果有需要，也可以使用替代形式 `(vlax-put-property <object> <propertyname> <value>)`。同样地，在使用 AutoCAD 对象时，这些是可以互换的。

17.1 图元通用属性与方法

属性

^① 译者注：这里及本章标题原作者的用词都是“Entities”，本应译作“实体”，但考虑 AutoCAD 的一种图元 `SOLID` 也会被翻译成“实体”，为了不使二者混淆，本章使用“图元”作为“Entities”的翻译。而且，作者也说了“AutoCAD Entities are graphical objects ...”

属性	说明	数据类型	DXF
Application (RO)	AutoCAD 会话	对象（指针）	
Color	图元颜色	整数/枚举	62 ^a
Document (RO)	父文档	对象（指针）	
EntityTransparency	透明度设置	字符串	
Handle	图元句柄	字符串	5
HasExtensionDictionary (RO)	是否带有扩展数据	布尔型	
Hyperlinks (RO)	网络超链接	对象（集合）	
Layer	图层	字符串	8
Linetype	线型	字符串	6 ^a
LinetypeScale	线型比例	实数/双精度	48
Lineweight	线宽	整数/枚举	370
Material	材料分配	字符串	
ObjectID (RO)	对象标识值	整数	
ObjectName (RO)	图元类型名称	字符串	100 ^b
OwnerID (RO)	父对象	整数	
Normal	拉伸方向向量	数组	210
PlotStyleName	打印样式名称	字符串	
Thickness	图元厚度	实数/双精度	39 ^a
TrueColor	RGB/Pantone 色值	acCmColor	
Visible	可见性	布尔型	60
Layout-Name	布局名称	(N/A)	410
(N/A)	DXF 图元名称	整数	0
(N/A)	DXF 空间	整数	67
(N/A)	DXF XRECORD 组	字符串	102 ^c
(N/A)	DXF 材料对象硬指针	字符串	347

^a 短暂的^b 存在多项^c 成对出现

注：取决于在活动绘图中使用基于颜色还是基于名称的打印样式

方法

方法	说明	参数数量
ArrayPolar	极轴阵列	3
ArrayRectangular	矩形阵列	6
Copy	复制图元	0
Delete	删除图元	2
GetBoundingBox	获取矩形包围框对角线点坐标	0
GetExtensionDictionary	查询对象是否附加了 XRECORD	3
GetXData	查询对象附加的 XDATA	1
Highlight	高亮显示对象	1
IntersectWith	与其他对象/图元求交点	2
Mirror	对于二维轴镜像	2
Mirror3D	对于二维轴镜像	3
Move	移动与重定位	2
Offset	创建偏移后的副本	1
Rotate	以二维点进行二维旋转	2
Rotate3D	以二维点进行二维旋转	3
ScaleEntity	缩放几何形状	2
SetXData	向对象附加 XDATA	2
TransformBy	进行矩阵变换	1
Update	更新图元属性列表	0

17.2 各种图元的属性与方法

圆弧（Arc）

圆弧（Arc）的属性说明如下：

属性	说明	数据类型
ArcLength (RO)	圆弧的周长	Double
Area (RO)	圆弧的封闭区域面积	Double
Center	圆弧的中心点	Array (Double)
EndAngle	圆弧的终止角度 (弧度制)	Double
EndPoint	圆弧的终止点	Array (Double)
Radius	圆弧的半径值	Double
StartAngle	圆弧的起始角度 (弧度制)	Double
StartPoint	圆弧的起始点	Array (Double)
TotalAngle (RO)	圆弧的总角度 (弧度制)	Double



不能更改圆弧或椭圆的 **StartPoint**。要编辑圆弧，请使用 **EndAngle** 和 **Radius** 属性。要编辑椭圆，请使用 **EndAngle**、**MajorAxis** 和 **RadiusRatio** 属性。
Area 属性实际计算的是弓形的面积，就好像它被从起点到终点的矢量封闭一样。这与圆弧段的面积不同，圆弧段的面积包括从中心点到周边的面积，作为整个圆形面积的一部分。

块参照/块插入 (BlockReference/INSERT)

块参照/块插入 (BlockReference/INSERT) 的属性说明如下：

属性	说明	数据类型
EffectiveName (RO)	原始图块名	String
InsertionPoint	图块的插入点	Array (Double)
InsUnits (RO)	与图块一起保存的插入单位	String
InsUnitsFactor (RO)	在图块单位和图形单位间的转换比例因子	Double
IsDynamicBlock (RO)	图块是否为动态块	Boolean
Name	图块对象的名称	String

属性	说明	数据类型
Rotation	图块的旋转角度	Double
XEffectiveScaleFactor	图块的有效 X 方向比例因子	Double
XScaleFactor	图块或者外部参照的 X 方向缩放比例	Double
YEffectiveScaleFactor	图块的有效 Y 方向比例因子	Double
YScaleFactor	图块或者外部参照的 Y 方向缩放比例	Double
ZEffectiveScaleFactor	图块的有效 Z 方向比例因子	Double
ZScaleFactor	图块或者外部参照的 Z 方向缩放比例	Double

块参照/块插入 (**BlockReference/INSERT**) 的方法说明如下:

方法	说明	参数数量
ConvertToAnonymousBlock	转换动态块为常规的匿名块	0
ConvertToStaticBlock	转换动态块为常规的命名块	1
Explode	分解图块	0
GetAttributes	获取块参照中的属性 (Attribute)	0
GetConstantAttributes	获取块参照中的固定属性 (Attribute)	0
GetDynamicBlockProperties	获取动态块的属性	0
ResetBlock	重置动态块的默认状态	0

圆 (**Circle**)

圆 (**Circle**) 的属性说明如下:

属性	说明	数据类型
Area	圆封闭区域的面积	Double
Center	圆心	Array (Double)

属性	说明	数据类型
Circumference	圆的周长	Double
Diameter	圆的直径值	Double
Radius	圆的半径值	Double

标注 (DIMENSION)：角度标注 (DimAngular)

角度标注 (DimAngular) 的属性说明如下：

属性	说明	数据类型
AngleFormat	角度标注的单位形式	Integer/Enum
Arrowhead1Block	角度标注尺寸线第一端自定义箭头的块名	String
Arrowhead1Type	角度标注第一个标注箭头的类型	Integer/Enum
Arrowhead2Block	角度标注尺寸线第二端自定义箭头的块名	String
Arrowhead2Type	角度标注第二个标注箭头的类型	Integer/Enum
ArrowheadSize	角度标注标注箭头的尺寸	Double
DecimalSeparator	角度标注文字的小数点分隔符	String
DimConstrDesc	角度标注约束的说明	String
DimConstrExpression	角度标注约束的表达式或值	String
DimConstrForm	角度标注约束的类型为动态或注释性	Boolean
DimConstrName	角度标注约束的名称	String
DimConstrReference	角度标注是否为参照标注	Boolean
DimConstrValue	角度标注约束的值	String
DimensionLineColor	角度标注尺寸线颜色	Integer/Enum
DimensionLinetype	角度标注尺寸线线型	String
DimensionLineWeight	角度标注尺寸线的线宽	Integer/Enum
DimLine1Suppress	是否显示第一条尺寸线	Boolean

属性	说明	数据类型
<code>DimLine2Suppress</code>	是否显示第二条尺寸线	Boolean
<code>DimLineInside</code>	是否只在尺寸线内侧绘制尺寸线	Boolean
<code>DimTxtDirection</code>	角度标注文字阅读方向是否为逆序（从右至左）	Boolean
<code>ExtensionLineColor</code>	角度标注尺寸界线的颜色	Integer/Enum
<code>ExtensionLineExtend</code>	角度标注尺寸界线超出尺寸线的距离	Double
<code>ExtensionLineOffset</code>	角度标注尺寸界线偏移起点的距离	Double
<code>ExtensionLineWeight</code>	角度标注尺寸界线的线宽	Integer/Enum
<code>ExtLine1EndPoint</code>	角度标注第一条尺寸界线的终点	Array (Double)
<code>ExtLine1Linetype</code>	角度标注第一条尺寸界线的线型	String
<code>ExtLine1StartPoint</code>	角度标注第一条尺寸界线的起点	Array (Double)
<code>ExtLine1Suppress</code>	是否隐藏第一条尺寸界线	Boolean
<code>ExtLine2EndPoint</code>	角度标注第二条尺寸界线的终点	Array (Double)
<code>ExtLine2Linetype</code>	角度标注第二条尺寸界线的线型	String
<code>ExtLine2StartPoint</code>	角度标注第二条尺寸界线的起点	Array (Double)
<code>ExtLine2Suppress</code>	是否隐藏第二条尺寸界线	Boolean
<code>ExtLineFixedLen</code>	角度标注尺寸界线的固定长度	Double
<code>ExtLineFixedLenSuppress</code>	角度标注尺寸界线是否采用固定长度	Boolean
<code>Fit</code>	角度标注文字与箭头的适应模式	Integer/Enum
<code>ForceLineInside</code>	是否强制在尺寸界线之间绘制尺寸线	Boolean
<code>HorizontalTextPosition</code>	角度标注文字的水平对齐方式	Integer/Enum
<code>Measurement</code>	角度标注的实测值	Double
<code>StyleName</code>	角度标注样式名称	String
<code>SuppressLeadingZeros</code>	角度标注文字是否消去标注的前导的 0	Boolean
<code>SuppressTrailingZeros</code>	角度标注文字是否消去标注的后续的 0	Boolean

属性	说明	数据类型
TextColor	角度标注文字的颜色	Integer/Enum
TextFill	角度标注文字是否进行背景填充	Boolean
TextFillColor	角度标注文字背景填充颜色	ACAD_COLOR
TextGap	在打断尺寸线、放入标注文字时，标注文字与尺寸线之间的距离	Double
TextHeight	角度标注的文字高度	Double
TextInside	是否强制在尺寸界线之间绘制文字	Boolean
TextInsideAlign	是否强制尺寸界线内侧标注文字水平对齐	Boolean
TextMovement	标注文字移动后的绘制方式	Integer/Enum
TextOutsideAlign	是否强制尺寸界线外侧标注文字水平对齐	Boolean
TextOverride	角度标注文字字符串	String
TextPosition	角度标注文字位置	Array (Double)
TextPrecision	角度标注文字显示精度的小数位数	Integer
TextPrefix	角度标注文字前缀	String
TextRotation	角度标注文字的旋转角度	Double
TextStyle	角度标注文字的文字样式	String
TextSuffix	角度标注文字后缀	String
ToleranceDisplay	角度标注文字的标注公差显示模式	Integer/Enum
ToleranceHeightScale	公差文字高度相对于标注文字高度的比例因子	Double
ToleranceJustification	公差文字相对于标注文字的垂直对正	Integer/Enum
ToleranceLowerLimit	角度标注公差的极限下偏差	Double
TolerancePrecision	角度标注主单位公差值精度的小数位数	Integer/Enum
ToleranceSuppressLeadingZeros	是否消去主单位公差值的前导的 0	Boolean
ToleranceSuppressTrailingZeros	是否消去主单位公差值的后续的 0	Boolean
ToleranceUpperLimit	角度标注公差的极限上偏差	Double

属性	说明	数据类型
<code>VerticalTextPosition</code>	角度标注文字相对于尺寸线的垂直位置	<code>Integer/Enum</code>

标注 (**DIMENSION**): 直径标注 (**DimDiametric**)

直径标注 (**DimDiametric**) 的属性说明如下:

属性	说明	数据类型
<code>AltRoundDistance</code>	直径标注换算单位的舍入值	<code>Double</code>
<code>AltSuppressLeadingZeros</code>	是否消除直径标注换算单位前导 0	<code>Boolean</code>
<code>AltSuppressTrailingZeros</code>	是否消除直径标注换算单位后续 0	<code>Boolean</code>
<code>AltSuppressZeroFeet</code>	是否消除直径标注换算单位中的 0 英尺	<code>Boolean</code>
<code>AltSuppressZeroInches</code>	是否消除直径标注换算单位中的 0 英寸	<code>Boolean</code>
<code>AltTextPrefix</code>	直径标注换算单位标注文字的前缀	<code>String</code>
<code>AltTextSuffix</code>	直径标注换算单位标注文字的后缀	<code>String</code>
<code>AltTolerancePrecision</code>	直径标注换算标注公差值的小数位数	<code>Integer/Enum</code>
<code>AltToleranceSuppressLeadingZeros</code>	是否消除直径标注换算单位公差值的前导 0	<code>Boolean</code>
<code>AltToleranceSuppressTrailingZeros</code>	是否消除直径标注换算单位公差值的后续 0	<code>Boolean</code>
<code>AltToleranceSuppressZeroFeet</code>	是否消除直径标注换算单位公差值中的 0 英尺	<code>Boolean</code>
<code>AltToleranceSuppressZeroInches</code>	是否消除直径标注换算单位公差值中的 0 英寸	<code>Boolean</code>
<code>AltUnits</code>	是否开启直径标注的换算单位	<code>Boolean</code>
<code>AltUnitsFormat</code>	直径标注换算单位的格式	<code>Integer/Enum</code>
<code>AltUnitsPrecision</code>	直径标注换算单位的小数位数	<code>Integer</code>
<code>AltUnitsScale</code>	直径标注换算单位的比例因子	<code>Double</code>

属性	说明	数据类型
Arrowhead1Block	直径标注尺寸线第一端自定义箭头的块名	String
Arrowhead1Type	直径标注第一个标注箭头的类型	Integer/Enum
Arrowhead2Block	直径标注尺寸线第二端自定义箭头的块名	String
Arrowhead2Type	直径标注第二个标注箭头的类型	Integer/Enum
ArrowheadSize	直径标注标注箭头的尺寸	Double
CenterMarkSize	直径标注圆心标记的尺寸	Double
CenterType	直径标注圆心标记的类型	Integer/Enum
DecimalSeparator	直径标注文字的小数点分隔符	String
DimConstrDesc	直径标注约束的说明	String
DimConstrExpression	直径标注约束的表达式或值	String
DimConstrForm	直径标注约束的类型为动态或注释性	Boolean
DimConstrName	直径标注约束的名称	String
DimConstrReference	直径标注是否为参照标注	Boolean
DimConstrValue	直径标注约束的值	String
DimensionLineColor	直径标注尺寸线颜色	Integer/Enum
DimensionLinetype	直径标注尺寸线线型	String
DimensionLineWeight	直径标注尺寸线的线宽	Integer/Enum
DimLine1Suppress	是否显示第一条尺寸线	Boolean
DimLine2Suppress	是否显示第二条尺寸线	Boolean
DimTxtDirection	直径标注文字阅读方向是否为逆序（从右至左）	Boolean
Fit	直径标注文字与箭头的适应模式	Integer/Enum
ForceLineInside	是否强制在尺寸界线之间绘制尺寸线	Boolean
FractionFormat	直径标注中分数值的格式	Integer/Enum
LeaderLength	直径标注上引线的长度	Double

属性	说明	数据类型
<code>LinearScaleFactor</code>	直径标注测量值的全局比例因子	Double
<code>Measurement</code>	直径标注的实测值	Double
<code>PrimaryUnitsPrecision</code>	直径标注主要单位显示的小数位数	Integer/Enum
<code>RoundDistance</code>	直径标注的舍入值	Double
<code>StyleName</code>	直径标注样式名称	String
<code>SuppressLeadingZeros</code>	直径标注文字是否消去标注的前导的 0	Boolean
<code>SuppressTrailingZeros</code>	直径标注文字是否消去标注的后续的 0	Boolean
<code>SuppressZeroFeet</code>	是否消除直径标注中的 0 英尺	Boolean
<code>SuppressZeroInches</code>	是否消除直径标注中的 0 英寸	Boolean
<code>TextColor</code>	直径标注文字的颜色	Integer/Enum
<code>TextFill</code>	直径标注文字是否进行背景填充	Boolean
<code>TextFillColor</code>	直径标注文字背景填充颜色	ACAD_COLOR
<code>TextGap</code>	在打断尺寸线、放入标注文字时，标注文字与尺寸线之间的距离	Double
<code>TextHeight</code>	直径标注的文字高度	Double
<code>TextInside</code>	是否强制在尺寸界线之间绘制文字	Boolean
<code>TextInsideAlign</code>	是否强制尺寸界线内侧标注文字水平对齐	Boolean
<code>TextMovement</code>	标注文字移动后的绘制方式	Integer/Enum
<code>TextOutsideAlign</code>	是否强制尺寸界线外侧标注文字水平对齐	Boolean
<code>TextOverride</code>	直径标注文字字符串	String
<code>TextPosition</code>	直径标注文字位置	Array (Double)
<code>TextPrecision</code>	直径标注文字显示精度的小数位数	Integer
<code>TextPrefix</code>	直径标注文字前缀	String
<code>TextRotation</code>	直径标注文字的旋转角度	Double
<code>TextStyle</code>	直径标注文字的文字样式	String

属性	说明	数据类型
TextSuffix	直径标注文字后缀	String
ToleranceDisplay	直径标注文字的标注公差显示模式	Integer/Enum
ToleranceHeightScale	公差文字高度相对于标注文字高度的比例因子	Double
ToleranceJustification	公差文字相对于标注文字的垂直对正	Integer/Enum
ToleranceLowerLimit	直径标注公差的极限下偏差	Double
TolerancePrecision	直径标注主单位公差值精度的小数位数	Integer/Enum
ToleranceSuppressLeadingZeros	是否消去主单位公差值的前导的 0	Boolean
ToleranceSuppressTrailingZeros	是否消去主单位公差值的后续的 0	Boolean
ToleranceSuppressZeroFeet	是否消除直径标注公差值中的 0 英尺	Boolean
ToleranceSuppressZeroInches	是否消除直径标注公差值中的 0 英寸	Boolean
ToleranceUpperLimit	直径标注公差的极限上偏差	Double
UnitsFormat	直径标注的单位格式	Integer/Enum
VerticalTextPosition	直径标注文字相对于尺寸线的垂直位置	Integer/Enum

标注 (DIMENSION): 线性标注 (DimAligned)

线性标注 (DimAligned) 的属性说明如下:

属性	说明	数据类型
AltRoundDistance	线性标注换算单位的舍入值	Double
AltSubUnitsFactor	线性标注转换为子单位时换算单位的比例因子	Double
AltSubUnitsSuffix	线性标注转换为子单位时换算单位的后缀	String
AltSuppressLeadingZeros	是否抑制线性标注换算单位前导 0	Boolean
AltSuppressTrailingZeros	是否抑制线性标注换算单位后续 0	Boolean

属性	说明	数据类型
<code>AltSuppressZeroFeet</code>	是否抑制线性标注换算单位中的 0 英尺	Boolean
<code>AltSuppressZeroInches</code>	是否抑制线性标注换算单位中的 0 英寸	Boolean
<code>AltTextPrefix</code>	线性标注换算单位标注文字的前缀	String
<code>AltTextSuffix</code>	线性标注换算单位标注文字的后缀	String
<code>AltTolerancePrecision</code>	线性标注换算标注公差值的小数位数	Integer/Enum
<code>AltToleranceSuppressLeadingZeros</code>	是否消除线性标注换算单位公差值的前导 0	Boolean
<code>AltToleranceSuppressTrailingZeros</code>	是否消除线性标注换算单位公差值的后续 0	Boolean
<code>AltToleranceSuppressZeroFeet</code>	是否消除线性标注换算单位公差值中的 0 英尺	Boolean
<code>AltToleranceSuppressZeroInches</code>	是否消除线性标注换算单位公差值中的 0 英寸	Boolean
<code>AltUnits</code>	是否开启线性标注的换算单位	Boolean
<code>AltUnitsFormat</code>	线性标注换算单位的格式	Integer/Enum
<code>AltUnitsPrecision</code>	线性标注换算单位的小数位数	Integer
<code>AltUnitsScale</code>	线性标注换算单位的比例因子	Double
<code>Arrowhead1Block</code>	线性标注尺寸线第一端自定义箭头的块名	String
<code>Arrowhead1Type</code>	线性标注第一个标注箭头的类型	Integer/Enum
<code>Arrowhead2Block</code>	线性标注尺寸线第二端自定义箭头的块名	String
<code>Arrowhead2Type</code>	线性标注第二个标注箭头的类型	Integer/Enum
<code>ArrowheadSize</code>	线性标注标注箭头的尺寸	Double
<code>DecimalSeparator</code>	线性标注文字的小数点分隔符	String
<code>DimensionLineColor</code>	线性标注尺寸线颜色	Integer/Enum
<code>DimensionLineExtend</code>	绘制倾斜笔划而不是绘制箭头时尺寸线超出尺寸界线的距离	Double
<code>DimensionLinetype</code>	线性标注尺寸线线型	String

属性	说明	数据类型
DimensionLineWeight	线性标注尺寸线的线宽	Integer/Enum
DimLine1Suppress	是否显示第一条尺寸线	Boolean
DimLine2Suppress	是否显示第二条尺寸线	Boolean
DimLineInside	是否只在尺寸线内侧绘制尺寸线	Boolean
DimTxtDirection	线性标注文字阅读方向是否为逆序（从右至左）	Boolean
ExtensionLineColor	线性标注尺寸界线的颜色	Integer/Enum
ExtensionLineExtend	线性标注尺寸界线超出尺寸线的距离	Double
ExtensionLineOffset	线性标注尺寸界线偏移起点的距离	Double
ExtensionLineWeight	线性标注尺寸界线的线宽	Integer/Enum
ExtLine1Suppress	是否隐藏第一条尺寸界线	Boolean
ExtLine2Suppress	是否隐藏第二条尺寸界线	Boolean
ExtLineFixedLen	线性标注尺寸界线的固定长度	Double
ExtLineFixedLenSuppress	线性标注尺寸界线是否采用固定长度	Boolean
Fit	线性标注文字与箭头的适应模式	Integer/Enum
ForceLineInside	是否强制在尺寸界线之间绘制尺寸线	Boolean
FractionFormat	线性标注中分数值的格式	Integer/Enum
HorizontalTextPosition	线性标注文字的水平对齐方式	Integer/Enum
LinearScaleFactor	线性标注测量值的全局比例因子	Double
Measurement	线性标注的实测值	Double
PrimaryUnitsPrecision	线性标注主要单位显示的小数位数	Integer/Enum
Rotation	线性标注对象的旋转角度（弧度制）	Double
RoundDistance	线性标注的舍入值	Double
ScaleFactor	线性标注的比例因子	Double
StyleName	线性标注样式名称	String

属性	说明	数据类型
<code>SubUnitsFactor</code>	线性标注单位转化为子单位时的比例因子	Double
<code>SubUnitsSuffix</code>	线性标注单位转化为子单位时的后缀	String
<code>SuppressLeadingZeros</code>	线性标注文字是否消去标注的前导的 0	Boolean
<code>SuppressTrailingZeros</code>	线性标注文字是否消去标注的后续的 0	Boolean
<code>SuppressZeroFeet</code>	是否消除线性标注中的 0 英尺	Boolean
<code>SuppressZeroInches</code>	是否消除线性标注中的 0 英寸	Boolean
<code>TextColor</code>	线性标注文字的颜色	Integer/Enum
<code>TextFill</code>	线性标注文字是否进行背景填充	Boolean
<code>TextFillColor</code>	线性标注文字背景填充颜色	ACAD_COLOR
<code>TextGap</code>	在打断尺寸线、放入标注文字时，标注文字与尺寸线之间的距离	Double
<code>TextHeight</code>	线性标注的文字高度	Double
<code>TextInside</code>	是否强制在尺寸界线之间绘制文字	Boolean
<code>TextInsideAlign</code>	是否强制尺寸界线内侧标注文字水平对齐	Boolean
<code>TextMovement</code>	标注文字移动后的绘制方式	Integer/Enum
<code>TextOutsideAlign</code>	是否强制尺寸界线外侧标注文字水平对齐	Boolean
<code>TextOverride</code>	线性标注文字字符串	String
<code>TextPosition</code>	线性标注文字位置	Array (Double)
<code>TextPrecision</code>	线性标注文字显示精度的小数位数	Integer
<code>TextPrefix</code>	线性标注文字前缀	String
<code>TextRotation</code>	线性标注文字的旋转角度	Double
<code>TextStyle</code>	线性标注文字的文字样式	String
<code>TextSuffix</code>	线性标注文字后缀	String
<code>ToleranceDisplay</code>	线性标注文字的标注公差显示模式	Integer/Enum
<code>ToleranceHeightScale</code>	公差文字高度相对于标注文字高度的比例因子	Double

属性	说明	数据类型
<code>ToleranceJustification</code>	公差文字相对于标注文字的垂直对正	Integer/Enum
<code>ToleranceLowerLimit</code>	线性标注公差的极限下偏差	Double
<code>TolerancePrecision</code>	线性标注主单位公差值精度的小数位数	Integer/Enum
<code>ToleranceSuppressLeadingZeros</code>	是否消去主单位公差值的前导的 0	Boolean
<code>ToleranceSuppressTrailingZeros</code>	是否消去主单位公差值的后续的 0	Boolean
<code>ToleranceSuppressZeroFeet</code>	是否消除线性标注公差值中的 0 英尺	Boolean
<code>ToleranceSuppressZeroInches</code>	是否消除线性标注公差值中的 0 英寸	Boolean
<code>ToleranceUpperLimit</code>	线性标注公差的极限上偏差	Double
<code>UnitsFormat</code>	线性标注的单位格式	Integer/Enum
<code>VerticalTextPosition</code>	线性标注文字相对于尺寸线的垂直位置	Integer/Enum



延伸线和尺寸线的控制点不通过 ActiveX 公开。要获得这些控制点，必须使用 DXF 组码 10、11、13 和 14。说明如下：

- 13 —第一条延长线的起点（第一控制点）；
- 14 —第二条延长线的起点（第二控制点）；
- 11 —`MTEXT` 尺寸标签的中心点；
- 10 —第二条尺寸线的箭头点。

椭圆（`Ellipse`）

椭圆（`Ellipse`）的属性说明如下：

属性	说明	数据类型
<code>Area (RO)</code>	椭圆封闭区域的面积	Double
<code>Center</code>	椭圆的中心点	Array (Double)
<code>EndAngle</code>	椭圆的终止角度（弧度）	Double
<code>EndParameter</code>	椭圆的终止参数	Double

属性	说明	数据类型
EndPoint (R0)	椭圆的终点	Array (Double)
MajorAxis	椭圆的长轴方向	Array (Double)
MajorRadius	椭圆的长轴半径	Double
MinorAxis	椭圆的短轴方向	Array (Double)
MinorRadius	椭圆的短轴半径	Double
RadiusRatio	椭圆的长短轴半径比	Double
StartAngle	椭圆的起始角度（弧度）	Double
StartParameter	椭圆的起始参数	Double
StartPoint (R0)	椭圆的起点	Array (Double)



如果椭圆是闭合的, **StartAngle** 值为 0, **EndAngle** 值为 2π 。
不能更改椭圆的 **StartPoint** 或 **Endpoint** 属性。

填充（Hatch）

填充（Hatch）的属性说明如下：

属性	说明	数据类型
Area (R0)	填充封闭区域的面积	Double
AssociativeHatch (R0)	图案填充是否关联	Boolean
BackgroundColor	背景填充颜色	Object
GradientAngle	渐变填充的角度	Double
GradientCentered	是否按中心渐变填充	Boolean
GradientColor1	渐变填充的起始颜色	Object
GradientColor2	渐变填充的结束颜色	Object
GradientName	渐变填充的图案名称	String

属性	说明	数据类型
HatchObjectType	图案填充的类型	Integer/Enum
HatchStyle	图案填充的样式	Integer/Enum
ISOPenWidth	图案填充的 WCS 笔宽	Integer/Enum
NumberOfLoops (RO)	图案填充的边界数量	Integer
Origin	指定填充在 WCS 中的 UCS 原点	Array
PatternAngle	填充图案角度	Double
PatternDouble	用户定义图案填充是否为双向图案填充	Boolean
PatternName (RO)	填充图案名称	String
PatternScale	填充图案比例	Double
PatternSpace	用户定义填充图案的间距	Double
PatternType (RO)	用于图案填充的图案类型	Integer/Enum

填充（Hatch）的方法说明如下：

方法	说明	参数数量
AppendInnerLoop	向图案填充中附加内边界	1
AppendOuterLoop	向图案填充中附加外边界	1
Evaluate	计算给定的图案填充	0
GetLoopAt	获取给定索引位置的图案填充边界	2
InsertLoopAt	按给定索引在图案填充中插入边界	3
SetPattern	设置图案填充的图案类型和名称	2

引线（Leader）

引线（Leader）的属性说明如下：

属性	说明	数据类型
ArrowheadBlock	引线上作为自定义箭头的图块	String
ArrowheadSize	引线箭头的尺寸	Double
ArrowheadType	引线箭头的类型	Integer/Enum
Coordinate	引线中单个顶点的坐标	Array (Double)
Coordinates	引线中每个顶点的坐标	Array
DimensionLineColor	引线尺寸线颜色	Integer/Enum
DimensionLineWeight	引线尺寸线线宽	Integer/Enum
ScaleFactor	引线的比例因子	Double
StyleName	引线的标注样式名	String
TextGap	引线标注文字与尺寸线之间的距离	Double
Type	引线对象的类型	Integer/Enum
VerticalTextPosition	引线标注文字相对于尺寸线的垂直位置	Integer/Enum

引线 (Leader) 的方法说明如下:

方法	说明	参数数量
Evaluate	计算给定的引线	0

直线 (Line)

直线 (Line) 的属性说明如下:

属性	说明	数据类型
Angle(R0)	直线的角度	Double
Delta(R0)	直线 X、Y、Z 的增量	Array (Double)
EndPoint	直线的终点	Array (Double)
Length(R0)	直线的长度	Double

属性	说明	数据类型
StartPoint	直线的起点	Array (Double)

轻量化多段线（LWPolyline）

轻量化多段线（LWPolyline）的属性说明如下：

属性	说明	数据类型
Area (RO)	轻量化多段线封闭区域的面积	Double
Closed	轻量化多段线闭合标志	Boolean
ConstantWidth	轻量化多段线的全局宽度	Double
Coordinate	轻量化多段线单个顶点的坐标	Array
Coordinates	轻量化多段线每个顶点的坐标	Array (Double)
Elevation	轻量化多段线的标高	Double
LinetypeGeneration	轻量化多段线的线型生成标志	Boolean

轻量化多段线（LWPolyline）的方法说明如下：

方法	说明	参数数量
AddVertex	向轻量化多段线中添加顶点	2
Explode	分解轻量化多段线图元	0
GetBulge	获取轻量化多段线上给定索引位置的凸度值	1
GetWidth	获取轻量化多段线的起始和终止宽度	3
SetBulge	设置轻量化多段线在给定索引位置的凸度	2
SetWidth	设置轻量化多段线上给定段索引的起始和终止宽度	3

多线（MLine）

多线（MLine）的属性说明如下：

属性	说明	数据类型
Coordinates	多线每个顶点的坐标	Array (Double)
Justification	多线的对齐方式	Integer/Enum
MlineScale	多线的缩放比例因子	Double
StyleName(R0)	多线的样式名称	String



MLine 的以下属性均已弃用: **Angle**、**EndPoint**、**StartPoint**、**Delta** 和 **Length**。

多行文字 (MText)

多行文字 (MText) 的属性说明如下:

属性	说明	数据类型
AttachmentPoint	多行文字的附着基点 (相对文字边界的具体位置)	Integer/Enum
BackgroundFill	多行文字是否存在背景填充	Boolean
DrawingDirection	多行文字的排版方向	Integer/Enum
LineSpacingDistance	多行文字的行间距	Double
LineSpacingFactor	多行文字的行间距比例因子	Double
LineSpacingStyle	多行文字的行间距样式	Integer/Enum
Rotation	多行文字的旋转角度 (弧度制)	Double
StyleName	多行文字的文字样式	String
TextString	多行文字的文字字符串	String
Width	多行文字对象的宽度	Double

多行文字 (MText) 的方法说明如下:

方法	说明	参数数量
FieldCode	获取含有字段代码的字符串	0



行间距是使用基本 **Height** 属性值计算的。当在给定字符串中使用不同的高度时，属性 **LineSpacingFactor** 仅应用于基本 **Height** 属性值。

分数堆叠行为由 **TSTACKSIZE** 和 **TSTACKALIGN** 系统变量控制。**TSTACKSIZE** 是一个整数，表示文字高度百分比。值 70 表示 $0.7 \times \text{Height}$ 。**TSTACKALIGN** 控制堆叠的应用方式。0 — 无堆叠，1 — 对角线（例如 $\frac{1}{4}$ ），2 — 垂直（例如 $\frac{1}{4}$ ）。

格式化控制代码说明如下：

- **\\P** 表示换行/回车；
- **\\S** 表示分数堆叠开始分组；
- **\\A** 表示相对高度变化（使用相对于基本 **Height** 属性的相对因子）。

点（Point）

点（Point）的属性说明如下：

属性	说明	数据类型
Coordinates	点的坐标	Array (Double)



Point 图元的显示由系统变量 **PDMODE** 控制。设置为 1 会隐藏它们。设置为 0 或其他正数将以各种符号类型显示它们。系统变量 **PDSIZE** 对应缩放因子，控制 **Point** 图元符号的相对大小。

多段线（Polyline）

多段线（Polyline）的属性说明如下：

属性	说明	数据类型
Area(R0)	多段线封闭区域的面积	Double

属性	说明	数据类型
Closed	多段线的闭合标志	Boolean
ConstantWidth	多段线的全局宽度	Double
Coordinate	多段线单个顶点的坐标	Array (Double)
Coordinates	多段线每个顶点的坐标	Array (Double)
Elevation	多段线的标高	Double
Length(R0)	多段线的总长	Double
LinetypeGeneration	多段线的线型生成标志	Boolean
Type	多段线的类型	Integer/Enum

多段线（Polyline）的方法说明如下：

方法	说明	参数数量
AddVertex	向多段线中添加顶点	2
Explode	分解多段线图元	0
GetBulge ^a	获取多段线上给定索引位置的凸度值	1
GetWidth ^b	获取多段线的起始和终止宽度	3
SetBulge ^a	设置多段线在给定索引位置的凸度	2
SetWidth ^b	设置多段线上给定段索引的起始和终止宽度	3

^a 当 Polyline 的 Type 属性 不为 acSimplePoly 时将会失败
^b 当 Polyline 的 Type 属性 为 acCubicSplinePoly 或 acQuadSplinePoly 时将会失败

光栅图像（RasterImage）

光栅图像（RasterImage）的属性说明如下：

属性	说明	数据类型
Brightness	光栅图像的亮度值	Integer

属性	说明	数据类型
ClippingEnabled	光栅图像是否可以裁剪边界	Boolean
Contrast	光栅图像的对比度值	Integer
Fade	光栅图像的淡化值	Integer
Height(R0)	光栅图像对象的高度（像素）	Double
ImageFile	光栅图像文件的完整路径和文件名	String
ImageHeight	光栅图像的高度（图形单位）	Double
ImageVisibility	光栅图像的可见性	Boolean
ImageWidth	光栅图像的高度（图形单位）	Double
Name	光栅图像的名称	String
Origin	光栅图像在 WCS 坐标系下的插入基点	Array (Double)
ShowRotation	光栅图像是否按旋转角度值旋转显示	Boolean
Width(R0)	光栅图像对象的宽度（像素）	Double

光栅图像（**RasterImage**）的方法说明如下：

方法	说明	参数数量
ClipBoundary	指定光栅图像的剪裁边界	1

射线（**Ray**）

射线（**Ray**）的属性说明如下：

属性	说明	数据类型
BasePoint	射线经过的基点	Array (Double)
DirectionVector	射线方向的矢量	Array (Double)
SecondPoint	射线经过的第二点	Array (Double)

实体 (Solid)

实体 (Solid) 的属性说明如下:

属性	说明	数据类型
Coordinate	实体单个顶点的坐标	Array (Double)
Coordinates	实体每个顶点的坐标	Array (Double)

样条曲线 (Spline)

样条曲线 (Spline) 的属性说明如下:

属性	说明	数据类型
Area (R0)	样条曲线封闭区域的面积	Double
Closed (R0)	样条曲线的闭合标志	Boolean
Closed2	样条曲线的闭合标志2	Boolean
ControlPoints	样条曲线的控制点	Array (Double)
Degree (R0)	样条曲线多项式表达式的次数	Integer
Degree2	样条曲线多项式表达式的次数2	Integer
EndTangent	样条曲线终点切向向量	Array (Double)
FitPoints	样条曲线的拟合点	Array (Double)
FitTolerance	样条曲线的拟合公差	Double
IsPeriodic (R0)	样条曲线是否有周期性	Boolean
IsPlanar (R0)	样条曲线是否为平面曲线	Boolean
IsRational (R0)	样条曲线是否为有理曲线	Boolean
KnotParameterization	样条曲线的参数化类型	Integer/Enum
Knots	样条曲线的节点向量	Array (Double)
NumberOfControlPoints (R0)	样条曲线的控制点数量	Integer

属性	说明	数据类型
NumberOfFitPoints (RO)	样条曲线的拟合点数量	Integer
SplineFrame	样条曲线控制点的显示	Integer/Enum
SplineMethod	样条曲线生成方法	Integer/Enum
StartTangent	样条曲线起点切向向量	Array (Double)
Weights	样条曲线的权重向量	Array (Double)

样条曲线（Spline）的方法说明如下：

方法	说明	参数数量
AddFitPoint	在样条曲线给定索引处添加拟合点	2
DeleteFitPoint	删除给定索引处样条曲线的拟合点	1
ElevateOrder	将样条曲线的阶数提升到给定的阶数	2
GetControlPoint	获取给定索引处样条曲线的控制点	1
GetFitPoint	获取给定索引处样条曲线的拟合点	1
GetWeight	获取样条曲线给定索引处控制点的权重	1
PurgeFitData	清除样条曲线的拟合数据	0
Reverse	反转样条曲线的方向	0
SetControlPoint	设定给定索引处样条曲线的拟合点	2
SetFitPoint	在样条曲线给定索引处设定拟合点	2
SetWeight	设定样条曲线给定索引处控制点的权重	2

单行文字（Text）

单行文字（Text）的属性说明如下：

属性	说明	数据类型
Alignment	单行文字水平和竖向对齐的方式	Integer/Enum
Backward	单行文字是否左右反向	Boolean
Height	单行文字的高度	Double
InsertionPoint	单行文字的插入基点	Array (Double)
ObliqueAngle	单行文字的倾斜角度（弧度制）	Double
Rotation	单行文字的旋转角度（弧度制）	Double
ScaleFactor	单行文字的宽度比例因子	Double
StyleName	单行文字的文字样式名称	String
TextAlignmentPoint	单行文字的对齐点	Array (Double)
TextGenerationFlag	单行文字的生成标志（颠倒反向特性）	Integer/Enum
TextString	单行文字字符串	String
UpsideDown	单行文字是否上下颠倒	Boolean

单行文字（Text）的方法说明如下：

方法	说明	参数数量
FieldCode	获取含有字段代码的字符串	0

跟踪（Trace）

跟踪（Trace）^①的属性说明如下：

^① 译者注：这个翻译我自己也感觉很奇怪，但在 AutoCAD® 2020 中文版里，选中这类图元在 [特性] 栏里显示的就是“跟踪”。这是一类非常古老的图元，表达带宽度的直线，使用 TRACE 命令创建。这个命令我从来都没有用过，有这种需要的时候我都使用轻量化多段线（LWPOLYLINE）。如果不翻译这本书，我甚至都不知道这个图元的存在。事实上，TRACE 命令在 AutoCAD® 2012 至 AutoCAD® 2021 中已经被删掉了，为了向后兼容，跟踪（TRACE）图元还保留着，如果你有考古的兴趣一定要用命令创建这种图元，可以用加 . 的内部命令 .TRACE。为什么说是 AutoCAD® 2012 至 AutoCAD® 2021 中被删掉呢？因为从 AutoCAD® 2022 起，TRACE 这个命令又被加回来了，不过命令是用于在 AutoCAD Web 应用程序、移动应用程序或桌面程序中创建跟踪（这回才是名副其实的“跟踪”），以提供图形的反馈、注释、标记和设计研究，但并不改变图形的内容。至于 .TRACE 这时究竟是哪个命令，我不知道，我最高只用过 AutoCAD® 2021。

属性	说明	数据类型
Coordinate	跟踪的单个顶点坐标	Array (Double)
Coordinates	跟踪的每个顶点坐标	Array (Double)


视口 (**Viewport**/**PViewport**)

视口 (**Viewport**/**PViewport**) 的属性说明如下:

属性	说明	数据类型
ArcSmoothness	视口中圆、弧和椭圆的平滑度	Integer
Center	视口的中心点	Array (Double)
Clipped(RO)^a	视口是否被裁剪	Integer/Boolean
CustomScale	视口的自定义比例因子	Double
Direction	视口的查看方向向量	Array (Double)
DisplayLocked	视口显示是否被锁定	Boolean
GridOn	视口网格是否被打开	Boolean
Height	视口对象在图纸空间的高度	Double
LensLength	视口的透视镜头长度	Double
SnapBasePoint	视口的捕捉基点	Array (Double) ^b
SnapOn	视口的捕捉是否开启	Boolean
SnapRotationAngle	视口相对于当前 UCS 的捕捉旋转角度	Double
StandardScale	视口的标准比例	Integer/Enum
Target	视口的目标点	Array (Double)
TwistAngle	视口的扭曲角度 (弧度制)	Double
UCSIconAtOrigin	UCS 标志是否显示在原点	Boolean
UCSIconOn	UCS 标志显示是否打开	Boolean
UCSPerViewport	UCS 坐标系是否随视口保存	Boolean

属性	说明	数据类型
ViewportOn	视口显示是否被打开	Boolean
Width	视口对象在图纸空间的宽度	Double

^a 仅适用于 PViewport 对象
^b 二维 WCS 坐标点



RemoveHiddenLines 属性是过时的属性，未来将从 AutoCAD 中移除。

构造线 (XLine)

构造线 (XLine) 的属性说明如下：

属性	说明	数据类型
BasePoint	构造线经过的基点	Array (Double)
DirectionVector	构造线方向的矢量	Array (Double)
SecondPoint	构造线经过的第二点	Array (Double)

表格 (Table)

表格 (Table) 的属性说明如下：

属性	说明	数据类型
AllowManualHeights	是否允许表格对象高度动态调整	Boolean
AllowManualPositions	是否允许表格动态定位	Boolean
BreaksEnabled	是否允许表格断开	Boolean
BreakSpacing	表格断开的水平或竖向间距	Double
Columns	表格的列数量	Integer
ColumnWidth	表格列的统一宽度	Double

属性	说明	数据类型
Direction	表格列的排列方向向量	Array (Double)
EnableBreak	切换表格断开状态	Boolean
FlowDirection	标题和标题行是在表格的底部还是顶部	Integer/Enum
HasSubSelection(R0)	表格是否包含子选择集	Boolean
HeaderSuppressed	是否隐藏表格的表头	Boolean
Height	表格对象的高度	Double
HorzCellMargin	表格单元格水平边距	Double
InsertionPoint	表格的插入基点	Array (Double)
MinimumTableHeight(R0)	表格的最小高度	Double
MinimumTableWidth(R0)	表格的最小宽度	Double
RegenerateTableSuppressed	是否抑制表格的重生	Boolean
RepeatBottomLabels	底部标签行是否在每个断开的部分的底部重复	Boolean
RepeatTopLabels	顶部标签行是否在每个断开的部分的顶部重复	Boolean
RowHeight	表格行的统一高度	Double
Rows	表格的列数量	Integer
StyleName	表格的样式名称	String
TableBreakFlowDirection	表格断开部分排列方向	Integer/Enum
TableBreakHeight	初始表格部件和任何其他未手动设置高度的表格部件的断开高度	Double
TableStyleOverrides(R0)	表格的样式替代	String
TitleSuppressed	是否隐藏表格的标题	Boolean
VertCellMargin	表格单元格垂直边距	Double
Width	表格对象的宽度	Double

表格（Table）的方法说明如下：

方法	说明	参数数量
<code>ClearSubSelection</code>	移去表格中单元格的子选择集	0
<code>ClearTableStyleOverrides</code>	清除表格样式替换	1
<code>CreateContent</code>	在单元格中创建新内容	3
<code>DeleteCellContent</code>	删除指定行和列的单元格内容	2
<code>DeleteColumns</code>	删除表格中的列	2
<code>DeleteContent</code>	删除单元格中内容	2
<code>DeleteRows</code>	删除表格中的行	2
<code>EnableMergeAll</code>	控制表格的合并状态	3
<code>FormatValue</code>	获取指定行和列的格式化文字字符串	4
<code>GenerateLayout</code>	生成表格的布局	0
<code>GetAlignment</code>	获取行类型的单元格对齐方式	1
<code>GetAttachmentPoint</code>	获取指定行和列的附着点	2
<code>GetAutoScale</code>	获取指定的行和列是否使用自动缩放比例的值	2
<code>GetAutoScale2</code>	获取单元格的自动缩放标志值	3
<code>GetBackgroundColor</code>	获取指定行类型的背景颜色值	1
<code>GetBackgroundColorNone</code>	获取指定的行类型是否没有背景颜色的值	1
<code>GetBlockAttributeValue</code>	获取指定单元格的块中所包含的属性定义对象的属性值	3
<code>GetBlockAttributeValue2</code>	获取指定行和列的块的旋转角度	4
<code>GetBlockRotation</code>	获取指定行和列的块的旋转角度	2
<code>GetBlockScale</code>	获取指定行和列的块的比例因子	2
<code>GetBlockTableRecordId</code>	获取单元格的块表格记录 ID	2
<code>GetBlockTableRecordId2</code>	获取与指定单元格关联的块表格记录对象 ID	3
<code>GetBoundingBox</code>	获取对象边框的最大和最小点	2
<code>GetBreakHeight</code>	获取表格的断开高度	1
<code>GetCellAlignment</code>	获取指定行和列的单元格的对齐方式	2

方法	说明	参数数量
<code>GetCellBackgroundColor</code>	获取指定行和列的单元格的背景真彩色值	2
<code>GetCellBackgroundColorNone</code>	获取指定的行和列是否无背景颜色	2
<code>GetCellContentColor</code>	获取指定行和列中内容的真彩色值	2
<code>GetCellDataType</code>	获取给定行和列的单元格数据和单位类型	4
<code>GetCellExtents</code>	获取指行和列的单元格区域	3
<code>GetCellFormat</code>	获取给定行和列的单元格格式	2
<code>GetCellGridColor</code>	获取指定行和列的一个边的格线颜色值	3
<code>GetCellGridLineWeight</code>	获取指定行和列的一个边的格线线宽	3
<code>GetCellGridVisibility</code>	获取指定行和列的一个边的格线可见性	3
<code>GetCellState</code>	获取单元格的状态	2
<code>GetCellStyle</code>	获取单元格的样式	2
<code>GetCellStyleOverrides</code>	获取单元格样式替换	2
<code>GetCellTextHeight</code>	获取指定行和列的文字高度	2
<code>GetCellTextStyle</code>	获取指定行和列的文字样式名称	2
<code>GetCellType</code>	获取指定行和列的单元格类型	2
<code>GetCellValue</code>	获取给定行和列的单元格值	2
<code>GetColumnName</code>	获取列的名称	1
<code>GetColumnWidth</code>	获取表格中指定列索引所在的列的列宽	1
<code>GetContentColor</code>	获取指定行类型的真彩色值	1
<code>GetContentColor2</code>	获取单元格的真彩色值	3
<code>GetContentLayout</code>	获取单元格的内容布局	2
<code>GetContentType</code>	获取单元格的内容类型	2
<code>GetCustomData</code>	获取与单元格关联的自定义数据	4
<code>GetDataFormat</code>	获取单元格的数据格式	3
<code>GetDataType</code>	返回指定行类型的数据类型和单位类型	3

方法	说明	参数数量
<code>GetDataType2</code>	获取单元格的行数据类型和单位类型	5
<code>GetExtensionDictionary</code>	获取与对象关联的扩展词典	0
<code>GetFieldId</code>	获取指定单元格的字段对象 ID	2
<code>GetFieldId2</code>	获取与指定单元格关联的字段对象 ID	3
<code>GetFormat</code>	获取指定行类型的格式	1
<code>GetFormula</code>	获取单元格的公式	3
<code>GetGridColor</code>	获取指定格线类型和行类型的格线颜色值	2
<code>GetGridColor2</code>	获取指定单元格的网格颜色	3
<code>GetGridDoubleLineSpacing</code>	获取指定单元格的行距值	3
<code>GetGridLineStyle</code>	获取单元格的网格线样式	3
<code>GetGridLinetype</code>	获取具有指定单元格的网格线型值的线型的对象 Id	3
<code>GetGridLineWeight</code>	获取指定格线类型和行类型的格线线宽值	2
<code>GetGridLineWeight2</code>	获取网格线型和行类型的网格线宽值	3
<code>GetGridVisibility</code>	获取指定网格线型和行类型的网格可见性值	2
<code>GetGridVisibility2</code>	获取指定单元格的网格可见性值	3
<code>GetHasFormula</code>	检查单元格是否有公式	3
<code>GetMargin</code>	获取单元格的边距值	3
<code>GetMinimumColumnWidth</code>	获取表格中指定列索引所在的列的最小列宽	1
<code>GetMinimumRowHeight</code>	获取指定行的最小行高	1
<code>GetOverride</code>	获取单元格的覆盖	3
<code>GetRotation</code>	获取单元格的旋转角度	3
<code>GetRowHeight</code>	获取表格中指定行索引所在的行的行高	1
<code>GetRowType</code>	获取指定行的行类型	1
<code>GetScale</code>	获取单元格的比例值	3
<code>GetSubSelection</code>	获取在子选择集中单元格的行和列索引	4

方法	说明	参数数量
<code>GetText</code>	获取指定行和列的文字值	2
<code>GetTextHeight</code>	获取指定行类型的文字高度	1
<code>GetTextHeight2</code>	获取单元格的文字高度	3
<code>GetTextRotation</code>	获取指定行和列的文字旋转角度	2
<code>GetTextString</code>	获取单元格的文字字符串	3
<code>GetTextStyle</code>	获取指定行类型的文字样式名称	1
<code>GetTextStyle2</code>	获取单元格的文字样式名称	3
<code>GetValue</code>	获取单元格的值	3
<code>HitTest</code>	返回指定位置的单元格	4
<code>InsertColumns</code>	在表格中插入列	3
<code>InsertColumnsAndInherit</code>	将列从另一个表插入到表格中	3
<code>InsertRows</code>	在表格中插入行	3
<code>InsertRowsAndInherit</code>	将行从另一个表插入到表格中	3
<code>IntersectWith</code>	获取表格对象与图形中其它对象的相交点	2
<code>IsContentEditable</code>	检查单元格是否具有可编辑的内容	2
<code>IsEmpty</code>	检查单元格是否为空	2
<code>IsFormatEditable</code>	检查单元格是否具有可编辑格式	2
<code>IsMergeAllEnabled</code>	检查单元格是否在合并状态	2
<code>IsMergedCell</code>	获取单元格的合并状态	6
<code>MergeCells</code>	合并表格中的单元格	4
<code>MoveContent</code>	移动单元格的索引内容	4
<code>RecomputeTableBlock</code>	更新表格块	1
<code>RemoveAllOverrides</code>	删除单元格的所有覆盖	2
<code>ReselectSubRegion</code>	重新选择当前子选择集	0
<code>ResetCellValue</code>	重置给定行和列的单元格值	2

方法	说明	参数数量
Select	选择表格中的单元格	8
SelectSubRegion	选择表格中一些单元格	10
SetAlignment	设置指定行类型的单元格对齐方式	2
SetAutoScale	设置指定的行和列是否使用自动缩放比例的值	3
SetAutoScale2	设置单元格的自动缩放标志值	4
SetBackgroundColor	设置指定行类型的背景颜色值	2
SetBackgroundColorNone	设置指定的行类型是否没有背景颜色的值	2
SetBlockAttributeValue	设置指定单元格的块中所包含的属性定义对象的属性值	4
SetBlockAttributeValue2	设置与指定块单元关联的属性值	5
SetBlockRotation	设置指定行和列的块的旋转角度	3
SetBlockScale	设置指定行和列的块的比例因子	3
SetBlockTableRecordId	设置单元格的块表格记录 ID	4
SetBlockTableRecordId2	设置与指定单元格关联的块表记录对象 ID	5
SetBreakHeight	设置表格的断开高度	2
SetCellAlignment	设置指定行和列的单元格的对齐方式	3
SetCellBackgroundColor	设置指定行和列的单元格的背景真彩色值	3
SetCellBackgroundColorNone	设置指定的行和列是否无背景颜色	3
SetCellContentColor	设置指定行和列中内容的真彩色值	3
SetCellDataType	设置给定行和列的单元格数据和单位类型	4
SetCellFormat	设置给定行和列的单元格格式	3
SetCellGridColor	设置指定行和列的一个边的格线颜色值	4
SetCellGridLineWeight	设置指定行和列的一个边的格线线宽	4
SetCellGridVisibility	设置指定行和列的一个边的格线可见性值	4
SetCellState	设置单元格的状态	3
SetCellStyle	设置指定行和列的单元格类型	3

方法	说明	参数数量
<code>SetCellTextHeight</code>	设置指定行和列的文字高度	3
<code>SetCellTextStyle</code>	设置指定行和列的文字样式名称	3
<code>SetCellType</code>	设置指定行和列的单元格类型	3
<code>SetCellValue</code>	设置给定行和列的单元格值	3
<code>SetCellValueFromText</code>	设置给定行和列的单元格值	4
<code>SetColumnName</code>	为列设置名称	2
<code>SetColumnWidth</code>	设置表格中指定列索引所在的列的列宽	2
<code>SetContentColor</code>	设置指定行类型的真彩色值	2
<code>SetContentColor2</code>	根据行、列和内容位置设置单元格内容的颜色值	4
<code>SetContentLayout</code>	设置单元格的内容布局	3
<code>SetCustomData</code>	设置与单元格关联的自定义数据	4
<code>SetDataFormat</code>	设置单元格的格式	4
<code>SetDataType</code>	为指定的行类型设置数据类型和单位类型	3
<code>SetDataType2</code>	为指定的行类型和内容设置行数据类型和单位类型	5
<code>SetFieldId</code>	设置指定单元格的字段对象 ID	3
<code>SetFieldId2</code>	设置关联到指定单元格的字段对象 ID	5
<code>SetFormat</code>	设置指定行类型的格式	2
<code>SetFormula</code>	设置单元格的公式	4
<code>SetGridColor</code>	设置指定格线类型和行类型的格线颜色值	3
<code>SetGridColor2</code>	设置指定单元格的网格颜色	4
<code>SetGridDoubleLineSpacing</code>	设置指定单元格的行间距值	4
<code>SetGridLineStyle</code>	设置单元格的网格线样式	4
<code>SetGridLinetype</code>	为指定单元格设置具有网格线型值的对象	4
<code>SetGridLineWeight</code>	设置指定格线类型和行类型的格线线宽值	3
<code>SetGridLineWeight2</code>	设置指定单元格的网格线宽值	4

方法	说明	参数数量
<code>SetGridVisibility</code>	设置指定格线类型和行类型的格线可见性	3
<code>SetGridVisibility2</code>	设置指定单元格或单元格样式的网格可见性值	4
<code>SetMargin</code>	设置单元格的边距	4
<code>SetOverride</code>	设置单元格的覆盖	4
<code>SetRotation</code>	设置单元格的旋转角度（弧度制）	4
<code>SetRowHeight</code>	设置表格中指定行索引所在的行的行高	2
<code>SetScale</code>	设置单元格的比例值	4
<code>SetSubSelection</code>	设置子选择集中单元格的行和列索引	4
<code>SetText</code>	设置指定行和列的文字值	3
<code>SetTextHeight</code>	设置指定行类型的文字高度	2
<code>SetTextHeight2</code>	设置单元格的文字高度	4
<code>SetTextRotation</code>	设置行和列的文字旋转角度	3
<code>SetTextString</code>	设置单元格的文字字符串	4
<code>SetTextStyle</code>	设置指定行类型的文字样式名称	2
<code>SetTextStyle2</code>	设置单元格的文字样式名称	4
<code>SetToolTip</code>	设置单元格的工具提示	3
<code>SetValue</code>	为单元格赋值	4
<code>SetValueFromText</code>	解析格式化文字并给单元格赋值	5
<code>UnmergeCells</code>	拆分合并的单元格	4

文档

绘图文件在一般意义上被视为文档 (Documents)，这就是在 AutoCAD 以及其它大多数支持 ActiveX 的应用程序从编程角度引用它们的方式。Document 对象是 AutoCAD 中 Documents 集合的成员。

要访问当前绘图会话，你可以请求 AcadApplication 对象的 ActiveDocument 属性，而无需转到 Documents 集合。但是，如果你需要访问另一个文档或迭代遍历所有打开的文档，则需要访问 Documents 集合。

18.1 Documents 集合

Documents 集合包含活动 AutoCAD 会话中所有打开的文档。每次创建或打开另一个绘图文件时，它会立即将其添加到此集合中。文档通常按照打开的顺序输入到 Documents 集合中。要导航集合，可以使用 (vlax-for) 函数处理所有成员，如果需要，也可以使用 Item 方法按索引位置或名称访问单个成员。

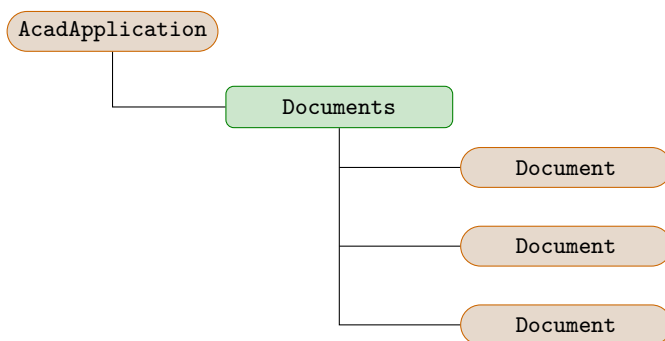


图 18.1 Documents 集合与 Document 对象

例 18.1 检索所有打开的文档名称列表

```
(defun Documents-ListAll ( / out)
  (vlax-for each (vla-get-documents (vlax-get-acad-object))
    (setq out (cons (vla-get-name each) out)))
  (if out (reverse out)))
```

例 18.1 显示了一个示例代码片段，用于检索当前 AutoCAD 会话中所有打开的文档名称的列表。使用与 AutoLISP (`foreach`) 函数几乎相同的 (`vlax-for`) 迭代函数，我们可以迭代遍历 `Documents` 集合并获取每个文档的 `Name` 属性并生成表输出。

你可以非常轻松地调整这段代码以在每个文档上执行其他任务，或者搜索文档中的特定条件并根据结果采取行动。

还可以使用 `Documents` 集合做什么呢？好吧，让我们从检查 `Documents` 集合支持哪些属性和方法开始：

```
Command: (setq docs (vla-get-Documents acadapp))
#<VLA-OBJECT IAcadDocuments 00f20440>
```

```
Command: (vlax-dump-object docs T)

; IAcadDocuments: The collection of all AutoCAD drawings open in the
  ↳ current session
; Property values:
; Application (R0) = #<VLA-OBJECT IAcadApplication 00a8a730>
; Count (R0) = 1
; Methods supported:
; Add (1)
; Close ()
; Item (1)
; Open (2)
```

`Documents` 集合只有两个属性，但是有四个方法。正如我之前提到的，Visual LISP 不提供修改集合属性的方法。然而，它通常有一组用于添加、访问和删除成员的方法。在这种情况下，`Add` 方法与命令 `NEW` 同义，而 `Open` 方法与命令 `OPEN` 相同。

要访问已打开的单个绘图文件，可以对文档名称或索引号（它在集合中的位置）使用 `Item` 方法。我将获取我现在打开的 `Drawing1.dwg` 文档的对象：


```
Command: (setq docs (vla-get-Documents acadapp))
#<VLA-OBJECT IAcadDocuments 00f20440>
```

现在就可以检查这个文档对象以查看它提供了哪些属性和方法:

```
Command: (vlax-dump-object docs T)

; IAcadDocument: An AutoCAD drawing
; Property values:
; Active (RO) = -1
; ActiveDimStyle = #<VLA-OBJECT IAcadDimStyle 110ed974>
; ActiveLayer = #<VLA-OBJECT IAcadLayer 110edb04>
; ActiveLayout = #<VLA-OBJECT IAcadLayout 110edba4>
; ActiveLinetype = #<VLA-OBJECT IAcadLinetype 110edb54>
; ActiveMaterial = #<VLA-OBJECT IAcadMaterial 110edbf4>
; ActivePViewport = AutoCAD: No active viewport in paperspace
; ActiveSelectionSet (RO) = #<VLA-OBJECT IAcadSelectionSet 11158d64>
; ActiveSpace = 1
; ActiveTextStyle = #<VLA-OBJECT IAcadTextStyle 110edc94>
; ActiveUCS = AutoCAD: Null object ID
; ActiveViewport = #<VLA-OBJECT IAcadViewport 110edd34>
; Application (RO) = #<VLA-OBJECT IAcadApplication 01877c20>
; Blocks (RO) = #<VLA-OBJECT IAcadBlocks 110edd84>
; Database (RO) = #<VLA-OBJECT IAcadDatabase 11158ac4>
; Dictionaries (RO) = #<VLA-OBJECT IAcadDictionaries 110edce4>
; DimStyles (RO) = #<VLA-OBJECT IAcadDimStyles 110eddd4>
; ElevationModelSpace = 0.0
; ElevationPaperSpace = 0.0
; FileDependencies (RO) = #<VLA-OBJECT IAcadFileDependencies 110d2eb4>
; FullName (RO) = ""
; Groups (RO) = #<VLA-OBJECT IAcadGroups 110ede74>
; Height = 515
; HWND (RO) = 2032484
; Layers (RO) = #<VLA-OBJECT IAcadLayers 110edec4>
; Layouts (RO) = #<VLA-OBJECT IAcadLayouts 110ede24>
; Limits = (0.0 0.0 12.0 9.0)
; Linetypes (RO) = #<VLA-OBJECT IAcadLineTypes 110edf14>
; Materials (RO) = #<VLA-OBJECT IAcadMaterials 110edf64>
; ModelSpace (RO) = #<VLA-OBJECT IAcadModelSpace 110edfb4>
; MSpace = AutoCAD: Invalid mode
```

```

; Name (RO) = "Drawing1.dwg"
; ObjectSnapMode = 0
; PaperSpace (RO) = #<VLA-OBJECT IAcadPaperSpace 110ee054>
; Path (RO) = "C:\\Users\\dstein\\Documents"
; PickfirstSelectionSet (RO) = #<VLA-OBJECT IAcadSelectionSet 11158f74>
; Plot (RO) = #<VLA-OBJECT IAcadPlot 110d85ec>
; PlotConfigurations (RO) = #<VLA-OBJECT IAcadPlotConfigurations 110ee0a4>
; Preferences (RO) = #<VLA-OBJECT IAcadDatabasePreferences 110d2edc>
; ReadOnly (RO) = 0
; RegisteredApplications (RO) = #<VLA-OBJECT IAcadRegisteredApplications
    ↪ 110ee0f4>
; Saved (RO) = -1
; SectionManager (RO) = Exception occurred
; SelectionSets (RO) = #<VLA-OBJECT IAcadSelectionSets 10fef9a4>
; SummaryInfo (RO) = #<VLA-OBJECT IAcadSummaryInfo 110d2e8c>
; TextStyles (RO) = #<VLA-OBJECT IAcadTextStyles 110ee144>
; UserCoordinateSystems (RO) = #<VLA-OBJECT IAcadUCSs 110ee004>
; Utility (RO) = #<VLA-OBJECT IAcadUtility 11114f54>
; Viewports (RO) = #<VLA-OBJECT IAcadViewports 110ee194>
; Views (RO) = #<VLA-OBJECT IAcadViews 110ee1e4>
; Width = 1020
; WindowState = 3
; WindowTitle (RO) = "Drawing1.dwg"
; Methods supported:
; Activate ()
; AuditInfo (1)
; Close (2)
; CopyObjects (3)
; EndUndoMark ()
; Export (3)
; GetVariable (1)
; HandleToObject (1)
; Import (3)
; LoadShapeFile (1)
; New (1)
; ObjectIdToObject (1)
; Open (2)
; PurgeAll ()

```

```
; Regen (1)
; Save ()
; SaveAs (3)
; SendCommand (1)
; SetVariable (2)
; StartUndoMark ()
; Wblock (2)
```

哇！如果还没有习惯使用 Visual LISP 或 ActiveX，你应该会感慨这对软件开发人员来说有多么强大。仔细查看上述结果将揭示可以获得和修改关于给定文档的所有内容。使用 Visual LISP 之前的 AutoLISP 根本不可能做到这一点。如你所见，你现在可以直接访问该文档中的所有表，实际上是集合，以及系统变量、方法等。

请注意，Plot 是属性而不是一种方法。这是因为 Document 对象的 Plot 属性实际上是指向 Plot 对象的指针。Plot 对象是在 AutoCAD 的 ActiveX 世界中配置和执行打印的方式。

让我们将其中一些方法与 Documents 集合结合使用，看看我们如何迭代遍历所有打开的文档并对每个文档执行一个简单的任务。如果我们想对所有打开的绘图文件运行 **AUDIT** 命令怎么样：

```
(defun AllDocs-Audit ( / docs)
  (vlax-for dwg (vla-get-Documents (vlax-get-acad-object))
    (vla-AuditInfo dwg T))) ; T denotes fix errors = Yes
```

我们还可以保存所有打开的文档，如下所示：

```
(defun AllDocs-Save ( / docs)
  (vlax-for dwg (vla-get-Documents (vlax-get-acad-object))
    (vla-Save dwg)))
```

对所有打开的文档运行 **PURGE** 命令是怎样的呢？


```
(defun AllDocs-Purge ( / docs)
  (vlax-for dwg (vla-get-Documents (vlax-get-acad-object))
    (vla-PurgeAll dwg)))
```



迭代遍历 `Documents` 集合以执行某些任务时要小心。请记住，`LISP` 在文档上下文中运行，而不是在应用程序上下文中运行。这意味着虽然可以在同一应用程序命名空间中对其他打开的文档进行操作，但仍然从执行函数的文档中调用更改。你应该始终尝试让代码在完成时或由于错误而失败时将焦点返回到原始文档。如果你计划实施反应器并从你实例化反应器和回调的绘图会话以外的其他绘图会话调用它们的回调，这一点尤其重要。

Preferences 对象

这不是印刷错误，本章标题提到了对象（重点是复数）。原因很简单，AutoCAD 中有两个主要的 **Preferences** 集合对象。第一个是 **AcadPreferences** 集合对象，它适用于 AutoCAD 本身。第二个是 **Document** 对象的 **Preferences** 集合对象，也称为 **DatabasePreferences** 集合对象。后者仅适用于 **Document** 对象，不适用于 AutoCAD。

举一个更好的例子，如果你打开 **选项** 对话框并浏览所有可用的选项卡，你会注意到许多设置旁边都有一个小图形文件图标 。这表示设置仅保存到当前图形文件中，不会传递到其他图形文件。这些项目实际上是 **DatabasePreferences** 集合的一部分。

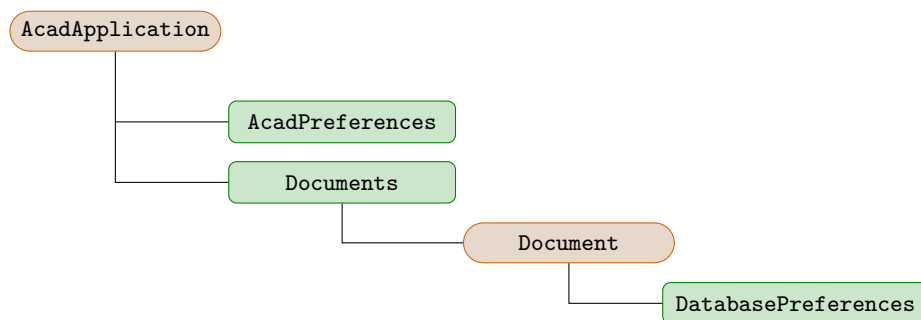


图 19.1 **AcadPreferences** 与 **DatabasePreferences** 集合对象

19.1 AcadPreferences

AcadPreferences 集合实际上是其他集合的容器，每个集合都有自己的对象。**选项** 对话框的图形特性在表示这些集合的方式上并不总是正确的，不应用作理解这些集合的指南或地图。

有些对象具有不同的名称，有些在 **选项** 中显示为集合，但在 **AcadPreferences** 集合中存储为单个对象。

举例来说，**选项** 文件选项卡中显示的支持路径列表显示为子项列表、路径名的字符串值。这很容易被误认为是路径的集合。但实际上它是单个字符串值，每个路径值之间有一个分号(;)分隔符，并存储为 **PreferencesFiles** 对象中的 **SupportPath** 属性。令人困惑？有可能。另一个例子是 **选项** 对话框窗体的 **文件** 选项卡上的数据源位置路径设置。这实际上存储为 **PreferencesFiles** 对象下的 **WorkSpace** 属性，而不是 **DataSourcesPath** 或更直观命名的东西。

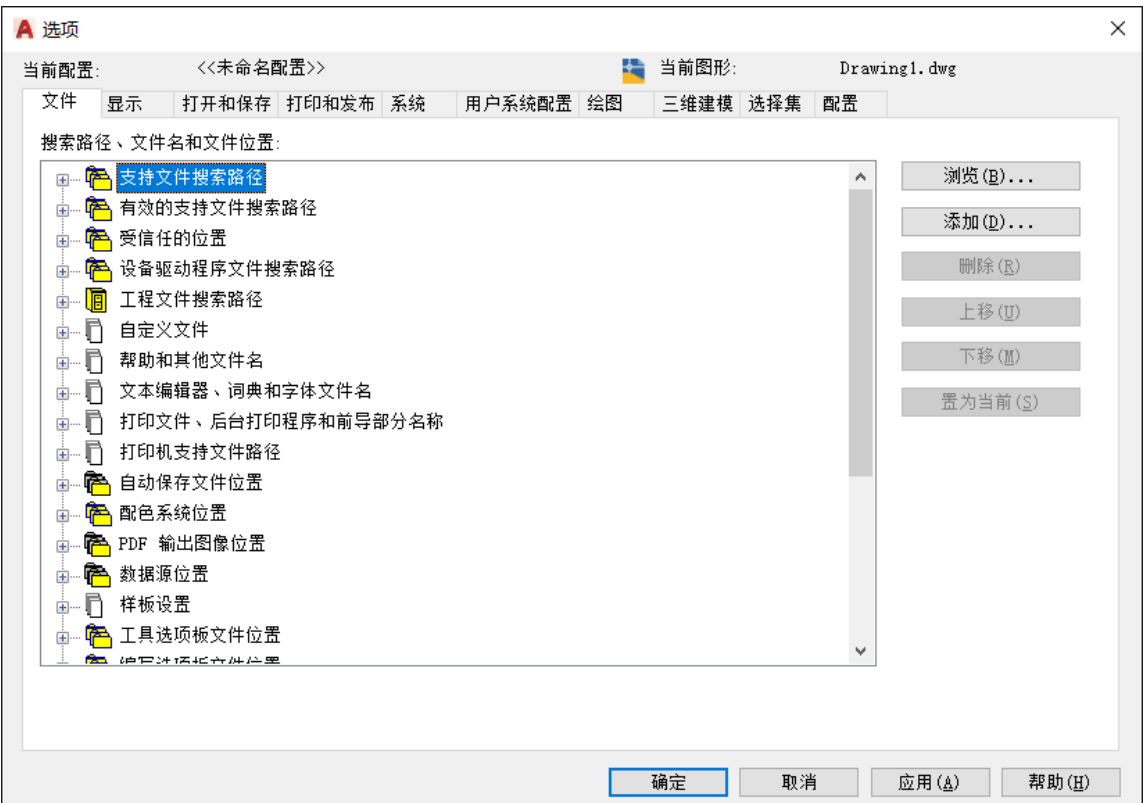


图 19.2 显示文件选项卡的 AutoCAD 选项对话框

AcadPreferences 集合中一共有九个对象，一个属性（应用程序属性），没有方法。**AcadPreferences** 集合大致对应于 **选项** 对话框选项卡。下面是 **选项** 对话框表单中的对象名称及其相应选项卡的列表：

PreferencesDisplay	显示
PreferencesDrafting	绘图
PreferencesFiles	文件

PreferencesOpenSave	打开和保存
PreferencesOutput	打印和发布
PreferencesProfiles	配置
PreferencesSelection	选择集
PreferencesSystem	系统
PreferencesUser	用户系统配置



三维建模选项卡明显不在 **AcadPreferences** 集合中。

下例显示了 **AcadPreferences** 集合对象的导出。

```
Command: (vlax-dump-object (vla-get-Preferences (vlax-get-Acad-object)) t)
; IAcadPreferences: This object specifies the current AutoCAD settings
; Property values:
; Application (RO) = #<VLA-OBJECT IAcadApplication 00a8a730>
; Display (RO) = #<VLA-OBJECT IAcadPreferencesDisplay 04c5df7c>
; Drafting (RO) = #<VLA-OBJECT IAcadPreferencesDrafting 04c5df78>
; Files (RO) = #<VLA-OBJECT IAcadPreferencesFiles 04c5df80>
; OpenSave (RO) = #<VLA-OBJECT IAcadPreferencesOpenSave 04c5df84>
; Output (RO) = #<VLA-OBJECT IAcadPreferencesOutput 04c5df88>
; Profiles (RO) = #<VLA-OBJECT IAcadPreferencesProfiles 04c5df8c>
; Selection (RO) = #<VLA-OBJECT IAcadPreferencesSelection 04c5df90>
; System (RO) = #<VLA-OBJECT IAcadPreferencesSystem 04c5df94>
; User (RO) = #<VLA-OBJECT IAcadPreferencesUser 04c5df98>
; No methods
```

为了更深入，我们将检查 **PreferencesFiles** 对象以查看它包含的内容。本书中将采用 ⇨ 表示字符串在排版时的换行^①。

```
Command: (vlax-dump-object (vla-get-Files (vla-get-Preferences
(vlax-get-Acad-object))) T)
; IAcadPreferencesFiles: This object contains the options from the Files
⇨ tab on the Options dialog
; Property values:
; AltFontFile = "simplex.shx"
; AltTabletMenuFile = ""
; Application (RO) = #<VLA-OBJECT IAcadApplication 01877c20>
```

^① 译者注：本书排版时改变了原作者采用的换行方式。

```
; AutoSavePath = "C:\\Users\\dstein\\appdata\\local\\temp\\"
; ColorBookPath = "c:\\program files\\autodesk\\autocad 2011\\support\\"
    ↳ color;C:\\Users\\dstein\\appdata\\roaming\\autodesk\\autocad 2011\\
    ↳ r18.1\\enu\\support\\color"
; ConfigFile (R0) = "C:\\Users\\dstein\\AppData\\Local\\Autodesk\\AutoCAD
    ↳ 2011\\R18.1\\enu\\acad2011.cfg"
; CustomDictionary = "C:\\Users\\dstein\\appdata\\roaming\\autodesk\\
    ↳ autocad 2011\\r18.1\\enu\\support\\sample.cus"
; CustomIconPath = "C:\\Users\\dstein\\appdata\\roaming\\autodesk\\autocad
    ↳ 2011\\r18.1\\enu\\support\\icons"
; DefaultInternetURL = "http://www.autodesk.com"
; DriversPath = "C:\\Program Files\\Autodesk\\AutoCAD 2011\\drv"
; EnterpriseMenuFile = "."
; FontFileMap = "C:\\Users\\dstein\\appdata\\roaming\\autodesk\\autocad
    ↳ 2011\\r18.1\\enu\\support\\acad.fmp"
; HelpFilePath = "C:\\Program Files\\Autodesk\\AutoCAD 2011\\Help\\index.
    ↳ html"
; LogFilePath = "C:\\Users\\dstein\\appdata\\local\\autodesk\\autocad
    ↳ 2011\\r18.1\\enu\\"
; MainDictionary = "enu"
; MenuFile = "C:\\Users\\dstein\\appdata\\roaming\\autodesk\\autocad
    ↳ 2011\\r18.1\\enu\\support\\acad"
; PageSetupOverridesTemplateFile = "C:\\Users\\dstein\\appdata\\local\\
    ↳ autodesk\\autocad 2011\\r18.1\\enu\\template\\sheetsets\\
    ↳ architectural imperial.dwt"
; PlotLogFilePath = "C:\\Users\\dstein\\appdata\\local\\autodesk\\autocad
    ↳ 2011\\r18.1\\enu\\"
; PostScriptPrologFile = ""
; PrinterConfigPath = "C:\\Users\\dstein\\appdata\\roaming\\autodesk\\
    ↳ autocad 2011\\r18.1\\enu\\plotters"
; PrinterDescPath = "C:\\Users\\dstein\\appdata\\roaming\\autodesk\\
    ↳ autocad 2011\\r18.1\\enu\\plotters\\pmp files"
; PrinterStyleSheetPath = "C:\\Users\\dstein\\appdata\\roaming\\autodesk\\
    ↳ autocad 2011\\r18.1\\enu\\plotters\\plot styles"
; PrintFile = "."
; PrintSpoolerPath = "C:\\Users\\dstein\\appdata\\local\\temp\\"
; PrintSpoolExecutable = ""
; QNewTemplateFile = ""
```



```

; SupportPath = "C:\\Users\\dstein\\appdata\\roaming\\autodesk\\autocad
    ↳ 2011\\r18.1\\enu\\support;C:\\program files\\autodesk\\autocad
    ↳ 2011\\support;C:\\program files\\autodesk\\autocad 2011\\fonts;C:\\
    ↳ program files\\autodesk\\autocad 2011\\help;C:\\program files\\
    ↳ autodesk\\autocad 2011\\express;C:\\program files\\autodesk\\
    ↳ autocad 2011\\support\\color"
; TempFilePath = "C:\\Users\\dstein\\appdata\\local\\temp\\"
; TemplateDwgPath = "C:\\Users\\dstein\\appdata\\local\\autodesk\\autocad
    ↳ 2011\\r18.1\\enu\\template"
; TempXrefPath = "C:\\Users\\dstein\\appdata\\local\\temp\\"
; TextEditor = "Internal"
; TextureMapPath = "C:\\Program Files\\Common Files\\Autodesk Shared\\
    ↳ Materials2011\\assetlibrary_base.fbm\\1\\Mats"
; ToolPalettePath = "C:\\Users\\dstein\\AppData\\Roaming\\Autodesk\\
    ↳ AutoCAD 2011\\R18.1\\enu\\support\\ToolPalette"
; WorkspacePath = "C:\\Users\\dstein\\appdata\\roaming\\autodesk\\autocad
    ↳ 2011\\r18.1\\enu\\data links"
; Methods supported:
; GetProjectFilePath (1)
; SetProjectFilePath (2)

```

请注意, `PreferencsFiles` 对象有很多属性, 但只提供两种方法。此外注意 `SupportPath` 设置将搜索路径列表显示为单个字符串, 每个路径值之间使用分号 (;) 分隔符。

对这一点最重要的理解是: 可以使用 `(vla-get-<xxx>)` 和 `(vla-put-<xxx>)` 函数来获取和修改在整个 `AcadPreferences` 集合对象中显示的任何属性, 只要它们不是只读 (RO) ^①。想一想这个问题应该会明白, 这为开发人员提供了巨大的能力和灵活性。你可以轻松地以编程方式操作 AutoCAD 配置。更进一步, 当你开始使用配置文件时, 你会发现这开启了在网络环境中远程管理桌面的无限可能。

我们来演示如何将其与 Visual LISP 结合使用。假设你要修改网络上的所有 AutoCAD 安装, 以更改每个客户端查找图形模板文件的默认路径设置。也许你希望他们都使用一组标准的自定义模板, 这些模板存储在网络上共享服务器上的一个文件夹中。为此, 你只需将更改推送到 `PreferencsFiles` 集合对象下的 `TemplateDwgPath` 属性。当然, 你可以使用配置文件 (即 ARG 文件) 或通过注册表来执行此操作, 但是如果没有一些额外的工具或一些脚本来帮助它工作, 部署这些是很困难的。

^① 译者注: 只读属性只是不能用 `(vla-put-<xxx>)` 进行修改, 可以通过 `(vla-get-<xxx>)` 进行读取, 这里译者水平有限, 没有找到合适的表达方法。

一种解决方案是使用带有 Visual LISP 编码的 ActiveX 和 `AcadPreferences` 集合对象接口通过 AutoCAD 推送更新。下例显示了执行此操作的示例函数，并显示了如何在程序函数中使用它的示例。这可以通过使用 `acaddoc.lsp` (`s::startup`) 函数中的钩子来部署，你可以将其部署到所有客户端一次，并且能够从那时起轻松部署代码更改。下例中的代码可以从 `(s::startup)` 例程加载，并在客户端加载后自动启动。

```
(defun UpdateTemplatePath (pathname)
  (vla-put-TemplateDwgPath
    (vla-get-Files
      (vla-get-AcadPreferences
        (vlax-get-acad-object)))
    pathname))
(UpdateTemplatePath "X:\\acad\\configs\\templates")
```

你可能想稍微修饰一下这段代码，使其更加健壮和灵活。例如，你可以添加检查以验证现有路径设置在更改之前确实有误，从而在每个客户端的每个绘图会话开始时节省不必要的工作。

19.2 DatabasePreferences

`DatabasePreferences` 集合对象是一组仅适用于活动文档的首选项。如前所述，它们出现在 **选项** 对话框中，旁边有一个小的图形文件图标  来表示这一点。

下面是绘图会话中的集合导出，以显示其中包含的项目：

```
Command: (setq dbprefs (vla-get-Preferences activeDocument))
```

```
Command: (vlax-dump-object dbprefs t)

; IAcadDatabasePreferences: This object specifies the current AutoCAD
  ↳ drawing specific settings
; Property values:
; AllowLongSymbolNames = -1
; Application (RO) = #<VLA-OBJECT IAcadApplication 01877c20>
; ContourLinesPerSurface = 4
; DisplaySilhouette = 0
; Lineweight = -1
; LineWeightDisplay = 0
; MaxActiveViewports = 64
```

```

; ObjectSortByPlotting = -1
; ObjectSortByPSOutput = -1
; ObjectSortByRedraws = -1
; ObjectSortByRegens = -1
; ObjectSortBySelection = -1
; ObjectSortBySnap = -1
; OLELaunch = 0
; RenderSmoothness = 0.5
; SegmentPerPolyline = 8
; SolidFill = -1
; TextFrameDisplay = 0
; XRefEdit = -1
; XRefLayerVisibility = -1
; No methods

```

你可以继续向下深入每个属性，从而进一步地查询或修改其分配的值。在大多数情况下，它们将是 `:vlax-true` 或 `:vlax-false`（分别由 -1 和 0 表示）。例如：

```

Command: (vla-get-OLELaunch dbprefs)

:vlax-false

```

```

Command: (vla-get-ObjectSortByPlotting dbprefs)

:vlax-true

```

可以注意到这个对象没有方法。与 `AcadPreferences` 集合对象一样，可以使用与 Visual LISP 相同的方法访问和操作这些属性。例如，打开或关闭线宽显示：

```

(vla-put-LineWeightDisplay activedoc :vlax-true) ;; turns LWT on
(vla-put-LineWeightDisplay activedoc :vlax-false) ;; turns LWT off

```

19.3 重新加载配置文件

大多数 AutoCAD 开发人员都非常了解配置文件的工作原理以及它们的小怪癖。这里我先简单假设你不知道，主要是因为我想你可能有自己的生活，并将精力花在其他事情上，而不是浏览 AutoCAD 配置文件以弄清楚它们是如何工作的。

你可能已经知道可以将 `/p` 参数附加到 `acad.exe` 启动以指示它加载命名配置文件，例如：

```
$ >acad.exe /p myProfile
```

你还可以使用配置文件的文件名而不是配置文件名称^①，这将导致它将配置文件加载到注册表中，然后将其加载到绘图会话中。但是，如果当前用户已存在指定的配置文件，此特定功能将忽略 **ARG** 文件。要从 **ARG** 文件强制重新加载，你可以执行以下任一操作：

- 重命名注册表 **HKCU** 路径中的配置文件；
- 重命名 **ARG** 文件中的配置文件；
- 删除注册表 **HKCU** 路径中的配置文件。



不能简单地重命名 **ARG** 文件本身，因为配置文件名称是存储在其中的值，而不是从文件名本身派生的。

可能会出现下面一个典型的场景：

- 你向多个用户提供一个 **ARG** 文件以供导入；
- 用户导入 **ARG** 文件开始工作；
- 你更改源配置文件并导出新的 **ARG** 文件；
- 你将新的 **ARG** 文件提供给相同的用户；
- 用户导入新的 **ARG** 文件；
- 实际生效的还是原始配置文件设置。

这是因为新的 **ARG** 文件与原始文件具有相同的配置文件名称。当用户用新文件替换他们的 **ARG** 文件副本，然后使用 **/p** 参数启动 AutoCAD 时，AutoCAD 会读取 **ARG** 文件中的配置文件名称，并在注册表 **HKCU** 路径中搜索同名的现有配置文件。当它找到相同的名称时，它只是从注册表中加载配置文件并忽略 **ARG** 文件。

那么现在怎么办？嗯，由于可以通过 Visual LISP 访问和操作 **AcadPreferences** 集合对象，因此这个问题可以轻松地在幕后得到解决。查看下面的示例代码功能。

```
;;; Reloads a profile from an ARG file
;;; Replaces existing profile if defined
;;; Returns profile name if successful, otherwise returns nil
(defun Profile-Reload (name ARGname / bogus)
  (cond
    ( (and
      (Profile-Exists-p name)
      (findfile ARGname))
      (if (/= (strcase name) (strcase (vla-get-ActiveProfile (AcadProfiles))))
```

① 译者注：二者是 a profile filename 和 a profile name 的区别，直译成汉语很难区分，所以这里使用了不同的强调。

```

(Profile-Delete name)
(progn
  (setq bogus "bogus")
  (Profile-Rename name bogus)))
(Profile-Import name ARGname)
(vla-put-ActiveProfile (AcadProfiles) name)
(if bogus (Profile-Delete bogus))
name)
( (and
  (not (Profile-Exists-p name))
  (findfile ARGname))
  (Profile-Import name ARGname)
  (vla-put-ActiveProfile (AcadProfiles) name)
  name)
  ( (not (findfile ARGname))
  (princ (strcat "\nCannot locate ARG source: " ARGname))
  nil)))

;;; Renames an existing profile
;;; Returns new profile name if successful, otherwise returns nil

(defun Profile-Rename (from to / result)
  (if (Profile-Exists-p from)
    (if (not (Profile-Exists-p to))
      (cond
        ( (not
          (vl-catch-all-error-p
            (setq result
              (vl-catch-all-apply
                'vla-RenameProfile
                (list (AcadProfiles) from to))))))
          to))))); Return new name if successful!

;;; Deletes an existing profile
;;; Returns T if successful, otherwise returns nil

(defun Profile-Delete (strName / result)
  (if (Profile-Exists-p strName)
    (cond

```

```

    ( (not
      (vl-catch-all-error-p
        (setq result
          (vl-catch-all-apply
            'vla-DeleteProfile
            (list (AcadProfiles) strName))))))
    T ))); return T for success!

;;; Imports a profile from a given ARG file
;;; Returns profile name if successful, otherwise returns nil

(defun Profile-Import (argFile strName / result)
  (cond
    ( (findfile argFile)
      (cond
        ( (not
          (vl-catch-all-error-p
            (setq result
              (vl-catch-all-apply
                'vla-ImportProfile
                (list (AcadProfiles) strName argFile vlax-True))))))
          strName))))); return new profile name if successful!

;;; Determine if profile name is already defined (exists)
;;; Returns T or nil

(defun Profile-Exists-p (name)
  (get-item (AcadProfiles) name))

;;; Return Profiles collection object

(defun AcadProfiles ()
  (vla-get-profiles (vla-get-preferences (vlax-get-acad-object))))

```

最后但同样重要的是, 这是一个简单的函数, 用于返回所有已定义的配置文件名称的列表:

```

(defun Profiles-ListAll ( / hold)
  (vla-GetAllProfileNames (AcadProfiles) 'hold)

```

```
(if hold (vlax-SafeArray->List hold)))
```


菜单和工具条

正如 AutoCAD 绘图实体是 AutoCAD 对象模型的一部分一样，菜单和工具栏也是如此。菜单被组织成一个根 **MenuGroups** 集合，包含一个或多个 **MenuGroup** 对象。每个 **MenuGroup** 对象依次包含一组下拉菜单和一组工具栏。你可以在 **MenuGroup** 对象上使用 `(vla-load)` 方法添加至 **MenuGroups** 集合。这与使用 AutoCAD `MENULOAD` 命令执行的是相同的任务。

最酷的是你可以通过编程方式添加、修改和删除菜单项和配置。这使你能够从 Visual LISP 程序中完全控制地构建菜单。唯一没有从 **MenuGroups** 或 **MenuBar** 集合对象中暴露出来的是上古时的屏幕菜单^①。为此，须求助于标准 AutoLISP `(menucmd)` 函数来操作屏幕菜单，用 **AcadPreferences** 对象来控制显示（打开或关闭它）。

20.1 MenuBar 集合对象

MenuBar 集合对象包含了在 AutoCAD 会话中当前显示出的所有下拉菜单或弹出菜单项。它是 **AcadApplication** 对象的成员。如果加载的菜单中有一些与 AutoCAD 弹出菜单拼接在一起的弹出菜单组，则 **MenuBar** 集合对象将按集合中从左到右的顺序返回所有这些菜单。

```
Command: (setq acadApp (vlax-get-Acad-object))  
#<VLA-OBJECT IAcadApplication 00a8a730>
```

```
(setq mbar (vla-get-Menubar acadApp))
```

要访问单个弹出菜单，请使用 **Item** 方法，如下所示：

^① 译者注：确实是上古时期的东西了，从 AutoCAD® 2012 开始，这个东西已经被拿掉了，年轻的小朋友们不知道这是何物很正常。

```
Command: (setq popmenu1 (get-item mbar 0))
#<VLA-OBJECT IAcadMenuBar 00e8ae24>
```

要查看更多菜单栏信息，请按如下方式导出对象：

```
Command: (vlax-dump-object mbar T)
; IAcadMenuBar: A collection of PopupMenu objects representing the current
; ~> AutoCAD menu bar
; Property values:
; Application (RO) = #<VLA-OBJECT IAcadApplication 00a8a730>
; Count (RO) = 14
; Parent (RO) = #<VLA-OBJECT IAcadApplication 00a8a730>
; Methods supported:
; Item (1)
```

获取 MenuBar 成员

要访问 **MenuBar** 成员或检查集合中是否存在弹出菜单，你可以迭代集合。要检查特定的弹出菜单，请在 **MenuBar** 集合中搜索匹配的名称值，如下所示：

```
(defun PopMenu-MenuBar-p (name / mbar i found)
  (setq mbar (vla-get-Menubar (vlax-get-Acad-object)) i 0)
  (while (and (not found) (< i (1- (vla-get-count mbar))))
    (if (= (strcase name) (strcase (vla-get-name (get-item mbar i))))
      (setq found T)
      (setq i (1+ i)))
    (vlax-release-object mbar)
  found)
```

如果你知道弹出菜单的名称，则可以使用 **Item** 方法和字符串形式的 **Name** 属性直接访问它，如下所示：

```
(setq popmenu (get-item mbar "&File"))
```



注意，**Name** 属性包含作为名称字符串的一部分的助记字符 **&**。如果你尝试仅通过 **File** 的逻辑名称获取菜单，**(vla-item)** 方法将无法从 **MenuBar** 集合对象返回它。对于要通过字符串名称值使用 **Item** 方法获取的任何集合通常都是如此。

将 PopupMenu 插入 MenuBar 集合

可使用 **PopupMenu** 集合对象本身的 `(vla-InsertInMenuBar)` 方法将已加载菜单组的弹出菜单插入 **MenuBar** 集合。

```
(defun PopMenu-Insert (mgroup name loc / pmnu)
  (if (not (PopMenu-MenuBar-p name))
    (if (setq pmnu (vla-add mgroup name))
      (progn
        (vla-InsertInMenubar pmnu loc)
        (vlax-release-object pmnu)
        T)
      (princ (strcat "\nMenugroup or PopupMenu not loaded: " name)))
    (princ (strcat "\nPopupMenu already loaded: " name))))
```

将 PopupMenu 从 MenuBar 集合中删除

可使用 **PopupMenu** 集合对象本身的 `(vla-RemoveFromMenubar)` 方法将已命名的弹出菜单从 **MenuBar** 集合中删除。

20.2 MenuGroups 集合对象

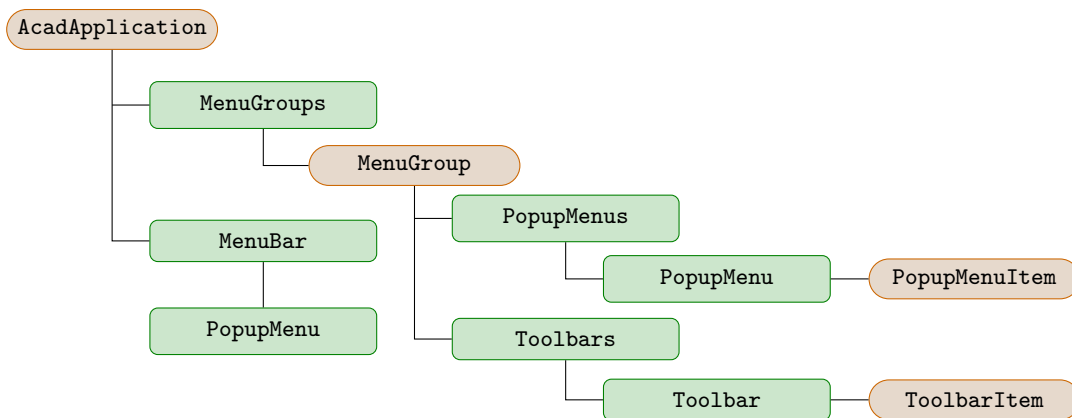


图 20.1 MenuGroups 集合对象

MenuGroups 集合对象(图 20.1)包含 AutoCAD 会话中找到的所有菜单组。每个菜单组都是

一个已加载的源菜单。通常，你会看到菜单组 **Acad**，如果已安装，你可能还会看到 **Express Tools** 的菜单组 **Express**。

```
(defun MenuGroups-ListAll ( / out)
  (vlax-for each (vla-get-MenuGroups acadapp)
    (setq out (cons (vla-get-name each) out)))
  out)
```

要将新的菜单组添加到 **MenuGroups** 集合，你可以使用 **Add** 或 **Load** 方法并指定适当的参数。要删除菜单组，你必须首先获取 **MenuGroup** 对象，然后调用其 **Delete** 方法。

20.3 MenuGroup 对象

MenuGroup 对象包含给定菜单组的所有弹出菜单和工具栏。它是 **MenuGroups** 集合对象的成员。要获取 **Acad** 菜单组，请使用以下命令：

```
Command: (setq mgroups (vla-get-MenuGroups acadapp))
#<VLA-OBJECT IAcadMenuGroups 01433208>
```

```
Command: (setq mg-acad (vla-item mgroups "Acad"))
#<VLA-OBJECT IAcadMenuGroup 00e9b38c>
```

20.4 PopupMenus 集合对象

PopupMenu 集合对象包含给定菜单组的所有弹出菜单或下拉菜单。要访问 **Acad** 菜单组的 **PopupMenu** 集合，请使用以下命令：

```
(setq pmnu (vla-get-Menus mg-acad))
```

对 **PopupMenu** 集合对象的导出显示了它的属性和方法。

20.5 PopupMenu 集合对象

PopupMenu 集合对象表示单个弹出菜单或下拉菜单。可以使用 **PopupMenu** 集合对象将其自身插入到 **MenuBar** 集合对象中，并访问其内部对象、属性和方法。

20.6 Toolbars 集合对象

Toolbars 集合对象是给定菜单组的所有工具栏的集合。使用 `(vla-get-Toolbars)` 方法从菜单组对象访问此集合的根。例如，要获取 Acad 菜单组的工具栏集合，请使用以下命令：

```
Command: (setq tbars (vla-get-Toolbars mg-acad))
#<VLA-OBJECT IAcadToolbars 00efa7c4>
```

你可能希望返回给定工具栏集合的所有工具栏名称的表。为此，只需迭代遍历 **Toolbars** 并返回它们各自的 **Name** 属性值表……

```
(progn
  (vlax-for each tbars
    (setq out (cons (vla-get-name each) out)))
  (if out (reverse out)))
```

20.7 Toolbar 集合对象

Toolbar 对象表示单个工具栏及其按钮。这些按钮包含为一个集合，可以使用 **Item** 方法对其进行迭代，而我将使用之前建议的 `(get-item)` 函数（详见第 43 页）。例如，要从 Acad 菜单组访问 **标注** 工具栏对象，请使用下列的命令：

```
Command: (setq tb1 (get-item tbars "Dimension"))
#<VLA-OBJECT IAcadToolbar 00eb6a7c>
```

```
Command: (vlax-dump-object tb1 T)

; IAcadToolbar: An AutoCAD toolbar
; Property values:
; Application (RO) = #<VLA-OBJECT IAcadApplication 01877c20>
; Count (RO) = 27
; DockStatus (RO) = 4
; FloatingRows = 1
; Height (RO) = AutoCAD: The toolbar is invisible. Please make it visible
; HelpString = "Dimension Toolbar\n "
; LargeButtons (RO) = 0
; left = 100
; Name = "Dimension"
```

```
; Parent (R0) = #<VLA-OBJECT IAcadToolbars 11115054>
; TagString (R0) = "ID_TbDimensi"
; top = 130
; Visible = 0
; Width (R0) = AutoCAD: The toolbar is invisible. Please make it visible
; Methods supported:
; AddSeparator (1)
; AddToolBarButton (5)
; Delete ()
; Dock (1)
; Float (3)
; Item (1)
```

下面的示例函数演示了如何从给定的菜单组对象中获取 **Toolbars** 集合，以及如何从指定的菜单组对象中按名称获取指定的工具栏对象。

```
(defun get-MenuGroups ()
  (vla-get-menugroups (vlax-get-acad-object)))

(defun get-MenuGroup (name)
  (if (menugroup name)
    (vla-item (get-MenuGroups) name)))

(defun get-Toolbars (mgroup / mg result)
  (if (setq mg (get-MenuGroup mgroup))
    (progn
      (setq result (vla-get-Toolbars mg))
      (vlax-release-object mg)))
  result)

(defun get-Toolbar (mgroup name / tbs result)
  (if (setq tbs (get-Toolbars mgroup))
    (progn
      (setq result (vla-item tbs name))
      (vlax-release-object tbs)))
  result)
```

接下来的示例函数将指定的工具栏停靠在左侧、右侧、顶部或底部。如果 **side** 参数不是

"LEFT"、"RIGHT"、"TOP" 或 "BOTTOM", 则默认为 "LEFT"。该参数不区分大小写。

```
(defun Toolbar-Dock (mgroup name side / tb loc)
  (cond
    ( (= (strcase side) "LEFT") (setq loc acToolbarDockLeft))
    ( (= (strcase side) "RIGHT") (setq loc acToolbarDockRight))
    ( (= (strcase side) "TOP") (setq loc acToolbarDockTop))
    ( (= (strcase side) "BOTTOM") (setq loc acToolbarDockBottom))
    ( T (setq loc acToolbarDockLeft)))
  (if (setq tb (get-Toolbar mgroup name))
    (progn
      (vla-Dock tb loc)
      (vlax-release-object tb))
    (princ (strcat "\nToolbar (" name ") not found."))))
```

以下示例函数将在指定位置浮动工具栏（距屏幕左上角偏移值为 x 和 y ）。参数是菜单组名称、工具栏名称、 y 坐标、 x 坐标和工具栏行布局。 x 和 y 坐标从屏幕的左上角起计。这是对话框窗体和工具栏的 Windows® 标准做法。此函数忽略隐藏的工具栏。


```
(defun Toolbar-Float (mgroup name top left rows)
  (if (setq tb (get-Toolbar mgroup name))
    (if (= (vla-get-Visible tb) :vlax-True)
      (progn
        (vla-Float tb top left rows)
        (vlax-release-object tb)
        1) ;; float and visible
      -1) ;; toolbar not visible
    0)) ;; toolbar not found
```

创建工具栏

我们来组装上面涵盖的一些代码并添加一些新内容来制作一个新的工具栏并为其分配几个按钮。在这种情况下,我们将向 AutoCAD 菜单组添加一个新工具栏并将其命名为 MyToolbar。第一个函数使用一些提供的属性值将按钮对象添加到工具栏对象。在此示例中,我对大图标和小图标位图属性使用相同的位图属性。

```
(defun Toolbar-AddButton
  (tb name macro bitmap1 tagstring helpstring / newButton index)
```

```
(setq index (vla-get-Count tb))
(cond
  ( (setq newButton
        (vla-AddToolbarButton tb
          (vlax-make-variant index vlax-vbInteger)
          name helpstring macro))
    (vla-put-TagString newButton tagstring)
    (vla-SetBitMaps newButton bitmap1 bitmap1)
    newButton)))
```

现在，我们将看到如何创建一个新工具栏并为其分配一个新按钮。此函数将创建一个名为 MyToolbar 的新工具栏，并向其添加一个调用 **LINE** 命令的按钮。从本书 CD 示例^①中加载示例文件  Toolbars.lsp 并在 AutoCAD 命令提示符下运行函数 **(Toolbar-Make)**。

```
(defun Toolbar-Make ( / tb)
  (cond
    ( (setq tb (vla-add (get-toolbars "acad") "MyToolbar"))
      (if (Toolbar-AddButton tb
        "Line" "\003\003\020\nLine"
        "ICON_16_LINE" "MyButton001" "Draws a line: LINE")
        (alert "I just added a button to my toolbar!")
        (alert "Uh oh! ..."))
      (vlax-release-object tb))))
```

可以通过其他方法（例如 **Add**、**Delete** 等）来进一步构建你的工具栏。可以操纵工具栏行配置、更改按钮顺序以及隐藏或显示工具栏。当将其与程序代码结合时，可以创建一些非常复杂的菜单管理功能。

20.8 功能区菜单

功能区菜单不会通过 **ActiveX** 公开给 Auto/Visual LISP。它们只公开给 **.NET** 接口。

^① 译者注：本书翻译时没有得到附件源代码，因此，这里也无法提供相应的文件。

外部接口

虽然 Visual LISP 在处理外部应用程序时比 VBA 要麻烦一些，但它确实有能力做一些非常强大的事情。在 Visual LISP 中使用 ObjectARX 应用程序非常普遍（例如 DOSLib），但这只是我们可以做更多事情的一小部分。一些示例可能是在 AutoCAD 和 Microsoft® Office 应用程序之间传递数据，以及使用 WSH 执行专门的桌面、文件、文件夹和网络任务。

21.1 Microsoft Excel

与新版本相比，旧版本的 Microsoft® Office 更多地依赖于外部 COM 接口的类型库。你仍然可以为某些版本的 Office 调用类型库接口，但很多时候你可能会发现这种努力是不值得的。你可以简单地实例化一个 Office 对象并执行迭代导出 ([vlax-dump-object](#)) 来探索接口。

```
Command: (setq objExcel (vlax-create-object "Excel.Application"))  
#<VLA-OBJECT _Application 11dd1f74>
```

```
Command: (vlax-dump-object objExcel t)
```

我不会费力粘贴上面导出语句的结果。它太长了，无论如何你都应该自己做一下。你会注意到 **Visible** 属性设置为 0，表示 **false**（或 **:vlax-false**）。如果执行以下代码行，它应该会导致 Excel 会话变得可见。

```
Command: (vla-put-Visible objExcel :vlax-true)
```

如果执行以下代码行，它将关闭/终止 Excel 会话。

```
Command: (vla-Quit objExcel)
```

我们打开 Excel 创建一个新工作簿并使用当前/活动绘图中的所有图层列表填充它，通过这来探索一些 Excel 接口：

```
(vl-load-com)
(setq acad (vlax-get-acad-object))
(setq activeDoc (vla-get-ActiveDocument acad))
(setq layers (vla-get-Layers activeDoc))

(setq excel (vlax-create-object "Excel.Application"))
(setq workbooks (vlax-get-property excel 'Workbooks))

(princ "\nSetting visible to True")
(vlax-put-property excel 'Visible :vlax-true)

(princ "\nGetting active workbook...")
(vlax-invoke-method workbooks 'Add)
(setq workbook (vlax-get-property excel 'ActiveWorkbook))

(princ "\nGetting active worksheet...")
(setq sheet (vlax-get-property excel 'ActiveSheet))

(princ "\nRename active worksheet...")
(vlax-put-property sheet 'Name "Layers")

(princ "\nGetting cells...")
(setq cells (vlax-get-property sheet 'Cells))

(princ "\nPrinting column headings...")
(vlax-put-property cells 'Item 1 1 "LayerName")
(vlax-put-property cells 'Item 1 2 "Color")
(vlax-put-property cells 'Item 1 3 "Linetype")

(princ "\nPopulating cells...")
(setq row 2)
(vlax-for layer layers
  (vlax-put-property cells 'Item row 1 (vla-get-Name layer))
  (vlax-put-property cells 'Item row 2 (itoa (vla-get-Color layer))))
```

```
(vlax-put-property cells 'Item row 3 (vla-get-Linetype layer))
(setq row (1+ row)))

(princ "\nSaving workbook...")
(vlax-invoke-method workbook 'Save)
(vlax-invoke-method workbooks 'Close)
(vlax-invoke-method excel 'Quit)
(foreach obj (list cells sheet workbook workbooks excel)
  (vlax-release-object obj))
(gc)
```

21.2 使用遗留的类型库

下面的示例代码显示了如何初始化 Excel 类型库（适用于 Excel 2000 或 Excel XP）以及如何从 Visual LISP 在 Excel 中创建一个新的 Excel 电子表格文件。请注意 Visual LISP 的一个特定方面：类型库接口。我为什么要这样说？好吧，虽然 VB 和 VBA 为你提供了称为 IntelliSense® 的自动完成功能，但 Visual LISP 没有。

在 VB 或 VBA 中工作并设置对组件库的引用时，它负责映射语法识别和弹出帮助字符串，但 Visual LISP 没有。即使从 Visual LISP 的角度来看你可能熟悉 Excel 2000，但不要假设在 Excel 10（也称为 Excel XP）中一切都是相同的。

本书没有用足够的篇幅来详细介绍这一点，但请放心，无论何时你打算将你的代码用于外部应用程序的更新版本，花时间调查更改是非常值得的。

```
(defun Excel-TypeLib-2000 ( / sysdrv officepath)
  (setq sysdrv (getenv "systemdrive"))
  (setq officepath (strcat sysdrv "\\program files\\microsoft office\\office"))
  (findfile (strcat officepath "\\excel9.olb")))
(defun Excel-TypeLib-XP ( / sysdrv officepath)
  (setq sysdrv (getenv "systemdrive"))
  (setq officepath (strcat sysdrv "\\program files\\microsoft offices\\office10"))
  (findfile (strcat officepath "\\excel.exe")))
(defun Excel-Load-TypeLib ( / tlbfile tlbver out)
  (cond
    ( (null msxl-xl24HourClock)
      (if (setq tlbfile (Excel-TypeLib-2000))
        (progn
```

```

(vlax-import-type-library
 :tlb-filename tlbfile
 :methods-prefix "msxl-"
 :properties-prefix "msxl-"
 :constants-prefix "msxl-")
(if msxl-xl24HourClock (setq out T))))
( T (setq out T)))
out)
(defun Excel-New-Spreadsheet (dmode / appsession result)
 (princ "\nCreating new Excel Spreadsheet file...")
 (cond
  ( (vl-catch-all-error-p
    (setq appsession
      (vl-catch-all-apply
        'vlax-create-object '("Excel.Application"))))
    (vl-exit-with-error
      (strcat "Error: " (vl-catch-all-error-message appsession))))
  ( T
    (princ "\nOpening Excel Spreadsheet file...")
    (cond
     ( (vl-catch-all-error-p
      (setq result
        (vl-catch-all-apply 'vlax-invoke-method
          (list (vlax-get-property appsession "Workbooks") "Add"))))
      (princ (strcat "\nError: " (vl-catch-all-error-message result))))
     ( T
      (if (= (strcase dmode) "SHOW")
        (vla-put-Visible appsession 1)
        (vla-put-Visible appsession 0))))))
  appsession)

```

特别注意上面函数中定义类型库接口的部分，分配给属性、方法和常量的字符串前缀是任意的，这实际上并没有很清楚地表示出来。在此示例中，我对所有三个都使用了相同的值，其他示例将使用唯一的前缀（例如 `msxp-`、`msxm-` 和 `msxc-`）来区分每种类型的接口对象。

```

(vlax-import-type-library
 :tlb-filename tlbfile
 :methods-prefix "msxl-"
 :properties-prefix "msxl-"

```

```
:constants-prefix "msxl-")
```

理论上，你也可以通过在上面的声明表达式中为每个属性使用空字符串 "" 来放弃分配前缀。但这样做会让多个应用程序类型库接口的使用变得困难，例如可能在同一 Visual LISP 代码中使用 Word 和 Excel。是的，你可以定义和使用完成工作所需的任意数量的类型库接口。如果可能的话，最好模块化 Visual LISP 代码以避免这种情况，并使每个类型库引用保持独立。保持一切有序和有条理，调试和测试将更容易管理。

你可以使用 Windows® 任务管理器并查看给定应用程序的进程列表来验证后台进程。例如，如果你打开 Excel 10（Office XP 的一部分）的会话并调用 `Excel.Application` 对象的 `Quit` 方法，你会期望在 Visual LISP 中释放该对象后进程将终止，但它通常来说不会。以下示例代码可用于在你的计算机上对此进行测试：

```
(defun excel-test ( / xlapp)
  (cond
    ( (setq xlapp (vlax-create-object "Excel.Application"))
      (vlax-put-property xlapp "Visible" T); show Excel
      (vlax-invoke-method xlapp "Quit"); close Excel
      (vlax-release-object xlapp); release object
      (gc)); force garbage collection
    ( T (princ "\nUnable to open Microsoft Excel.))))
```

将上述代码载入 VLIDE 编辑器窗口，载入 AutoCAD。打开 Windows® 任务管理器并选择进程列表选项卡。返回到 AutoCAD 并运行函数 `(excel-test)`，然后观察任务管理器进程列表中是否有 `Excel.exe` 出现在列表中。

正常的行为是 `Excel.exe` 进程会出现然后消失，你那里可能是这样。但在大多数情况下它不会消失。这里产生的问题是，如果尝试重新打开给定的电子表格，而 Excel 进程没有放开它。电子表格文件可能经常以只读模式打开，因为它认为其他人已经打开了该文件。

AUTODESK® 建议在释放此类对象后使用 `(gc)` 来强制终止，但是，`(gc)` 只是将对垃圾收集服务的调用放置在由 Windows® 资源服务管理的堆栈上。换句话说，无论你怎么尝试从 Visual LISP 终止会话，它通常可能根本不会终止。使用任务管理器手动终止进程时要小心，因为它通常会中断到 AutoCAD 的 RPC 通道，并导致对 Excel 的后续调用失败并出现错误。



我经常被问及如何在 Auto/Visual LISP 中使用 Microsoft® Access 和 SQL Server 或 Oracle。与其重新发明轮子，我更愿意将读者指向 JON FLEMING 的网站：<http://acad.fleming-group.com/index.html> 并建议下载和检查 ADOLISP 以了解如何执行此操作。

21.3 Windows ScriptControl

一种脚本语言可以利用另一种脚本语言来完成繁重的工作，特别是在被调用的语言能更有效完成这些工作的时候。我已经发布了几个这样的例子了，在那些调用 VBScript 的示例中，除了 VBScript 外，我通常会用到 KiXtart、PowerShell 或这三者之一的某些变体。在此示例中，我将从 WSH（运行 VBScript 的引擎）调用 `DateDiff()` 函数来计算某日期过去的时间（以天为单位）。

```
(defun daysold (d / sc cmd result)
  (setq sc (vlax-create-object "ScriptControl"))
  (vlax-put-property sc 'Language "vbscript")
  (setq cmd (strcat "DateDiff(\"d\", \"\" d \"\", Now)" ))
  (setq result (vlax-invoke-method sc 'Eval cmd))
  (vlax-release-object sc)
  (eval result))

(setq age (daysold "12/1/2010"))
```

这为其他脚本工具打开了一扇小而有用的门，可以帮助你扩大影响范围。你可以使用 ScriptControl 接口调用 Jscript、VBScript 甚至 ActivePerl。

21.4 Windows Scripting Host

Microsoft® 的 WSH 是一种功能强大但经常被忽视的免费服务，作为 Windows® 操作系统的一部分提供。基本上，WSH 是一个脚本引擎，可以从命令行界面或 GUI 界面运行脚本。命令行界面命令为 `CSCRIPT`，而 GUI 界面命令为 `WSCSCRIPT`。你可以通过键入 `CSCRIPT /?` 或 `WSCSCRIPT /?` 在 Windows® 命令外壳（DOS 窗口）中查看可用的运行时选项。

下面的示例演示了如何使用 WSH 外壳对象在收藏夹中创建快捷方式。你还可以访问当前用户和 AllUsers 组配置文件的桌面和开始菜单快捷方式存储库（取决于当前用户的本地权限）。

```
(defun AddFavoritesShortcut
  (target title / oWsh spfolders favorites shortcut)
  (cond
    ( (setq oWsh (vlax-create-object "WScript.Shell"))
      (setq spfolders (vlax-get-property oWsh "SpecialFolders")
        favorites (vla-item spfolders "Favorites")
```

```

        shortcut (vlax-invoke-method oWsh
                  "CreateShortcut"
                  (strcat favorites "\\\" title ".lnk"))))
    (vlax-put-property shortcut "TargetPath" target)
    (vlax-invoke-method shortcut "Save")
    (vlax-release-object oWsh)
    (gc);; forced garbage collection after object release
    (princ "\nShortcut created in Favorites..."))
    ( T (alert "Failed to obtain shortcut class object..."))))
(defun C:FAV ( / target name)
  (setq target (getstring "\nURL for Favorite shortcut: ")
        name (getstring t "\nName for Favorite: "))
  (AddFavoriteShortcut target name)
  (princ))

```

你可以调用 `WshShell` 对象的 `ExpandEnvironmentStrings` 方法来读取 Windows® 环境变量。你也可以使用 `(getenv)` 函数执行此操作，但需要注意一些差异。让我们试试看……

```

Command: (getenv "programfiles")
"C:\\Program Files"

```

```

(vl-load-com)
(setq objShell (vlax-create-object "Wscript.Shell"))
(vlax-invoke-method objShell
  'ExpandEnvironmentStrings "%programfiles%")

```

```
"C:\\Program Files"
```

看出区别了么？

这里是另一个不同的测试例子：

```

Command: (getenv "programfiles(X86)")
nil

```

```

(vlax-invoke-method objShell
  'ExpandEnvironmentStrings "%programfiles(x86)%")

```

```
"%programfiles(x86)%"
```

再次看出区别了吗？是的朋友！你一定会喜欢它。永远不要假设从两个不同的侧面戳同一只野兽会产生相同的结果。当 `(getenv)` 失败时，它返回 `nil`。当 `WshShell` 接口调用失败时，它只是吐回请求字符串。你仍然可以使用它来测试它是否失败，但关键是它们的行为不同。

21.5 KiXtart

我并不是建议你一开始就使用 KiXtart，尽管我个人碰巧喜欢它。但这是通过 COM 接口调用另一个工具来帮助扩展 LISP 代码的范围和灵活性的另一个例子。要完成这项工作，你需要从 <http://www.kixtart.org> 下载 KiXtart 并提取文件以获取 `kixtart.dll`。只需将它复制到你的硬盘驱动器并注册它（例如 `regsvr32 kixtart.dll`），你现在应该在 `HKCR` 下的 `KiXtart.Application` 注册表中有一个条目。

```
(setq kix (vlax-create-object "KiXtart.Application"))
(setq username (vlax-get kix "UserID")
      computer (vlax-get kix "WKSTA")
      userpriv (vlax-get kix "PRIV"))
```

请参阅 KiXtart 参考指南，了解你可以使用此接口调用的宏名称。宏是 KiXtart 引用全局变量的方式。

21.6 FileSystem 对象

`FileSystem` 对象，简称为 FSO，是一种功能强大的工具，用于通过 Windows® 操作系统连接和操作文件与文件夹。例如，我们可以使用它来迭代遍历所有驱动器映射并返回一个驱动器映射列表。

```
(defun ListDriveMappings (/ fso drive drives lst pth grp)
  (setq fso (vlax-create-object "Scripting.FileSystemObject"))
  (vlax-for drive (setq drives (vlax-get-property fso "Drives")))
  (setq ltr (strcat (vlax-get-property drive "DriveLetter") ":")
        pth (vlax-get-property drive "ShareName")
        grp (cons ltr pth)
        lst (cons grp lst))
  (vlax-release-object drives))
```



```
(vlax-Release-Object fso)
(reverse lst))
```

上述函数返回以点对表形式表示的驱动器映射列表，例如：

```
((("C:" . "")) ("D:" . "")) ("F:" . "\\server\share")...)
```

这对于构建列表和验证用户配置以支持你的应用程序驱动器映射需求很有用。我们还可以使用 `FileSystem` 对象查看特定 UNC 路径是否映射为驱动器号，如果是，则返回实际驱动器号。

```
(defun get-MappedShare (share / fso drives drive letter)
  (setq fso (vlax-create-object "Scripting.FileSystemObject"))
  (vlax-for drive (setq drives (vlax-get-property fso "Drives"))
    (if
      (=
        (strcase (vlax-get-property drive "ShareName"))
        (strcase share))
      (setq letter (strcat (vlax-get-property drive "DriveLetter") ":"))))
  (vlax-release-object drives)
  (vlax-release-object fso)
  letter)
```

使用上面的示例，你可以将 UNC 路径解析为实际驱动器号（如果它们已被当前用户映射）。语法如下：

```
(get-MappedShare "\\myserver\myshare");; might return a drive letter such as "F:"
```

`FileSystem` 对象提供许多其他方法和对各种对象属性的访问。例如，你可以复制、重命名和删除文件和文件夹。你甚至可以使用它将文件直接复制到命名端口，例如在使用绘图文件执行直接端口打印时：

```
(defun CopyFileToLPT1 (filename / file fso)
  (setq fso (vlax-create-object "Scripting.FileSystemObject"))
  (setq file (vlax-invoke-method fso "GetFile" filename))
  (vlax-invoke-method file "Copy" "LPT1")
  (vlax-release-object file)
  (vlax-release-object fso))
```

21.7 WMI and SWbemLocator

Microsoft® Windows® 管理规范 (Windows Management Instrumentation (WMI)) 服务是一个抽象层, 它提供与系统资源数据的编程接口。这包括硬件和软件, 还包括安全和安全上下文功能。要快速了解 WMI 的功能, 请打开 Windows® 2000 或 Windows® XP 中的计算机管理实用程序。你可以在该信息集合中找到的几乎所有内容都由 WMI 接口公开。WMI 可以通过任何 ActiveX 编程语言访问, 包括 Visual LISP。WMI 是 Microsoft® 的桌面管理任务组 (DMTF) CIM 规范参考模型的实现。

从 Visual LISP 中调用 WMI 最好通过使用基于 Web 的企业管理 (Web-Based Enterprise Management) 或 WBEM 脚本接口来处理。编程接口是 `WbemScripting.SwbemLocator`。如果你查看 MSDN 或 TechNet 上发布的从 VBScript 调用 WMI 的大多数示例, 你通常不会看到它是以这种方式完成的。但是如果你看的话, 那里有很多例子。建立 WMI 连接后, 你可以提交查询以获取一个集合 (安全数组), 该集合可以迭代以获取各个属性值。例如, `Win32_BIOS` 类公开有关计算机系统 BIOS 的信息:

Win32_BIOS

```
(setq wbem (vlax-create-object "WbemScripting.SWbemLocator"))
(setq strUser nil strPwd nil)
(setq wmiConn (vlax-invoke wbem 'ConnectServer nil nil strUser strPwd nil nil nil nil
↪ ))
(setq colItems (vlax-invoke wmiConn 'ExecQuery "Select * from Win32_BIOS"))
(vlax-for objItem colItems
  (princ (vlax-get objItem 'Name))
  (terpri)
  (princ (vlax-get objItem 'Version)))
```

将返回:

```
"PhoenixBIOS 4.0 Release 6.0"
"INTEL - 6040000"
```

Win32_OperatingSystem

```
(setq colItems (vlax-invoke wmiConn "ExecQuery" "Select * from Win32_OperatingSystem"
  ↪ ))
(vlax-for objItem colItems
  (princ
    (strcat
      (vlax-get objItem 'Manufacturer) ", "
      (vlax-get objItem 'Caption))))
```

```
"Microsoft Corporation, Microsoft Windows 7 Ultimate"
```

映射 WMI 属性名称

你可以传入带分隔符的列表，以便更轻松地解析属性名称，如以下示例所示：

```
(setq query "ExecQuery" "Select * from Win32_BIOS")
(setq colItems (vlax-invoke wmiConn query))
(setq fields "Caption,InstallDate,Manufacturer,ReleaseDate,SerialNumber,
  ↪ SMBIOSBIOSVersion,Version")
(vlax-for objItem colItems
  (foreach field (acet-str-to-list "," fields)
    (princ (vlax-get objItem field)))))
```

在第一部分中，你可能会注意到我将 `strUser` 和 `strPwd` 都设置为 `nil`。然后我将这些空值传递给 `ConnectServer` 方法调用。如果你使用安全连接，你可以指定一个明确的用户名和密码，但是，你应该记住，这样做将以明文形式传输信息，将其暴露给潜在的网络流量监控。

最常见的错误是 **拒绝访问** (0x80041003 或 2147749891) 和 **无效的命名空间** (0x8004100E 或 2147749902)。还有其他潜在的错误。这是另一种强烈建议使用 `(vl-catch-all-apply)` 的情况。

有关 `SwbemLocator` 接口和 `ConnectServer` 方法的更多信息，请查看 Microsoft® MSDN 文档：

- [http://msdn.microsoft.com/en-us/library/aa393720\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/aa393720(v=vs.85).aspx)

有关 WMI Win32 类的更多信息，请访问

- [http://msdn.microsoft.com/en-us/library/aa394554\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/aa394554(v=vs.85).aspx)

21.8 使用服务



调用 Windows® 服务时，通常使用带有显式 `<prog-id>` 标识符的 `GetObject` 方法 (`vlax-get-object`)。一个问题在于 Visual LISP 无法使用此接口调用某些服务，例如 LanmanServer 服务和上面提到的 WMI。

在本文发布时，唯一可用的解决方法是调用外部脚本或可执行文件，或提供“包装器”或中间组件来执行所需的操作，并以变体形式将结果返回给 Visual LISP®。你可以使用 Visual Basic 为几乎任何公开的服务开发包装器 DLL，并从 Visual LISP 或 VBA 调用该 DLL，然后执行需要执行的任何操作。



当然，你始终可以使用 AutoCAD 外壳命令来调用 Windows® 的 `SC.EXE` 命令或 Sysinternals 的 PsService 等实用程序。

在 Visual LISP 中使用 Visual Basic 的 DLL



我曾经考虑过完全删除这一部分，因为 VB6 是上古时期的历史文物了，但考虑到你仍然可以从 Visual Studio 2008/2010 编译 DLL 组件，所以仍然有一点相关性。

现在你已经了解了如何使用其他应用程序的接口，是时候考虑制作自己的自定义工具了。更准确地说，这涉及将你自己的服务开发为可由 Visual LISP（或其他使用 ActiveX 语言，如 VB 或 VBA）引用的组件。例如，你可以开发自己的 ActiveX 控件或 DLL，并从 Visual LISP 中使用它们。能够创建从 Visual LISP 内部访问其他资源的有效途径，这无疑为你开启了无限的潜力。

一个这样的例子可能是定义一组数据库接口例程，这些例程执行存储过程并将它们作为列表返回给你的 Visual LISP 应用程序。然后 DLL 可以处理 ADO 连接并自行执行命令和记录集管理。这使你不必担心在 Visual LISP 中执行此操作，尽管可以执行，但与使用更合适的语言（如 Visual Basic 或 Delphi）相比，这样做要繁琐得多。相反，使用  VLX 应用程序函数库允许你为其他 LISP 或 Visual LISP 应用程序提供类似的目的。嗯？是的，你可以将 DLL 功能包装在  VLX 应用程序中，这样即使你对 DLL 的调用也能保持私密和受保护。

例如，我们将创建一个 ActiveX DLL，它执行连接字符串并返回组合字符串结果的简单功能。这将涉及使用 Microsoft® Visual Basic 6.0 和一个新的 ActiveX DLL 工程（参见图 22.1）。

选择 **Open** 按钮后，将打开 Visual Basic 6.0 开发环境并显示默认代码窗口。将默认的工程名称从 Project1 更改为 vbStringClass，并将默认类模块的名称从 Class1 更改为 vbStrings。然后，在代码窗口中输入如图 22.2 所示的代码，定义三个不同的公共函数。公共函数是在加载 vbStringClass 类时可以向任何 ActiveX 使用者公开的函数。



图 22.1 Microsoft® Visual Basic 6.0 新建项目窗体

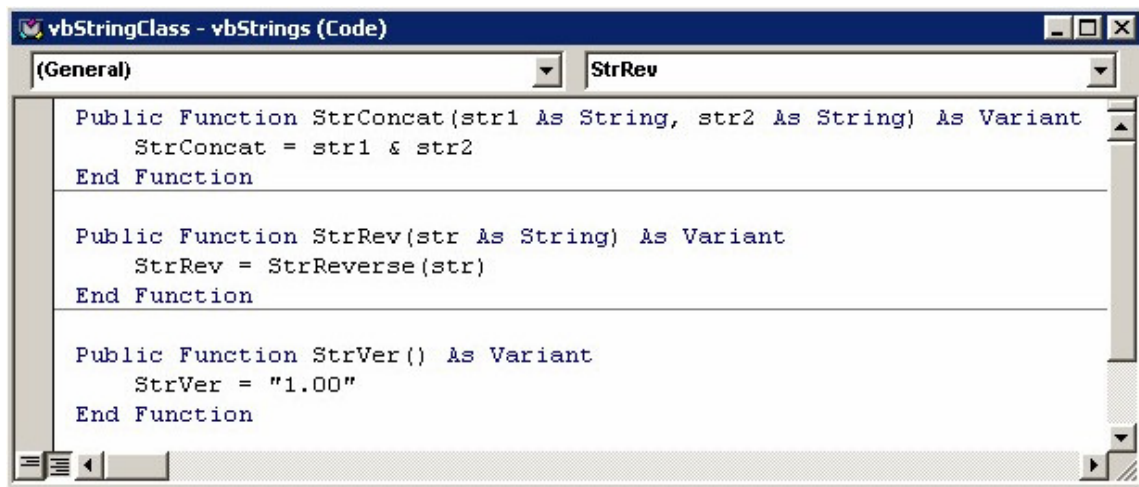


图 22.2 在类模块代码窗口中创建公共函数

在输入定义类函数的代码之后，先将类模块保存为 `vbStrCat.cls`，然后再将工程本身保存为 `vbStringClass.vbp`。选择菜单项 `File >> Make vbStringClass.DLL`。选择出现的窗体上的 `OK` 按钮，Visual Basic 会将你的类模块编译成 `DLL` 并将其注册到本地操作系统。此 `DLL` 现在可供任何其他程序使用，无论是 Visual LISP、VB.NET、C#.NET、F#、Python、C/C++ 还是其他任何程序。下一步是使用 Visual LISP 中的类型库接口加载此 `DLL` 并进行试用。



应该避免在程序代码中使用变量/符号名称 `acad`。某些第三方 `VLX` 应用程序将对此名称应用符号保护，这可能会导致你在代码中尝试使用该名称时遇到错误消息。

打开 Visual LISP® 编辑器，创建一个新的代码窗口并输入以下代码，定义这三个不同的 LISP 函数。

```
(vl-load-com)

(defun vbStrCat (string1 string2 / $acad vbstrcls out)
  (setq $acad (vlax-get-acad-object))
  (setq vbstrcls
    (vla-GetInterfaceObject $acad "vbStringClass.vbStrings"))
  (setq out (vlax-invoke-method vbstrcls "StrConcat" string1 string2))
  (vlax-release-object vbstrcls)
  (vlax-release-object $acad)
  out)

(defun vbStrRev (string / $acad vbstrcls out)
  (setq $acad (vlax-get-acad-object))
  (setq vbstrcls
    (vla-GetInterfaceObject $acad "vbStringClass.vbStrings"))
  (setq out (vlax-invoke-method vbstrcls "StrReverse" string))
  (vlax-release-object vbstrcls)
  (vlax-release-object $acad)
  out)

(defun vbStrVer ( / $acad vbstrcls out)
  (setq $acad (vlax-get-acad-object))
  (setq vbstrcls (vla-GetInterfaceObject $acad "vbStringClass.vbStrings"))
  (setq out (vlax-invoke-method vbstrcls "StrVer"))
  (vlax-release-object vbstrcls)
  (vlax-release-object $acad)
  out)
```

每个 `(defun)` 函数都使用 `AcadApplication` 对象方法 `GetInterfaceObject` 从操作系统中获取注册的 DLL，并在 Visual LISP 代码中公开类函数。请注意在返回结果值之前如何显式释放对象。



使用 Visual Basic 时，务必要小心定义函数（Function）而不是例程（Subroutine）。二者的区别在于函数可以返回值，而例程不能。此外，如果你未能在给定函数的末尾使用 `<Function>=<Result>` 返回，则对于你的 LISP 表达式，返回值将为 `nil`。你可能希望能生成 ActiveX 错误，但事实并非如此。此外，你必须将所有函数的返回数据类型定义为变体，以便将它们用于 Visual LISP。如果你重新运行一些其他数据类型，会导致你的 LISP 代码中出现 ActiveX 错误，因为它需要变体数据类型。

现在，将此 Visual LISP 代码加载到 AutoCAD 中，并通过在 AutoCAD 命令提示符下输入以下函数示例来测试它：

```
Command: (vbStrCat "THE " "DOG")
```

这将返回如下结果：

```
Initializing VBA System... "THE DOG"
```

```
Command: (vbStrRev "THE DOG BARKED")  
"DEKRAB GOD EHT"
```

```
Command: (vbStrVer)  
"1.0.0"
```

这是一个非常简单的示例，仅用于演示使用其他语言工具开发组件并在 Visual LISP 和其他语言环境中使用它们来完成工作。请注意，第一次引用这样的导入函数时；在返回结果之前，你会看到一条提示，上面写着 `Initializing VBA System ...`。这是因为 AutoCAD 使用 VBA 系统服务与涉及某些 ActiveX 功能的 ActiveX DLL 组件进行交互。在给定绘图会话中第一次调用后，将不会出现该消息，只会看到返回值。

注册 DLL

如果你计划使用此方法，则需要了解 DLL 组件的使用方式以及如何在给定计算机上注册它们。当你在 Visual Basic 中编译 ActiveX DLL 时，它会在你的本地计算机上处理这项工作；但是其他机器上的其他用户必须先以另一种方式注册 DLL，然后才能在他们的机

器上使用。Windows® 的 **REGSVR32** 命令用于在给定机器上手动注册 DLL 组件。语法为 **REGSVR32 <filename.dll>**，其中 **<filename.dll>** 是 DLL 所在位置的完整路径和文件名（它可以是本地的或网络共享的）。

例如，如果你要在另一台机器上部署我们上面创建的这个 DLL，并且你已将 **vbStringClass.DLL** 文件复制到 **G:\acadsupport\components\vbStringClass.DLL** 的网络文件夹中，将在每台客户端计算机上使用以下命令：

```
$ >regsvr32 G:\acadsupport\components\vbstringclass.dll
```

这可以通过任何命令提示符（在该本地计算机上），或通过 AutoCAD 调用外壳程序操作（同样，在该本地计算机上），或在 Visual LISP 中以编程方式使用一些代码来检查注册并处理注册 DLL（如果尚未在计算机上注册的话）。


重注册 DLL

每当你发布自定义 DLL 的更新版本时，它也会收到一个新的 GUID 标识符。这让其他应用程序知道他们正在使用你的 DLL 的特定版本。为了在另一台机器上更新 DLL，你必须首先使用 **REGSVR32 /U** 取消注册它，然后使用 **REGSVR32** 重新注册它以安装较新的版本并注册新的 GUID。例 14.1 中所示的函数 **(DLLUnRegister)** 可以在 Visual LISP 中用于取消注册一个已知的 DLL。然后你可以使用 **(DLLUnRegister)** 函数注册更新的版本。






这种类型的方法非常普遍，并构成了许多 Web 开发环境工作的基础。一个例子是 ASP 网络编程和使用带有 Visual Basic DLL 的 MTS。ASP 代码可以像我们使用 Visual LISP 一样调用 DLL，以便将复杂的处理任务交给专用组件，然后将结果返回给 ASP，从而在向用户呈现网页时使用。DLL 库同时向所有 ActiveX 语言提供它们的功能。所以当创建一个新的 DLL 时，请记住它通常可以被 Visual LISP、VBA、VB.NET、C#.NET、F#、Python、WSH 和许多其他语言使用。



最后，如果你确定继续创建自定义 DLL 组件，应该认真考虑使用 Visual Basic 6.0 的类生成器向导组件。这个方便的实用程序可帮助定义新类并为自己的 DLL 类开发对象模型。这为你以及你组织中使用其他支持 ActiveX 的语言的程序员开放了更大的可能性。

使用对话框表单

由于 Visual LISP 没有向 LISP 的 DCL 接口世界添加任何新内容，因此本章将重点介绍如何在  VLX 应用程序中使用 DCL。有很多方法可以处理来自对话框表单的回调。我不会向你传授任何特定的方法。此处显示的方法只是我自己使用 DCL 回调的习惯方式。如果你有其他偏好，请继续你的愉快方式，除非你现在想改变习惯。

23.1 引用 DCL 定义

在过去的 AutoLISP 时代，任何使用对话框窗体的应用程序都需要两个文件， LSP 文件和  DCL 文件。要想成功执行应用程序，二者都必须存在并且在运行时可供用户使用。使用新的 Visual LISP 应用程序功能，则可以将  LSP 和  DCL 源代码文件编译成单个  VLX 应用程序文件提供给用户。这不仅在一定程度上保护了源代码，而且还简化了部署和维护，尤其是在联网或分布式环境中。

例如，在 AutoLISP 中，一个  LSP 应用程序来访问  DCL 的源代码如下所示（同样，你的风格可能不同。这仅作为示例）：

```
(cond
  ( (setq dcfl (findfile "mydialog.dcl"))
    (setq dcid (load_dialog dcfl))
    (cond
      ( (new_dialog "myform" dcid)
        ;; ...do something with dialog callbacks here...
        (action_tile "accept" "(done_dialog 1)")
        (action_tile "cancel" "(done_dialog 0)")
        (start_dialog dlgstatus)
```

```
(unload_dialog dcid)))
( T (princ "\nUnable to locate DCL form file!") ))
```

你仍然可以使用这种“遗留”代码，只需进行非常小的更改，即可将其编译成 Visual LISP 的 VLX 应用程序文件。删除对文件位置的检查并假设它始终存在吧（因为，当你将其编译到 VLX 中时，它就会存在）。你只需要检查对话框自身的加载，这已经在原始代码中完成了。

```
(setq dcid (load_dialog "mydialog"))
(cond
  ( (new_dialog "myform" dcid)
    ...leave the rest as-is... ))
```

如你所见，它还稍微缩短了代码。你可以依此将从前的 AutoLISP 对话框应用程序移植到 Visual LISP 的 VLX 应用程序中，方法是参考第 13 章创建 Visual LISP 应用程序。

23.2 动态对话框交互

在基于对话框的应用程序中，一个非常常见的功能需求是让表单能够以动态的更改来响应用户交互，例如根据表单其他部分的选择更改图像或编辑框的值。可能最常见的是需要根据用户选择启用或禁用某些功能。

下面的示例将逐步介绍如何制作一个对话框表单，该表单根据另一部分的选择启用或禁用某些功能。首先，我们将展示示例对话框表单定义：

```
MyDialog.dcl

// save as MyDialog.dcl
myform : dialog {
  label = "My Dialog Form";
  : row {
    : boxed_radio_column {
      key = "viewpoint";
      label = "ViewPoint Options";
      : radio_button {key = "TOP"; label = "Top"; }
      : radio_button {key = "SIDE"; label = "Side"; }
      : radio_button {key = "FRONT"; label = "Front"; }
    }
    : boxed_column {
```

```

    label = "Other Options";
    : edit_box {key = "TOP2"; label = "Top Scale"; edit_width = 6; }
    : edit_box {key = "SIDE2"; label = "Side Scale"; edit_width = 6; }
    : edit_box {key = "FRONT2"; label = "Front Scale"; edit_width = 6; }
  }
}
ok_cancel;
}

```

现在，我们将看到如何实现在表单左侧选择某个单选按钮来启用相应的某一个编辑框。为此，我们将定义一些函数。第一个将使用单选列 viewpoint 的 (action_tile) 回调来处理对话框表单和回调，它将接收其集合中单选按钮选择的键名。然后我们可以使用该键名来执行条件操作，使用其他两个函数来操作其他控件。

```

;;; Save as MyDialog.lsp

(defun C:Myform ( / *error* dcid ok choice)
  (defun *error* (s)
    (princ (strcat "\nError: " s))
    (vl-bt)
    (princ))
  (setq dcid (load_dialog "mydialog"))
  (if (null $MYFORM1) (setq $MYFORM1 "TOP"))
  (cond
    ( (new_dialog "myform" dcid); form name is case sensitive!
      (set_tile "viewpoint" $MYFORM1)
      (change-form $MYFORM1); initialize tiles using default value
      (action_tile "viewpoint" "(setq choice (change-form $value))")
      (action_tile "accept" "(setq ok 1)(done_dialog)")
      (action_tile "cancel" "(setq ok 0)(done_dialog)")
      (start_dialog)
      (unload_dialog dcid)
      (if (= ok 1)
        (if choice (setq $MYFORM1 choice))))
    ( T (princ "\nUnable to load form from dialog definition.")))
  (princ))

```

```


;;; apply tile mode changes in response to given tile to enable

(defun change-form (val)
  (cond
    ( (= val "TOP")
      (mode-tiles '("TOP2") 0)
      (mode-tiles '("SIDE2" "FRONT2") 1))
    ( (= val "SIDE")
      (mode-tiles '("SIDE2") 0)
      (mode-tiles '("TOP2" "FRONT2") 1))
    ( (= val "FRONT")
      (mode-tiles '("FRONT2") 0)
      (mode-tiles '("TOP2" "SIDE2") 1)))
  val)

;;; apply tile mode to list of tilenames

(defun mode-tiles (tiles mode)
  (foreach tile tiles (mode_tile tile mode)))

```

现在只需按照第 13 章中的步骤将这两个源文件编译成一个  VLX 应用程序文件并将其加载到 AutoCAD 会话中。输入 **MYFORM** 运行命令并查看选择单选按钮时编辑框的反应。

如果得到正确编译和加载，对话框应该看起来如图 23.1 所示。

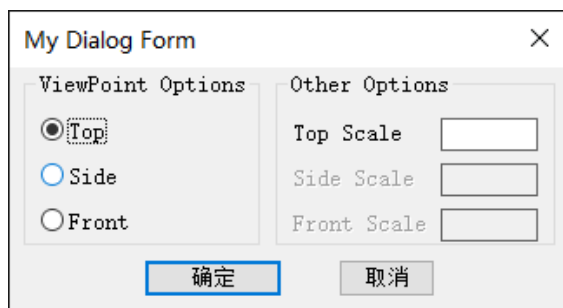











图 23.1 创建的对话框

23.3 从 DCL 回调控制图像

现在我们已经了解了如何通过回调值启用和禁用控件，让我们尝试更进一步并使用它来更改对话框表单图像控件中的幻灯片图像。为了演示这一点，你应该将示例代码幻灯片文件加载到一个公共文件夹中，并确保该文件夹位于你的默认搜索路径中。一旦我们编译并加载了  VLX 应用程序，它仍然需要找到用于图像平铺显示的  SLD 文件。不幸的是，Visual LISP 能够将  DCL、 DVB、 INI 和  LSP 文件编译到  VLX，却不提供将幻灯片文件编译到  VLX 的方法。

让我们采用上面定义的对话框形式并稍微修改它来添加单个图像控件：









```
// save as MyDialog2.dcl

myform : dialog {
  label = "My Dialog Form";
  : row {
    : boxed_column {
      label = "Preview";
      : image_button {
        key = "image";
        height = 10;
        aspect_ratio = 1.25;
        color = 0;
        fixed_height = true;
        fixed_width = true;
      }
    }
    : boxed_radio_column {
      key = "viewpoint";
      label = "ViewPoint Options";
      : radio_button {key = "TOP"; label = "Top"; }
      : radio_button {key = "SIDE"; label = "Side"; }
      : radio_button {key = "FRONT"; label = "Front"; }
    }
    : boxed_column {
      label = "Other Options";
      : edit_box {key = "TOP2"; label = "Top Scale"; edit_width = 6; }
    }
  }
}
```

```

        : edit_box {key = "SIDE2"; label = "Side Scale"; edit_width = 6; }
        : edit_box {key = "FRONT2"; label = "Front Scale"; edit_width = 6; }
    }
}
ok_cancel;
}





```

为了正确控制图像控件，我们应该定义一个特殊函数来负责查找图像文件和调整缩放比例以适应 DCL 配置。你可以从本书示例文件中的  MyDialog2.lsp 加载此示例^①。如果仔细观察，你会注意到此函数接受  SLD 或  SLB（幻灯片库）文件，从而可以将幻灯片捆绑到  SLB 中，并将总部署减少到仅需要  VLX 和  SLB 文件。

```

(defun slide-show
  (tile sld slb / w ky xc yc sldnam)
  (cond
    ( (or
      (and slb (findfile slb))
      (findfile sld))
      (setq
        xc (dimx_tile tile)
        yc (dimy_tile tile)
        (start_image tile)
        (fill_image 0 0 xc yc 0)
        (if slb
          (progn
            (setq slb (vl-filename-base slb))
            (setq sldnam (strcat slb "(" sld ")"))
            (setq sldnam sld))
          (slide_image 0 0 xc yc sldnam)
          (end_image))
        ( T (alert "Slide image file not found...")))))

```

假设你使用提供的幻灯片文件  top.sld、 side.sld 和  front.sld 并将它们放在当前默认搜索路径中的文件夹中，你应该能够编译和加载  mydialog2.vlx 并成功运行。

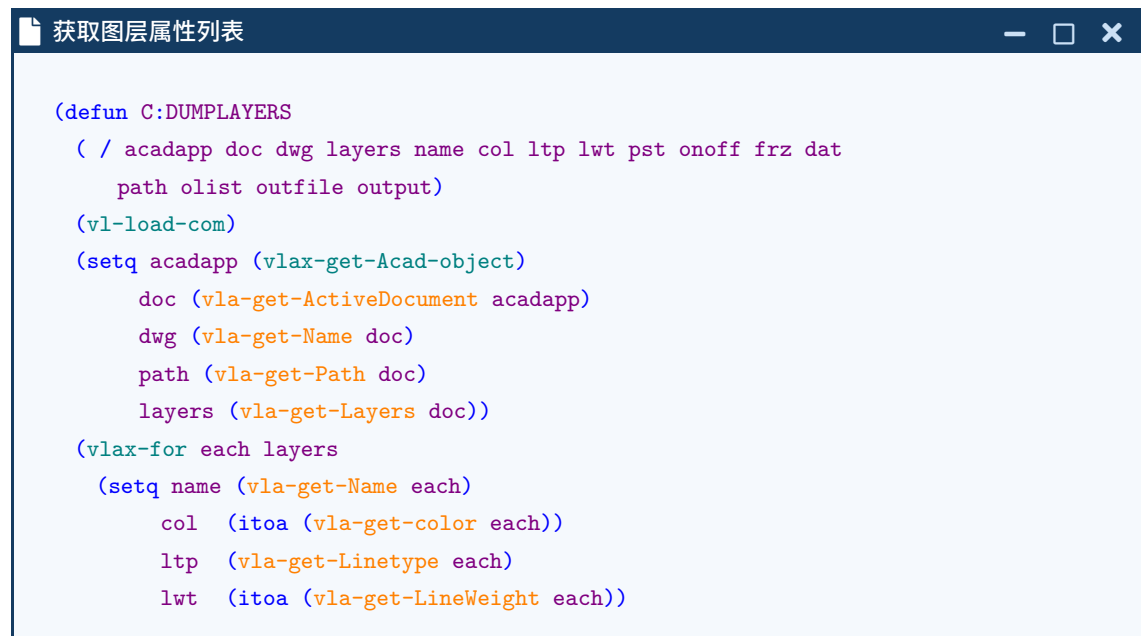
^① 译者注：本书翻译时没有得到附件源代码，因此，这里也无法提供相应的文件。

代码示例

本章将结合前几章内容提出解决常见任务的一些想法。你可能会发现其中一些有用，也许没有用。无论如何，提供它们只是为了演示如何使用 Visual LISP 来完成单独使用 AutoLISP 无法完成的事情。

24.1 示例 1 — 获取图层属性列表

这个例子将生成当前图形中所有图层的 HTML 报告的任务，包括它们的属性（颜色、线型等），并在完成后在 Web 浏览器中打开该报告。加载时，命令为 **DUMPLAYERS**。



```
(defun C:DUMPLAYERS
  ( / acadapp doc dwg layers name col ltp lwt pst onoff frz dat
    path olist outfile output)
  (vl-load-com)
  (setq acadapp (vlax-get-Acad-object)
        doc (vla-get-ActiveDocument acadapp)
        dwg (vla-get-Name doc)
        path (vla-get-Path doc)
        layers (vla-get-Layers doc))
  (vlax-for each layers
    (setq name (vla-get-Name each)
          col (itoa (vla-get-color each))
          ltp (vla-get-Linetype each)
          lwt (itoa (vla-get-LineWeight each))
```

```

    pst (vla-get-PlotStyleName each)
    onoff (if (= :vlax-True (vla-get-LayerOn each))
            "ON" "OFF")
    frz (if (= :vlax-True (vla-get-Freeze each))
          "FROZEN" "THAWED")
    dat (list name col ltp lwt pst onoff frz)
    olist (cons dat olist))
(vlax-release-object layers)
(vlax-release-object doc)
(vlax-release-object acadapp)
(cond
  ( olist
    (setq outfile (strcat (vl-filename-base dwg) ".htm"))
    (setq outfile (strcat path outfile))
    (cond
      ( (setq output (open outfile "w"))
        (write-line "<html>" output)
        (write-line "<head><title>" output)
        (write-line (strcat "Layer Dump: " dwg) output)
        (write-line "</title></head><body>" output)
        (write-line (strcat "<b>Drawing: " dwg "</b><br>" ) output)
        (write-line "<table border=1>" output)
        (foreach layset olist
          (write-line "<tr>" output)
          (foreach prop layset
            (write-line (strcat "<td>" prop "</td>" ) output))
          (write-line "</tr>" output))
        (write-line "</table></body></html>" output)
        (close output)
        (setq output nil)
        (princ "\nReport finished! Opening in browser...")
        (vl-cmdf "_.browser" outfile))
      ( T (princ "\nUnable to open output file.))))))
    ( T (princ "\nUnable to get layer table information.))))))

```

24.2 示例 2 —设置所有实体属性为 ByLayer

这个例子将当前工作空间中的所有实体的有关颜色、线型和线宽等属性设置为 ByLayer。

```
设置所有实体属性为 ByLayer

(defun C:BYLAYER
  (/ acadapp doc ssall i obj)
  (vl-load-com)
  (setq acadapp (vlax-get-acad-object)
    doc (vla-get-activedocument acadapp))
  (vla-startundomark acadapp)
  (vla-zoomextents acadapp)
  (cond
    ((setq ssall (ssget "x")); get all entities
      (setq i 0)
      (repeat (sslength ssall)
        (setq obj (vlax-ename->vla-object (ssname ssall i)))
        (vlax-put-property obj "Color" acByLayer)
        (vlax-put-property obj "Linetype" "ByLayer")
        (vlax-put-property obj "Lineweight" acLnWtByLwDefault)
        (vlax-release-object obj)
        (setq i (1+ i))))))
  (vla-endundomark acadapp)
  (princ "\nFinished processing all entities.")
  (princ))
```

24.3 示例 3 —对所有打开的图形文件进行清理、核查并保存

这个例子将遍历所有打开的图形文件，并对每个图形文件进行清理、核查并保存。

```
对所有打开的图形文件进行清理、核查并保存

(defun C:DOALL (/ $acad docs dnum this)
  (vl-load-com)
  (setq $acad (vlax-get-acad-object)
    docs (vla-get-documents $acad)
```

```
      this (vla-get-activedocument $acad)
      dnum (vla-get-count docs))
(vlax-for each docs
  (vla-purgeall each)
  (vla-auditinfo each T)
  (vla-save each))
(vla-get-activedocument this)
(vlax-release-object docs)
(vlax-release-object this)
(vlax-release-object $acad)
(princ (strcat "\nProcessed " (itoa dnum) " drawings."))
(princ))
```

24.4 示例 4 —对所有打开的图形文件缩放全景并保存

这个示例将迭代遍历每个打开的文档、对当前（活动）空间和选项卡缩放到全景并保存文档。最后，它在完成后返回到最初文档。



```
(defun C:ZOOMALL ( / $acad docs dnum this)
  (vl-load-com)
  (setq $acad (vlax-get-acad-object)
        docs (vla-get-documents $acad)
        this (vla-get-activedocument $acad)
        dnum (vla-get-count docs))
  (vlax-for each docs
    (vla-put-ActiveDocument each)
    (vla-ZoomExtents $acad)
    (vla-save each))
  (vla-put-activedocument this)
  (vlax-release-object docs)
  (vlax-release-object this)
  (vlax-release-object $acad)
  (princ (strcat "\nProcessed " (itoa dnum) " drawings."))
  (princ))
```

24.5 示例 5 —创建表格

创建表格

```
(if (setq pt (getpoint "\nSelect table basepoint: "))
  (progn
    (setq
      acad (vlax-get-acad-object)
      doc (vla-get-ActiveDocument acad)
      ms (vla-get-ModelSpace doc))
    (setq objTable
      (vla-addTable ms
        (vlax-3d-point pt)
        3 ; rows
        3 ; columns
        0.75 ; row height
        2.0)) ; column width
    (vla-setText objTable 0 0 "Heading")
    (vla-setText objTable 1 0 "Row 1, Col 0")
    (vla-setText objTable 1 1 "Row 1, Col 1")
    (vla-setText objTable 1 2 "Row 1, Col 2")
    (vla-setText objTable 2 0 "Row 2, Col 0")
    (vla-setText objTable 2 1 "Row 2, Col 1")
    (vla-setText objTable 2 2 "Row 2, Col 2")
    (vla-put-RegenerateTableSuppressed objTable :vlax-true)
    (vla-settextstyle objTable actitlerow "Standard")
    (vla-settextstyle objTable aheaderrow "Standard")
    (vla-settextstyle objTable acdatarow "Standard")
    (vla-put-TitleSuppressed objTable :vlax-false)
    (vla-put-headerSuppressed objTable :vlax-false)
    (vla-put-horzcellmargin objTable 0.4)
    (vla-put-vertcellmargin objTable 0.4)
    (vla-put-RegenerateTableSuppressed objTable :vlax-false)
    (vlax-release-object objTable)
    (vlax-release-object ms)))
```

如果一切顺利，上面的代码应该会生成一个如图 24.1 所示的表格：

Heading		
Row 1, Col 0	Row 1, Col 1	Row 1, Col 2
Row 2, Col 0	Row 2, Col 1	Row 2, Col 2

图 24.1 创建的表格

AutoCAD® 2011 的变化

我并无意无止境地就 AutoCAD® 2004 和 AutoCAD® 2011 每一处不同点做出比较，这事还是留给 AUTODESK® 和其他人来做吧。

25.1 一般性变化

大多数对应用程序的改变旨在提高对 Windows® Vista 和 Windows® 7 的兼容性，这有利于 IT 专业人员，并使他们的产品整体上更容易进行故障排除和技术支持。

25.2 Visual Lisp 在 AutoCAD® 2011 中的变化

我想用来描述 2004 年到 2011 年间 Visual LISP® 变化的最恰当一个比喻是：它就像是被当头浇了一桶冰水，涂上那种一元店里的口红，然后被拖回去参加另一个派对。这里那里有一些小的调整和微调，但总体来说也就是“呵呵”了。可怜的 LISP API 孤儿在寒冷的屋外，光着脚，看着衣着光鲜的 .NET 小孩子在温暖的房子里吃新鲜的美味佳肴。如果你仔细听，你会听到他们笑着说这个无家可归的 LISP 小孩子在冰冷的窗玻璃上呼吸看起来多么有趣。我仍然可以听到小提琴演奏的声音。

结论

虽然本书涵盖了有关使用 Visual LISP 的大量信息，但并不能涵盖所有内容。Visual LISP 有很多潜力，并且为 LISP 开发人员提供了很多强大的工具。我希望 AUTODESK® 能投入一些资源来改进它，使其与当前的开发工具保持同步。至少，修复一些不完整的功能和恼人的怪癖，但我认为 AUTODESK® 正在让 LISP 未老先衰^①，转而支持 VBA 和 ObjectARX。AutoCAD 编程的未来似乎是针对 .NET 和 Java，而不是 LISP。在我看来，在 AutoCAD 中忽略 LISP 是不幸的，也是一个大错误。

随着官方宣布 VBA 将在 AutoCAD® 2011 之后被弃用，或者至少降级为外部附加组件，LISP 引擎仍然存在这一事实不仅说明了它已变得多么不可或缺，而且还说明它对如此多的用户和合作伙伴来说是多么重要。

大多数可改进的功能只需要极少的时间和预算投资，并且会产生更强大的开发工具。一些改进可能是修复项目管理的工具、更好的编译控制、修复对话框的不一致、添加“智能”补全、引用外部对象和组件的简化功能，以及制作独立版本等等。

希望此处提供的信息和示例会给你一些额外的动力，让你进一步探索 Visual LISP®，从而成为更好的软件开发人员。如果没有，它也可以拿来盖泡面^②。无论如何，我希望你觉得这本书有用和有帮助。

编码快乐！

— Dave

^① 译者注：关于这个翻译在引言中已经解释了。

^② 译者注：作者原文是“it makes a great coffee cup stand”，直译“这是个不错的咖啡杯垫”，考虑我国的国情，我用了这个译法。

VLAX 枚举

常量符号	值	常量符号	值
<code>:vlax-False</code>	<code>:vlax-False</code>	<code>vlax-vbKatakana</code>	16
<code>:vlax-Null</code>	<code>:vlax-Null</code>	<code>vlax-vbLong</code>	3
<code>:vlax-True</code>	<code>:vlax-True</code>	<code>vlax-vbLowerCase</code>	2
<code>vlax-vbAbort</code>	3	<code>vlax-vbNarrow</code>	8
<code>vlax-vbAbortRetryIgnore</code>	2	<code>vlax-vbNo</code>	7
<code>vlax-vbApplicationModal</code>	0	<code>vlax-vbNormal</code>	0
<code>vlax-vbArchive</code>	32	<code>vlax-vbNull</code>	1
<code>vlax-vbArray</code>	8192	<code>vlax-vbObject</code>	9
<code>vlax-vbBoolean</code>	11	<code>vlax-vbOK</code>	1
<code>vlax-vbCancel</code>	2	<code>vlax-vbOKCancel</code>	1
<code>vlax-vbCritical</code>	16	<code>vlax-vbOKOnly</code>	0
<code>vlax-vbCurrency</code>	6	<code>vlax-vbProperCase</code>	3
<code>vlax-vbDataObject</code>	13	<code>vlax-vbQuestion</code>	32
<code>vlax-vbDate</code>	7	<code>vlax-vbReadOnly</code>	1
<code>vlax-vbDefaultButton1</code>	0	<code>vlax-vbRetry</code>	4

常量符号	值	常量符号	值
<code>vlax-vbDefaultButton2</code>	256	<code>vlax-vbRetryCancel</code>	5
<code>vlax-vbDefaultButton3</code>	512	<code>vlax-vbSingle</code>	4
<code>vlax-vbDirectory</code>	16	<code>vlax-vbString</code>	8
<code>vlax-vbDouble</code>	5	<code>vlax-vbSystem</code>	4
<code>vlax-vbEmpty</code>	0	<code>vlax-vbSystemModal</code>	4096
<code>vlax-vbError</code>	10	<code>vlax-vbUpperCase</code>	1
<code>vlax-vbExclamation</code>	48	<code>vlax-vbVariant</code>	12
<code>vlax-vbHidden</code>	2	<code>vlax-vbVolume</code>	8
<code>vlax-vbHiragana</code>	32	<code>vlax-vbWide</code>	4
<code>vlax-vbIgnore</code>	5	<code>vlax-vbYes</code>	6
<code>vlax-vbInformation</code>	64	<code>vlax-vbYesNo</code>	4
<code>vlax-vbInteger</code>	2	<code>vlax-vbYesNoCancel</code>	3



没有 `vlax-vbDecimal` 或 `vlax-vbNothing` 枚举。此外，`vlax-vbNull` 也不能很好地转换为 ActiveX 使用方（例如 VBA）作为真正的 Null 对象或值。请注意 VBA 6、VB 6 和 .NET 环境之间的数据类型变化。

VLA 函数名称

vla-Activate
vla-Add
vla-Add3DFace
vla-Add3DMesh
vla-Add3DPoly
vla-AddArc
vla-AddAttribute
vla-AddBox
vla-AddCircle
vla-AddCone
vla-AddCustomInfo
vla-AddCustomObject
vla-AddCylinder
vla-AddDim3PointAngular
vla-AddDimAligned
vla-AddDimAngular
vla-AddDimArc
vla-AddDimDiametric
vla-AddDimOrdinate
vla-AddDimRadial
vla-AddDimRadialLarge
vla-AddDimRotated

vla-AddEllipse
vla-AddEllipticalCone
vla-AddEllipticalCylinder
vla-AddExtrudedSolid
vla-AddExtrudedSolidAlongPath
vla-AddFitPoint
vla-AddHatch
vla-AddItems
vla-AddLeader
vla-AddLeaderLine
vla-AddLeaderLineEx
vla-AddLightWeightPolyline
vla-AddLine
vla-AddMenuItem
vla-AddMInsertBlock
vla-AddMLeader
vla-AddMLine
vla-AddMText
vla-AddObject
vla-AddPoint
vla-AddPolyfaceMesh
vla-AddPolyline
vla-AddPViewport
vla-AddRaster
vla-AddRay
vla-AddRegion
vla-AddRevolvedSolid
vla-AddSection
vla-AddSeparator
vla-AddShape
vla-AddSolid
vla-AddSphere
vla-AddSpline
vla-AddSubMenu

vla-AddTable
vla-AddText
vla-AddTolerance
vla-AddToolBarButton
vla-AddTorus
vla-AddTrace
vla-AddVertex
vla-AddWedge
vla-AddXline
vla-AddXRecord
vla-AngleFromXAxis
vla-AngleToReal
vla-AngleToString
vla-AppendInnerLoop
vla-AppendItems
vla-AppendOuterLoop
vla-AppendVertex
vla-ArrayPolar
vla-ArrayRectangular
vla-AttachExternalReference
vla-AttachToolBarToFlyout
vla-AuditInfo
vla-Bind
vla-Block
vla-Boolean
vla-CheckInterference
vla-Clear
vla-ClearSubSelection
vla-ClearTableStyleOverrides
vla-ClipBoundary
vla-Close
vla-ConvertToAnonymousBlock
vla-ConvertToStaticBlock
vla-Copy

vla-CopyFrom
vla-CopyObjects
vla-CopyProfile
vla-CreateCellStyle
vla-CreateCellStyleFromStyle
vla-CreateContent
vla-CreateEntry
vla-CreateJog
vla-CreateTypedArray
vla-Delete
vla-DeleteCellContent
vla-DeleteCellStyle
vla-DeleteColumns
vla-DeleteConfiguration
vla-DeleteContent
vla-DeleteFitPoint
vla-DeleteProfile
vla-DeleteRows
vla-Detach
vla-Display
vla-DisplayPlotPreview
vla-DistanceToReal
vla-Dock
vla-ElevateOrder
vla-EnableMergeAll
vla-EndUndoMark
vla-Erase
vla-Eval
vla-Evaluate
vla-Explode
vla-Export
vla-ExportProfile
vla-FieldCode
vla-Float

vla-FormatValue
vla-GenerateLayout
vla-GenerateSectionGeometry
vla-GenerateUsageData
vla-get-Action
vla-get-Active
vla-get-ActiveDimStyle
vla-get-ActiveDocument
vla-get-ActiveLayer
vla-get-ActiveLayout
vla-get-ActiveLinetype
vla-get-ActiveMaterial
vla-get-ActiveProfile
vla-get-ActivePViewport
vla-get-ActiveSelectionSet
vla-get-ActiveSpace
vla-get-ActiveTextStyle
vla-get-ActiveUCS
vla-get-ActiveViewport
vla-get-ADCInsertUnitsDefaultSource
vla-get-ADCInsertUnitsDefaultTarget
vla-get-AdjustForBackground
vla-get-AffectsGraphics
vla-get-Algorithm
vla-get-Alignment
vla-get-AlignmentPointAcquisition
vla-get-AlignSpace
vla-get-AllowedValues
vla-get-AllowLongSymbolNames
vla-get-AllowManualHeights
vla-get-AllowManualPositions
vla-get-AltFontFile
vla-get-AltRoundDistance
vla-get-AltSubUnitsFactor

vla-get-AltSubUnitsSuffix
vla-get-AltSuppressLeadingZeros
vla-get-AltSuppressTrailingZeros
vla-get-AltSuppressZeroFeet
vla-get-AltSuppressZeroInches
vla-get-AltTabletMenuFile
vla-get-AltTextPrefix
vla-get-AltTextSuffix
vla-get-AltTolerancePrecision
vla-get-AltToleranceSuppressLeadingZeros
vla-get-AltToleranceSuppressTrailingZeros
vla-get-AltToleranceSuppressZeroFeet
vla-get-AltToleranceSuppressZeroInches
vla-get-AltUnits
vla-get-AltUnitsFormat
vla-get-AltUnitsPrecision
vla-get-AltUnitsScale
vla-get-angle
vla-get-AngleFormat
vla-get-AngleVertex
vla-get-Annotation
vla-get-Annotative
vla-get-Application
vla-get-ArcEndParam
vla-get-ArcLength
vla-get-ArcPoint
vla-get-ArcSmoothness
vla-get-ArcStartParam
vla-get-Area
vla-get-Arrowhead1Block
vla-get-Arrowhead1Type
vla-get-Arrowhead2Block
vla-get-Arrowhead2Type
vla-get-ArrowheadBlock

vla-get-ArrowheadSize
vla-get-ArrowheadType
vla-get-ArrowSize
vla-get-ArrowSymbol
vla-get-AssociativeHatch
vla-get-AttachmentPoint
vla-get-Author
vla-get-AutoAudit
vla-get-AutomaticPlotLog
vla-get-AutoSaveInterval
vla-get-AutoSavePath
vla-get-AutoSnapAperture
vla-get-AutoSnapApertureSize
vla-get-AutoSnapMagnet
vla-get-AutoSnapMarker
vla-get-AutoSnapMarkerColor
vla-get-AutoSnapMarkerSize
vla-get-AutoSnapToolTip
vla-get-AutoTrackingVecColor
vla-get-AutoTrackTooltip
vla-get-AxisDirection
vla-get-AxisPosition
vla-get-BackgroundColor
vla-get-BackgroundFill
vla-get-BackgroundLinesColor
vla-get-BackgroundLinesHiddenLine
vla-get-BackgroundLinesLayer
vla-get-BackgroundLinesLinetype
vla-get-BackgroundLinesLinetypeScale
vla-get-BackgroundLinesLineweight
vla-get-BackgroundLinesPlotStyleName
vla-get-BackgroundLinesVisible
vla-get-Backward
vla-get-Bank

vla-get-BasePoint
vla-get-BaseRadius
vla-get-BatchPlotProgress
vla-get-BeepOnError
vla-get-BigFontFile
vla-get-BitFlags
vla-get-Block
vla-get-BlockColor
vla-get-BlockConnectionType
vla-get-BlockRotation
vla-get-Blocks
vla-get-BlockScale
vla-get-BlockScaling
vla-get-Blue
vla-get-BookName
vla-get-BottomHeight
vla-get-BreaksEnabled
vla-get-BreakSize
vla-get-BreakSpacing
vla-get-Brightness
vla-get-CanonicalMediaName
vla-get-Caption
vla-get-CategoryName
vla-get-Center
vla-get-CenterMarkSize
vla-get-CenterPlot
vla-get-CenterPoint
vla-get-CenterType
vla-get-Centroid
vla-get-Check
vla-get-ChordPoint
vla-get-Circumference
vla-get-Clipped
vla-get-ClippingEnabled

vla-get-Closed
vla-get-Closed2
vla-get-Color
vla-get-ColorBookPath
vla-get-ColorIndex
vla-get-ColorMethod
vla-get-ColorName
vla-get-Columns
vla-get-ColumnSpacing
vla-get-CommandDisplayName
vla-get-Comment
vla-get-Comments
vla-get-ConfigFile
vla-get-ConfigName
vla-get-Constant
vla-get-ConstantWidth
vla-get-Constrain
vla-get-ContentBlockName
vla-get-ContentBlockType
vla-get-ContentType
vla-get-ContinuousPlotLog
vla-get-ContourLinesPerSurface
vla-get-Contrast
vla-get-ControlPoints
vla-get-Coordinate
vla-get-Coordinates
vla-get-Count
vla-get-CreaseLevel
vla-get-CreaseType
vla-get-CreateBackup
vla-get-CurrentSectionType
vla-get-CursorSize
vla-get-CurveTangencyLinesColor
vla-get-CurveTangencyLinesLayer

vla-get-CurveTangencyLinesLinetype
vla-get-CurveTangencyLinesLinetypeScale
vla-get-CurveTangencyLinesLineweight
vla-get-CurveTangencyLinesPlotStyleName
vla-get-CurveTangencyLinesVisible
vla-get-CustomDictionary
vla-get-CustomIconPath
vla-get-CustomScale
vla-get-CvHullDisplay
vla-get-Database
vla-get-DecimalSeparator
vla-get-DefaultInternetURL
vla-get-DefaultOutputDevice
vla-get-DefaultPlotStyleForLayer
vla-get-DefaultPlotStyleForObjects
vla-get-DefaultPlotStyleTable
vla-get-DefaultPlotToFilePath
vla-get-Degree
vla-get-Degree2
vla-get-Delta
vla-get-DemandLoadARXApp
vla-get-Description
vla-get-DestinationBlock
vla-get-DestinationFile
vla-get-Diameter
vla-get-Dictionaries
vla-get-DimConstrDesc
vla-get-DimConstrExpression
vla-get-DimConstrForm
vla-get-DimConstrName
vla-get-DimConstrReference
vla-get-DimConstrValue
vla-get-DimensionLineColor
vla-get-DimensionLineExtend

vla-get-DimensionLinetype
vla-get-DimensionLineWeight
vla-get-DimLine1Suppress
vla-get-DimLine2Suppress
vla-get-DimLineInside
vla-get-DimLineSuppress
vla-get-DimStyles
vla-get-DimTxtDirection
vla-get-Direction
vla-get-DirectionVector
vla-get-Display
vla-get-DisplayGrips
vla-get-DisplayGripsWithinBlocks
vla-get-DisplayLayoutTabs
vla-get-DisplayLocked
vla-get-DisplayOLEScale
vla-get-DisplayScreenMenu
vla-get-DisplayScrollBars
vla-get-DisplaySilhouette
vla-get-DockedVisibleLines
vla-get-DockStatus
vla-get-Document
vla-get-Documents
vla-get-DogLegged
vla-get-DoglegLength
vla-get-Drafting
vla-get-DrawingDirection
vla-get-DrawLeaderOrderType
vla-get-DrawMLeaderOrderType
vla-get-DriversPath
vla-get-DWFFormat
vla-get-EdgeExtensionDistances
vla-get-EffectiveName
vla-get-Elevation

vla-get-ElevationModelSpace
vla-get-ElevationPaperSpace
vla-get-Enable
vla-get-EnableBlockRotation
vla-get-EnableBlockScale
vla-get-EnableDogleg
vla-get-EnableFrameText
vla-get-EnableLanding
vla-get-EnableStartupDialog
vla-get-EndAngle
vla-get-EndDraftAngle
vla-get-EndDraftMagnitude
vla-get-EndParameter
vla-get-EndPoint
vla-get-EndSmoothContinuity
vla-get-EndSmoothMagnitude
vla-get-EndSubMenuLevel
vla-get-EndTangent
vla-get-EnterpriseMenuFile
vla-get-EntityColor
vla-get-EntityTransparency
vla-get-Explodable
vla-get-ExtensionLineColor
vla-get-ExtensionLineExtend
vla-get-ExtensionLineOffset
vla-get-ExtensionLineWeight
vla-get-ExtLine1EndPoint
vla-get-ExtLine1Linetype
vla-get-ExtLine1Point
vla-get-ExtLine1StartPoint
vla-get-ExtLine1Suppress
vla-get-ExtLine2EndPoint
vla-get-ExtLine2Linetype
vla-get-ExtLine2Point

vla-get-ExtLine2StartPoint
vla-get-ExtLine2Suppress
vla-get-ExtLineFixedLen
vla-get-ExtLineFixedLenSuppress
vla-get-FaceCount
vla-get-Fade
vla-get-Feature
vla-get-FieldLength
vla-get-File
vla-get-FileDependencies
vla-get-FileName
vla-get-Files
vla-get-FileSize
vla-get-FingerprintGuid
vla-get-FirstSegmentAngleConstraint
vla-get-Fit
vla-get-FitPoints
vla-get-FitTolerance
vla-get-FloatingRows
vla-get-FlowDirection
vla-get-Flyout
vla-get-FontFile
vla-get-FontFileMap
vla-get-ForceLineInside
vla-get-ForegroundLinesColor
vla-get-ForegroundLinesEdgeTransparency
vla-get-ForegroundLinesFaceTransparency
vla-get-ForegroundLinesHiddenLine
vla-get-ForegroundLinesLayer
vla-get-ForegroundLinesLinetype
vla-get-ForegroundLinesLinetypeScale
vla-get-ForegroundLinesLineweight
vla-get-ForegroundLinesPlotStyleName
vla-get-ForegroundLinesVisible

vla-get-FoundPath
vla-get-FractionFormat
vla-get-Freeze
vla-get-FullCRCValidation
vla-get-FullFileName
vla-get-FullName
vla-get-FullScreenTrackingVector
vla-get-GenerationOptions
vla-get-GradientAngle
vla-get-GradientCentered
vla-get-GradientColor1
vla-get-GradientColor2
vla-get-GradientName
vla-get-GraphicsWinLayoutBackgrndColor
vla-get-GraphicsWinModelBackgrndColor
vla-get-Green
vla-get-GridOn
vla-get-GripColorSelected
vla-get-GripColorUnselected
vla-get-GripSize
vla-get-Groups
vla-get-Handle
vla-get-HasAttributes
vla-get-HasExtensionDictionary
vla-get-HasLeader
vla-get-HasSheetView
vla-get-HasSubSelection
vla-get-HasVpAssociation
vla-get-HatchObjectType
vla-get-HatchStyle
vla-get-HeaderSuppressed
vla-get-Height
vla-get-HelpFilePath
vla-get-HelpString

vla-get-History
vla-get-HistoryLines
vla-get-HorizontalTextPosition
vla-get-HorzCellMargin
vla-get-HWND
vla-get-HyperlinkBase
vla-get-HyperlinkDisplayCursor
vla-get-Hyperlinks
vla-get-ImageFile
vla-get-ImageFrameHighlight
vla-get-ImageHeight
vla-get-ImageVisibility
vla-get-ImageWidth
vla-get-IncrementalSavePercent
vla-get-index
vla-get-IndicatorFillColor
vla-get-IndicatorTransparency
vla-get-InsertionPoint
vla-get-InsUnits
vla-get-InsUnitsFactor
vla-get-IntersectionBoundaryColor
vla-get-IntersectionBoundaryDivisionLines
vla-get-IntersectionBoundaryLayer
vla-get-IntersectionBoundaryLinetype
vla-get-IntersectionBoundaryLinetypeScale
vla-get-IntersectionBoundaryLineweight
vla-get-IntersectionBoundaryPlotStyleName
vla-get-IntersectionBoundaryVisible
vla-get-IntersectionFillColor
vla-get-IntersectionFillFaceTransparency
vla-get-IntersectionFillHatchAngle
vla-get-IntersectionFillHatchPatternName
vla-get-IntersectionFillHatchPatternType
vla-get-IntersectionFillHatchScale

vla-get-IntersectionFillHatchSpacing
vla-get-IntersectionFillLayer
vla-get-IntersectionFillLinetype
vla-get-IntersectionFillLinetypeScale
vla-get-IntersectionFillLineweight
vla-get-IntersectionFillPlotStyleName
vla-get-IntersectionFillVisible
vla-get-Invisible
vla-get-IsCloned
vla-get-IsDynamicBlock
vla-get-IsLayout
vla-get-IsModified
vla-get-ISOPenWidth
vla-get-IsOwnerXlated
vla-get-IsPartial
vla-get-IsPeriodic
vla-get-IsPlanar
vla-get-IsPrimary
vla-get-IsQuiescent
vla-get-IsRational
vla-get-Issuer
vla-get-IsXRef
vla-get-ItemName
vla-get-JogAngle
vla-get-JogLocation
vla-get-Justification
vla-get-Key
vla-get-KeyboardAccelerator
vla-get-KeyboardPriority
vla-get-KeyLength
vla-get-Keywords
vla-get-KnotParameterization
vla-get-Knots
vla-get-Label

vla-get-LabelBlockId
vla-get-LandingGap
vla-get-LargeButtons
vla-get-LastHeight
vla-get-LastSavedBy
vla-get-Layer
vla-get-LayerOn
vla-get-LayerPropertyOverrides
vla-get-Layers
vla-get-LayerState
vla-get-Layout
vla-get-LayoutCreateViewport
vla-get-LayoutCrosshairColor
vla-get-LayoutDisplayMargins
vla-get-LayoutDisplayPaper
vla-get-LayoutDisplayPaperShadow
vla-get-LayoutId
vla-get-Layouts
vla-get-LayoutShowPlotSetup
vla-get-Leader1Point
vla-get-Leader2Point
vla-get-LeaderCount
vla-get-LeaderLineColor
vla-get-LeaderLineType
vla-get-LeaderLineTypeId
vla-get-LeaderLineWeight
vla-get-LeaderType
vla-get-Left
vla-get-Length
vla-get-LensLength
vla-get-Limits
vla-get-LinearScaleFactor
vla-get-LineSpacingDistance
vla-get-LineSpacingFactor

vla-get-LineSpacingStyle
vla-get-Linetype
vla-get-LinetypeGeneration
vla-get-Linetypes
vla-get-LinetypeScale
vla-get-LineWeight
vla-get-LineWeightDisplay
vla-get-LiveSectionEnabled
vla-get-LoadAcadLspInAllDocuments
vla-get-LocaleId
vla-get-Lock
vla-get-LockAspectRatio
vla-get-Locked
vla-get-LockPosition
vla-get-LogFileOn
vla-get-LogFilePath
vla-get-LowerLeftCorner
vla-get-Macro
vla-get-MainDictionary
vla-get-MaintainAssociativity
vla-get-MajorAxis
vla-get-MajorRadius
vla-get-Mask
vla-get-Material
vla-get-Materials
vla-get-MaxActiveViewports
vla-get-MaxAutoCADWindow
vla-get-MaxLeaderSegmentsPoints
vla-get-MClose
vla-get-MDensity
vla-get-Measurement
vla-get-MenuBar
vla-get-MenuFile
vla-get-MenuFileName

vla-get-MenuGroups
vla-get-Menus
vla-get-MinimumTableHeight
vla-get-MinimumTableWidth
vla-get-MinorAxis
vla-get-MinorRadius
vla-get-MLineScale
vla-get-Mode
vla-get-ModelCrosshairColor
vla-get-ModelSpace
vla-get-ModelType
vla-get-ModelView
vla-get-MomentOfInertia
vla-get-Monochrome
vla-get-MRUNumber
vla-get-MSpace
vla-get-MTextAttribute
vla-get-MTextAttributeContent
vla-get-MTextBoundaryWidth
vla-get-MTextDrawingDirection
vla-get-MVertexCount
vla-get-Name
vla-get-NameNoMnemonic
vla-get-NClose
vla-get-NDensity
vla-get-Normal
vla-get-NumberOfControlPoints
vla-get-NumberOfCopies
vla-get-NumberOfFaces
vla-get-NumberOfFitPoints
vla-get-NumberOfLoops
vla-get-NumberOfVertices
vla-get-NumCellStyles
vla-get-NumCrossSections

vla-get-NumGuidePaths
vla-get-NumVertices
vla-get-NVertexCount
vla-get-ObjectID
vla-get-ObjectName
vla-get-ObjectSnapMode
vla-get-ObjectSortByPlotting
vla-get-ObjectSortByPSOutput
vla-get-ObjectSortByRedraws
vla-get-ObjectSortByRegens
vla-get-ObjectSortBySelection
vla-get-ObjectSortBySnap
vla-get-ObliqueAngle
vla-get-OleItemType
vla-get-OLELaunch
vla-get-OlePlotQuality
vla-get-OLEQuality
vla-get-OleSourceApp
vla-get-OnMenuBar
vla-get-OpenSave
vla-get-Origin
vla-get-OrthoOn
vla-get-Output
vla-get-OverrideCenter
vla-get-OverwritePropChanged
vla-get-OwnerID
vla-get-PageSetupOverridesTemplateFile
vla-get-PaperSpace
vla-get-PaperUnits
vla-get-Parent
vla-get-Password
vla-get-Path
vla-get-PatternAngle
vla-get-PatternDouble

vla-get-PatternName
vla-get-PatternScale
vla-get-PatternSpace
vla-get-PatternType
vla-get-Perimeter
vla-get-Periodic
vla-get-PickAdd
vla-get-PickAuto
vla-get-PickBoxSize
vla-get-PickDrag
vla-get-PickFirst
vla-get-PickfirstSelectionSet
vla-get-PickGroup
vla-get-Plot
vla-get-PlotConfigurations
vla-get-PlotHidden
vla-get-PlotLegacy
vla-get-PlotLogFilePath
vla-get-PlotOrigin
vla-get-PlotPolicy
vla-get-PlotRotation
vla-get-PlotStyleName
vla-get-Plottable
vla-get-PlotType
vla-get-PlotViewportBorders
vla-get-PlotViewportsFirst
vla-get-PlotWithLineweights
vla-get-PlotWithPlotStyles
vla-get-PolarTrackingVector
vla-get-Position
vla-get-PostScriptPrologFile
vla-get-Preferences
vla-get-Preset
vla-get-PrimaryUnitsPrecision

vla-get-PrincipalDirections
vla-get-PrincipalMoments
vla-get-PrinterConfigPath
vla-get-PrinterDescPath
vla-get-PrinterPaperSizeAlert
vla-get-PrinterSpoolAlert
vla-get-PrinterStyleSheetPath
vla-get-PrintFile
vla-get-PrintSpoolerPath
vla-get-PrintSpoolExecutable
vla-get-ProductOfInertia
vla-get-ProfileRotation
vla-get-Profiles
vla-get-PromptString
vla-get-PropertyName
vla-get-ProviderName
vla-get-ProviderType
vla-get-ProxyImage
vla-get-QNewTemplateFile
vla-get-QuietErrorMode
vla-get-RadiiOfGyration
vla-get-Radius
vla-get-RadiusRatio
vla-get-ReadOnly
vla-get-Red
vla-get-ReferenceCount
vla-get-RegenerateTableSuppressed
vla-get-RegisteredApplications
vla-get-RenderSmoothness
vla-get-RepeatBottomLabels
vla-get-RepeatTopLabels
vla-get-RevisionNumber
vla-get-RevolutionAngle
vla-get-Rotation

vla-get-RoundDistance
vla-get-Rows
vla-get-RowSpacing
vla-get-SaveAsType
vla-get-Saved
vla-get-SavePreviewThumbnail
vla-get-scale
vla-get-ScaleFactor
vla-get-ScaleHeight
vla-get-ScaleLineweights
vla-get-ScaleWidth
vla-get-SCMCommandMode
vla-get-SCMDefaultMode
vla-get-SCMEditMode
vla-get-SCMTimeMode
vla-get-SCMTimeValue
vla-get-SecondPoint
vla-get-SecondSegmentAngleConstraint
vla-get-SectionManager
vla-get-SegmentPerPolyline
vla-get-Selection
vla-get-SelectionSets
vla-get-SerialNumber
vla-get-Settings
vla-get-ShadePlot
vla-get-SheetView
vla-get-ShortcutMenu
vla-get-ShortCutMenuDisplay
vla-get-Show
vla-get-ShowAssociativity
vla-get-ShowHistory
vla-get-ShowPlotStyles
vla-get-ShowProxyDialogBox
vla-get-ShowRasterImage

vla-get-ShowRotation
vla-get-ShowWarningMessages
vla-get-SingleDocumentMode
vla-get-Smoothness
vla-get-SnapBasePoint
vla-get-SnapOn
vla-get-SnapRotationAngle
vla-get-SolidFill
vla-get-SolidType
vla-get-SourceObjects
vla-get-SplineFrame
vla-get-SplineMethod
vla-get-StandardScale
vla-get-StandardScale2
vla-get-StartAngle
vla-get-StartDraftAngle
vla-get-StartDraftMagnitude
vla-get-StartParameter
vla-get-StartPoint
vla-get-StartSmoothContinuity
vla-get-StartSmoothMagnitude
vla-get-StartTangent
vla-get-State
vla-get-StatusId
vla-get-StoreSQLIndex
vla-get-StyleName
vla-get-StyleSheet
vla-get-Subject
vla-get-SubMenu
vla-get-SubUnitsFactor
vla-get-SubUnitsSuffix
vla-get-SummaryInfo
vla-get-SupportPath
vla-get-SuppressLeadingZeros

vla-get-SuppressTrailingZeros
vla-get-SuppressZeroFeet
vla-get-SuppressZeroInches
vla-get-SurfaceNormals
vla-get-SurfaceType
vla-get-SymbolPosition
vla-get-System
vla-get-TableBreakFlowDirection
vla-get-TableBreakHeight
vla-get-TablesReadOnly
vla-get-TableStyleOverrides
vla-get-TabOrder
vla-get-TagString
vla-get-TaperAngle
vla-get-Target
vla-get-TempFileExtension
vla-get-TempFilePath
vla-get-TemplateDwgPath
vla-get-TemplateId
vla-get-TempXrefPath
vla-get-TextAlignmentPoint
vla-get-TextAlignmentType
vla-get-TextAngleType
vla-get-TextAttachmentDirection
vla-get-TextBackgroundFill
vla-get-TextBottomAttachmentType
vla-get-TextColor
vla-get-TextDirection
vla-get-TextEditor
vla-get-TextFill
vla-get-TextFillColor
vla-get-TextFont
vla-get-TextFontSize
vla-get-TextFontStyle

vla-get-TextFrameDisplay
vla-get-TextGap
vla-get-TextGenerationFlag
vla-get-TextHeight
vla-get-TextInside
vla-get-TextInsideAlign
vla-get-TextJustify
vla-get-TextLeftAttachmentType
vla-get-TextLineSpacingDistance
vla-get-TextLineSpacingFactor
vla-get-TextLineSpacingStyle
vla-get-TextMovement
vla-get-TextOutsideAlign
vla-get-TextOverride
vla-get-TextPosition
vla-get-TextPrecision
vla-get-TextPrefix
vla-get-TextRightAttachmentType
vla-get-TextRotation
vla-get-TextString
vla-get-TextStyle
vla-get-TextStyleName
vla-get-TextStyles
vla-get-TextSuffix
vla-get-TextTopAttachmentType
vla-get-TextureMapPath
vla-get-TextWidth
vla-get-TextWinBackgrndColor
vla-get-TextWinTextColor
vla-get-Thickness
vla-get-TimeServer
vla-get-TimeStamp
vla-get-Title
vla-get-TitleSuppressed

vla-get-ToleranceDisplay
vla-get-ToleranceHeightScale
vla-get-ToleranceJustification
vla-get-ToleranceLowerLimit
vla-get-TolerancePrecision
vla-get-ToleranceSuppressLeadingZeros
vla-get-ToleranceSuppressTrailingZeros
vla-get-ToleranceSuppressZeroFeet
vla-get-ToleranceSuppressZeroInches
vla-get-ToleranceUpperLimit
vla-get-Toolbars
vla-get-ToolPalettePath
vla-get-Top
vla-get-TopHeight
vla-get-TopRadius
vla-get-TotalAngle
vla-get-TotalLength
vla-get-TranslateIDs
vla-get-Transparency
vla-get-TrueColor
vla-get-TrueColorImages
vla-get-TurnHeight
vla-get-Turns
vla-get-TurnSlope
vla-get-Twist
vla-get-TwistAngle
vla-get-Type
vla-get-UCSIconAtOrigin
vla-get-UCSIconOn
vla-get-UCSPerViewport
vla-get-UIsolineDensity
vla-get-UnderlayLayerOverrideApplied
vla-get-UnderlayName
vla-get-UnderlayVisibility

vla-get-Units
vla-get-UnitsFormat
vla-get-UnitsType
vla-get-UpperRightCorner
vla-get-UpsideDown
vla-get-URL
vla-get-URLDescription
vla-get-URLNamedLocation
vla-get-Used
vla-get-UseEntityColor
vla-get-UseLastPlotSettings
vla-get-User
vla-get-UserCoordinateSystems
vla-get-UseStandardScale
vla-get-Utility
vla-get-Value
vla-get-VBE
vla-get-Verify
vla-get-Version
vla-get-VersionGuid
vla-get-VertCellMargin
vla-get-VertexCount
vla-get-VerticalDirection
vla-get-VerticalTextPosition
vla-get-Vertices
vla-get-ViewingDirection
vla-get-ViewportDefault
vla-get-ViewportOn
vla-get-Viewports
vla-get-Views
vla-get-ViewToPlot
vla-get-VisibilityEdge1
vla-get-VisibilityEdge2
vla-get-VisibilityEdge3

vla-get-VisibilityEdge4
vla-get-Visible
vla-get-VIsolineDensity
vla-get-VisualStyle
vla-get-Volume
vla-get-Weights
vla-get-Width
vla-get-WindowLeft
vla-get-WindowState
vla-get-WindowTitle
vla-get-WindowTop
vla-get-WireframeType
vla-get-WorkspacePath
vla-get-XEffectiveScaleFactor
vla-get-XRefDatabase
vla-get-XrefDemandLoad
vla-get-XRefEdit
vla-get-XRefFadeIntensity
vla-get-XRefLayerVisibility
vla-get-XScaleFactor
vla-get-XVector
vla-get-YEffectiveScaleFactor
vla-get-YScaleFactor
vla-get-YVector
vla-get-ZEffectiveScaleFactor
vla-get-ZScaleFactor
vla-GetAcadState
vla-GetAlignment
vla-GetAlignment2
vla-GetAllProfileNames
vla-GetAngle
vla-GetAttachmentPoint
vla-GetAttributes
vla-GetAutoScale

vla-GetAutoScale2
vla-GetBackgroundColor
vla-GetBackgroundColor2
vla-GetBackgroundColorNone
vla-GetBitmaps
vla-GetBlockAttributeValue
vla-GetBlockAttributeValue2
vla-GetBlockRotation
vla-GetBlockScale
vla-GetBlockTableRecordId
vla-GetBlockTableRecordId2
vla-GetBoundingBox
vla-GetBreakHeight
vla-GetBulge
vla-GetCanonicalMediaNames
vla-GetCellAlignment
vla-GetCellBackgroundColor
vla-GetCellBackgroundColorNone
vla-GetCellClass
vla-GetCellContentColor
vla-GetCellDataType
vla-GetCellExtents
vla-GetCellFormat
vla-GetCellGridColor
vla-GetCellGridLineWeight
vla-GetCellGridVisibility
vla-GetCellState
vla-GetCellStyle
vla-GetCellStyleOverrides
vla-GetCellStyles
vla-GetCellTextHeight
vla-GetCellTextStyle
vla-GetCellType
vla-GetCellValue

vla-GetColor
vla-GetColor2
vla-GetColumnName
vla-GetColumnWidth
vla-GetConstantAttributes
vla-GetContentColor
vla-GetContentColor2
vla-GetContentLayout
vla-GetContentType
vla-GetControlPoint
vla-GetCorner
vla-GetCustomByIndex
vla-GetCustomByKey
vla-GetCustomData
vla-GetCustomScale
vla-GetDataFormat
vla-GetDataType
vla-GetDataType2
vla-GetDistance
vla-GetDoglegDirection
vla-GetDynamicBlockProperties
vla-GetEntity
vla-GetExtensionDictionary
vla-GetFieldId
vla-GetFieldId2
vla-GetFitPoint
vla-GetFont
vla-GetFormat
vla-GetFormat2
vla-GetFormula
vla-GetFullDrawOrder
vla-GetGridColor
vla-GetGridColor2
vla-GetGridDoubleLineSpacing

vla-GetGridLineStyle
vla-GetGridLinetype
vla-GetGridLineWeight
vla-GetGridLineWeight2
vla-GetGridSpacing
vla-GetGridVisibility
vla-GetGridVisibility2
vla-GetHasFormula
vla-GetInput
vla-GetInteger
vla-GetInterfaceObject
vla-GetInvisibleEdge
vla-GetIsCellStyleInUse
vla-GetIsMergeAllEnabled
vla-GetKeyword
vla-GetLeaderIndex
vla-GetLeaderLineIndexes
vla-GetLeaderLineVertices
vla-GetLiveSection
vla-GetLocaleMediaName
vla-GetLoopAt
vla-GetMargin
vla-GetMinimumColumnWidth
vla-GetMinimumRowHeight
vla-GetName
vla-GetObject
vla-GetObjectIdString
vla-GetOrientation
vla-GetOverride
vla-GetPaperMargins
vla-GetPaperSize
vla-GetPlotDeviceNames
vla-GetPlotStyleTableNames
vla-GetPoint

vla-GetProjectFilePath
vla-GetReal
vla-GetRelativeDrawOrder
vla-GetRemoteFile
vla-GetRotation
vla-GetRowHeight
vla-GetRowType
vla-GetScale
vla-GetSectionTypeSettings
vla-GetSnapSpacing
vla-GetString
vla-GetSubEntity
vla-GetSubSelection
vla-GetText
vla-GetTextHeight
vla-GetTextHeight2
vla-GetTextRotation
vla-GetTextString
vla-GetTextStyle
vla-GetTextStyle2
vla-GetTextStyleId
vla-GetUCSMatrix
vla-GetUniqueCellStyleName
vla-GetUniqueSectionName
vla-GetValue
vla-GetVariable
vla-GetVertexCount
vla-GetWeight
vla-GetWidth
vla-GetWindowToPlot
vla-GetXData
vla-GetXRecordData
vla-HandleToObject
vla-Highlight

vla-HitTest
vla-Import
vla-ImportProfile
vla-IndexOf
vla-InitializeUserInput
vla-InsertBlock
vla-InsertColumns
vla-InsertColumnsAndInherit
vla-InsertInMenuBar
vla-InsertLoopAt
vla-InsertMenuInMenuBar
vla-InsertRows
vla-InsertRowsAndInherit
vla-IntersectWith
vla-IsContentEditable
vla-IsEmpty
vla-IsFormatEditable
vla-IsMergeAllEnabled
vla-IsMergedCell
vla-IsRemoteFile
vla-IsURL
vla-Item
vla-LaunchBrowserDialog
vla-ListArx
vla-Load
vla-LoadArx
vla-LoadDVB
vla-LoadShapeFile
vla-MergeCells
vla-Mirror
vla-Mirror3D
vla-Modified
vla-Move
vla-MoveAbove

vla-MoveBelow
vla-MoveContent
vla-MoveToBottom
vla-MoveToTop
vla-New
vla-NumCustomInfo
VLA-OBJECT
vla-ObjectIDToObject
vla-Offset
vla-OnModified
vla-Open
vla-PlotToDevice
vla-PlotToFile
vla-PolarPoint
vla-Prompt
vla-PurgeAll
vla-PurgeFitData
vla-put-Action
vla-put-ActiveDimStyle
vla-put-ActiveDocument
vla-put-ActiveLayer
vla-put-ActiveLayout
vla-put-ActiveLinetype
vla-put-ActiveMaterial
vla-put-ActiveProfile
vla-put-ActivePViewport
vla-put-ActiveSpace
vla-put-ActiveTextStyle
vla-put-ActiveUCS
vla-put-ActiveViewport
vla-put-ADCInsertUnitsDefaultSource
vla-put-ADCInsertUnitsDefaultTarget
vla-put-AdjustForBackground
vla-put-Algorithm

vla-put-Alignment
vla-put-AlignmentPointAcquisition
vla-put-AlignSpace
vla-put-AllowLongSymbolNames
vla-put-AllowManualHeights
vla-put-AllowManualPositions
vla-put-AltFontFile
vla-put-AltRoundDistance
vla-put-AltSubUnitsFactor
vla-put-AltSubUnitsSuffix
vla-put-AltSuppressLeadingZeros
vla-put-AltSuppressTrailingZeros
vla-put-AltSuppressZeroFeet
vla-put-AltSuppressZeroInches
vla-put-AltTabletMenuFile
vla-put-AltTextPrefix
vla-put-AltTextSuffix
vla-put-AltTolerancePrecision
vla-put-AltToleranceSuppressLeadingZeros
vla-put-AltToleranceSuppressTrailingZeros
vla-put-AltToleranceSuppressZeroFeet
vla-put-AltToleranceSuppressZeroInches
vla-put-AltUnits
vla-put-AltUnitsFormat
vla-put-AltUnitsPrecision
vla-put-AltUnitsScale
vla-put-AngleFormat
vla-put-AngleVertex
vla-put-Annotation
vla-put-Annotative
vla-put-ArcEndParam
vla-put-ArcPoint
vla-put-ArcSmoothness
vla-put-ArcStartParam

vla-put-Area
vla-put-Arrowhead1Block
vla-put-Arrowhead1Type
vla-put-Arrowhead2Block
vla-put-Arrowhead2Type
vla-put-ArrowheadBlock
vla-put-ArrowheadSize
vla-put-ArrowheadType
vla-put-ArrowSize
vla-put-ArrowSymbol
vla-put-AssociativeHatch
vla-put-AttachmentPoint
vla-put-Author
vla-put-AutoAudit
vla-put-AutomaticPlotLog
vla-put-AutoSaveInterval
vla-put-AutoSavePath
vla-put-AutoSnapAperture
vla-put-AutoSnapApertureSize
vla-put-AutoSnapMagnet
vla-put-AutoSnapMarker
vla-put-AutoSnapMarkerColor
vla-put-AutoSnapMarkerSize
vla-put-AutoSnapToolTip
vla-put-AutoTrackingVecColor
vla-put-AutoTrackTooltip
vla-put-AxisPosition
vla-put-BackgroundColor
vla-put-BackgroundFill
vla-put-BackgroundLinesColor
vla-put-BackgroundLinesHiddenLine
vla-put-BackgroundLinesLayer
vla-put-BackgroundLinesLinetype
vla-put-BackgroundLinesLinetypeScale

vla-put-BackgroundLinesLineweight
vla-put-BackgroundLinesPlotStyleName
vla-put-BackgroundLinesVisible
vla-put-Backward
vla-put-Bank
vla-put-BasePoint
vla-put-BaseRadius
vla-put-BatchPlotProgress
vla-put-BeepOnError
vla-put-BigFontFile
vla-put-BitFlags
vla-put-Block
vla-put-BlockColor
vla-put-BlockConnectionType
vla-put-BlockRotation
vla-put-BlockScale
vla-put-BlockScaling
vla-put-BottomHeight
vla-put-BreaksEnabled
vla-put-BreakSize
vla-put-BreakSpacing
vla-put-Brightness
vla-put-CanonicalMediaName
vla-put-CategoryName
vla-put-Center
vla-put-CenterMarkSize
vla-put-CenterPlot
vla-put-CenterPoint
vla-put-CenterType
vla-put-Check
vla-put-ChordPoint
vla-put-Circumference
vla-put-ClippingEnabled
vla-put-Closed

vla-put-Closed2
vla-put-Color
vla-put-ColorBookPath
vla-put-ColorIndex
vla-put-ColorMethod
vla-put-Columns
vla-put-ColumnSpacing
vla-put-ColumnWidth
vla-put-CommandDisplayName
vla-put-Comment
vla-put-Comments
vla-put-ConfigName
vla-put-Constant
vla-put-ConstantWidth
vla-put-Constrain
vla-put-ContentBlockName
vla-put-ContentBlockType
vla-put-ContentType
vla-put-ContinuousPlotLog
vla-put-ContourLinesPerSurface
vla-put-Contrast
vla-put-ControlPoints
vla-put-Coordinate
vla-put-Coordinates
vla-put-CreaseLevel
vla-put-CreaseType
vla-put-CreateBackup
vla-put-CurrentSectionType
vla-put-CursorSize
vla-put-CurveTangencyLinesColor
vla-put-CurveTangencyLinesLayer
vla-put-CurveTangencyLinesLinetype
vla-put-CurveTangencyLinesLinetypeScale
vla-put-CurveTangencyLinesLineweight

vla-put-CurveTangencyLinesPlotStyleName
vla-put-CurveTangencyLinesVisible
vla-put-CustomDictionary
vla-put-CustomIconPath
vla-put-CustomScale
vla-put-CvHullDisplay
vla-put-DecimalSeparator
vla-put-DefaultInternetURL
vla-put-DefaultOutputDevice
vla-put-DefaultPlotStyleForLayer
vla-put-DefaultPlotStyleForObjects
vla-put-DefaultPlotStyleTable
vla-put-DefaultPlotToFilePath
vla-put-Degree2
vla-put-DemandLoadARXApp
vla-put-Description
vla-put-DestinationBlock
vla-put-DestinationFile
vla-put-Diameter
vla-put-DimConstrDesc
vla-put-DimConstrExpression
vla-put-DimConstrForm
vla-put-DimConstrName
vla-put-DimConstrReference
vla-put-DimConstrValue
vla-put-DimensionLineColor
vla-put-DimensionLineExtend
vla-put-DimensionLinetype
vla-put-DimensionLineWeight
vla-put-DimLine1Suppress
vla-put-DimLine2Suppress
vla-put-DimLineInside
vla-put-DimLineSuppress
vla-put-DimTxtDirection

vla-put-Direction
vla-put-DirectionVector
vla-put-DisplayGrips
vla-put-DisplayGripsWithinBlocks
vla-put-DisplayLayoutTabs
vla-put-DisplayLocked
vla-put-DisplayOLEScale
vla-put-DisplayScreenMenu
vla-put-DisplayScrollBars
vla-put-DisplaySilhouette
vla-put-DockedVisibleLines
vla-put-DogLegged
vla-put-DoglegLength
vla-put-DrawingDirection
vla-put-DrawLeaderOrderType
vla-put-DrawMLeaderOrderType
vla-put-DriversPath
vla-put-DWFFormat
vla-put-EdgeExtensionDistances
vla-put-Elevation
vla-put-ElevationModelSpace
vla-put-ElevationPaperSpace
vla-put-Enable
vla-put-EnableBlockRotation
vla-put-EnableBlockScale
vla-put-EnableBreak
vla-put-EnableDogleg
vla-put-EnableFrameText
vla-put-EnableLanding
vla-put-EnableStartupDialog
vla-put-EndAngle
vla-put-EndDraftAngle
vla-put-EndDraftMagnitude
vla-put-EndParameter

vla-put-EndPoint
vla-put-EndSmoothContinuity
vla-put-EndSmoothMagnitude
vla-put-EndSubMenuLevel
vla-put-EndTangent
vla-put-EnterpriseMenuFile
vla-put-EntityColor
vla-put-EntityTransparency
vla-put-Explodable
vla-put-ExtensionLineColor
vla-put-ExtensionLineExtend
vla-put-ExtensionLineOffset
vla-put-ExtensionLineWeight
vla-put-ExtLine1EndPoint
vla-put-ExtLine1Linetype
vla-put-ExtLine1Point
vla-put-ExtLine1StartPoint
vla-put-ExtLine1Suppress
vla-put-ExtLine2EndPoint
vla-put-ExtLine2Linetype
vla-put-ExtLine2Point
vla-put-ExtLine2StartPoint
vla-put-ExtLine2Suppress
vla-put-ExtLineFixedLen
vla-put-ExtLineFixedLenSuppress
vla-put-Fade
vla-put-FieldLength
vla-put-File
vla-put-FirstSegmentAngleConstraint
vla-put-Fit
vla-put-FitPoints
vla-put-FitTolerance
vla-put-FloatingRows
vla-put-FlowDirection

vla-put-FontFile
vla-put-FontFileMap
vla-put-ForceLineInside
vla-put-ForegroundLinesColor
vla-put-ForegroundLinesEdgeTransparency
vla-put-ForegroundLinesFaceTransparency
vla-put-ForegroundLinesHiddenLine
vla-put-ForegroundLinesLayer
vla-put-ForegroundLinesLinetype
vla-put-ForegroundLinesLinetypeScale
vla-put-ForegroundLinesLineweight
vla-put-ForegroundLinesPlotStyleName
vla-put-ForegroundLinesVisible
vla-put-FractionFormat
vla-put-Freeze
vla-put-FullCRCValidation
vla-put-FullScreenTrackingVector
vla-put-GenerationOptions
vla-put-GradientAngle
vla-put-GradientCentered
vla-put-GradientColor1
vla-put-GradientColor2
vla-put-GradientName
vla-put-GraphicsWinLayoutBackgrndColor
vla-put-GraphicsWinModelBackgrndColor
vla-put-GridOn
vla-put-GripColorSelected
vla-put-GripColorUnselected
vla-put-GripSize
vla-put-HasLeader
vla-put-HasVpAssociation
vla-put-HatchObjectType
vla-put-HatchStyle
vla-put-HeaderSuppressed

vla-put-Height
vla-put-HelpFilePath
vla-put-HelpString
vla-put-History
vla-put-HistoryLines
vla-put-HorizontalTextPosition
vla-put-HorzCellMargin
vla-put-HyperlinkBase
vla-put-HyperlinkDisplayCursor
vla-put-ImageFile
vla-put-ImageFrameHighlight
vla-put-ImageHeight
vla-put-ImageVisibility
vla-put-ImageWidth
vla-put-IncrementalSavePercent
vla-put-IndicatorFillColor
vla-put-IndicatorTransparency
vla-put-InsertionPoint
vla-put-IntersectionBoundaryColor
vla-put-IntersectionBoundaryDivisionLines
vla-put-IntersectionBoundaryLayer
vla-put-IntersectionBoundaryLinetype
vla-put-IntersectionBoundaryLinetypeScale
vla-put-IntersectionBoundaryLineweight
vla-put-IntersectionBoundaryPlotStyleName
vla-put-IntersectionBoundaryVisible
vla-put-IntersectionFillColor
vla-put-IntersectionFillFaceTransparency
vla-put-IntersectionFillHatchAngle
vla-put-IntersectionFillHatchPatternName
vla-put-IntersectionFillHatchPatternType
vla-put-IntersectionFillHatchScale
vla-put-IntersectionFillHatchSpacing
vla-put-IntersectionFillLayer

vla-put-IntersectionFillLinetype
vla-put-IntersectionFillLinetypeScale
vla-put-IntersectionFillLineweight
vla-put-IntersectionFillPlotStyleName
vla-put-IntersectionFillVisible
vla-put-Invisible
vla-put-ISOPenWidth
vla-put-IsPartial
vla-put-Issuer
vla-put-ItemName
vla-put-JogAngle
vla-put-JogLocation
vla-put-Justification
vla-put-KeyboardAccelerator
vla-put-KeyboardPriority
vla-put-KeyLength
vla-put-Keywords
vla-put-KnotParameterization
vla-put-Knots
vla-put-Label
vla-put-LabelBlockId
vla-put-LandingGap
vla-put-LargeButtons
vla-put-LastHeight
vla-put-LastSavedBy
vla-put-Layer
vla-put-LayerOn
vla-put-LayerState
vla-put-LayoutCreateViewport
vla-put-LayoutCrosshairColor
vla-put-LayoutDisplayMargins
vla-put-LayoutDisplayPaper
vla-put-LayoutDisplayPaperShadow
vla-put-LayoutId

vla-put-LayoutShowPlotSetup
vla-put-Leader1Point
vla-put-Leader2Point
vla-put-LeaderLength
vla-put-LeaderLineColor
vla-put-LeaderLineType
vla-put-LeaderLineTypeId
vla-put-LeaderLineWeight
vla-put-LeaderType
vla-put-Left
vla-put-LensLength
vla-put-Limits
vla-put-LinearScaleFactor
vla-put-LineSpacingDistance
vla-put-LineSpacingFactor
vla-put-LineSpacingStyle
vla-put-Linetype
vla-put-LinetypeGeneration
vla-put-LinetypeScale
vla-put-LineWeight
vla-put-LineWeightDisplay
vla-put-LiveSectionEnabled
vla-put-LoadAcadLspInAllDocuments
vla-put-Lock
vla-put-LockAspectRatio
vla-put-Locked
vla-put-LockPosition
vla-put-LogFileOn
vla-put-LogFilePath
vla-put-Macro
vla-put-MainDictionary
vla-put-MaintainAssociativity
vla-put-MajorAxis
vla-put-MajorRadius

vla-put-Mask
vla-put-Material
vla-put-MaxActiveViewports
vla-put-MaxAutoCADWindow
vla-put-MaxLeaderSegmentsPoints
vla-put-MClose
vla-put-MDensity
vla-put-MenuFile
vla-put-MinorRadius
vla-put-MLineScale
vla-put-Mode
vla-put-ModelCrosshairColor
vla-put-ModelView
vla-put-Monochrome
vla-put-MSpace
vla-put-MTextAttribute
vla-put-MTextAttributeContent
vla-put-MTextBoundaryWidth
vla-put-MTextDrawingDirection
vla-put-Name
vla-put-NClose
vla-put-NDensity
vla-put-Normal
vla-put-NumberOfCopies
vla-put-ObjectSnapMode
vla-put-ObjectSortByPlotting
vla-put-ObjectSortByPSOutput
vla-put-ObjectSortByRedraws
vla-put-ObjectSortByRegens
vla-put-ObjectSortBySelection
vla-put-ObjectSortBySnap
vla-put-ObliqueAngle
vla-put-OleItemType
vla-put-OLELaunch

vla-put-OlePlotQuality
vla-put-OLEQuality
vla-put-OleSourceApp
vla-put-Origin
vla-put-OrthoOn
vla-put-OverrideCenter
vla-put-PageSetupOverridesTemplateFile
vla-put-PaperUnits
vla-put-Password
vla-put-Path
vla-put-PatternAngle
vla-put-PatternDouble
vla-put-PatternScale
vla-put-PatternSpace
vla-put-Periodic
vla-put-PickAdd
vla-put-PickAuto
vla-put-PickBoxSize
vla-put-PickDrag
vla-put-PickFirst
vla-put-PickGroup
vla-put-PlotHidden
vla-put-PlotLegacy
vla-put-PlotLogFilePath
vla-put-PlotOrigin
vla-put-PlotPolicy
vla-put-PlotRotation
vla-put-PlotStyleName
vla-put-Plottable
vla-put-PlotType
vla-put-PlotViewportBorders
vla-put-PlotViewportsFirst
vla-put-PlotWithLineweights
vla-put-PlotWithPlotStyles

vla-put-PolarTrackingVector
vla-put-Position
vla-put-PostScriptPrologFile
vla-put-Preset
vla-put-PrimaryUnitsPrecision
vla-put-PrinterConfigPath
vla-put-PrinterDescPath
vla-put-PrinterPaperSizeAlert
vla-put-PrinterSpoolAlert
vla-put-PrinterStyleSheetPath
vla-put-PrintFile
vla-put-PrintSpoolerPath
vla-put-PrintSpoolExecutable
vla-put-ProfileRotation
vla-put-PromptString
vla-put-ProviderName
vla-put-ProviderType
vla-put-ProxyImage
vla-put-QNewTemplateFile
vla-put-QuietErrorMode
vla-put-Radius
vla-put-RadiusRatio
vla-put-RegenerateTableSuppressed
vla-put-RenderSmoothness
vla-put-RepeatBottomLabels
vla-put-RepeatTopLabels
vla-put-RevisionNumber
vla-put-RevolutionAngle
vla-put-Rotation
vla-put-RoundDistance
vla-put-RowHeight
vla-put-Rows
vla-put-RowSpacing
vla-put-SaveAsType

vla-put-SavePreviewThumbnail
vla-put-scale
vla-put-ScaleFactor
vla-put-ScaleHeight
vla-put-ScaleLineweights
vla-put-ScaleWidth
vla-put-SCMCommandMode
vla-put-SCMDefaultMode
vla-put-SCMEditMode
vla-put-SCMTimeMode
vla-put-SCMTimeValue
vla-put-SecondPoint
vla-put-SecondSegmentAngleConstraint
vla-put-SegmentPerPolyline
vla-put-SerialNumber
vla-put-ShadePlot
vla-put-SheetView
vla-put-ShortCutMenuDisplay
vla-put-ShowAssociativity
vla-put-ShowHistory
vla-put-ShowPlotStyles
vla-put-ShowProxyDialogBox
vla-put-ShowRasterImage
vla-put-ShowRotation
vla-put-ShowWarningMessages
vla-put-SingleDocumentMode
vla-put-Smoothness
vla-put-SnapBasePoint
vla-put-SnapOn
vla-put-SnapRotationAngle
vla-put-SolidFill
vla-put-SourceObjects
vla-put-SplineFrame
vla-put-SplineMethod

vla-put-StandardScale
vla-put-StandardScale2
vla-put-StartAngle
vla-put-StartDraftAngle
vla-put-StartDraftMagnitude
vla-put-StartParameter
vla-put-StartPoint
vla-put-StartSmoothContinuity
vla-put-StartSmoothMagnitude
vla-put-StartTangent
vla-put-State
vla-put-StoreSQLIndex
vla-put-StyleName
vla-put-StyleSheet
vla-put-Subject
vla-put-SubUnitsFactor
vla-put-SubUnitsSuffix
vla-put-SupportPath
vla-put-SuppressLeadingZeros
vla-put-SuppressTrailingZeros
vla-put-SuppressZeroFeet
vla-put-SuppressZeroInches
vla-put-SurfaceNormals
vla-put-SymbolPosition
vla-put-TableBreakFlowDirection
vla-put-TableBreakHeight
vla-put-TablesReadOnly
vla-put-TabOrder
vla-put-TagString
vla-put-TaperAngle
vla-put-Target
vla-put-TempFileExtension
vla-put-TempFilePath
vla-put-TemplateDwgPath

vla-put-TemplateId
vla-put-TempXrefPath
vla-put-TextAlignmentPoint
vla-put-TextAlignmentType
vla-put-TextAngleType
vla-put-TextAttachmentDirection
vla-put-TextBackgroundFill
vla-put-TextBottomAttachmentType
vla-put-TextColor
vla-put-TextDirection
vla-put-TextEditor
vla-put-TextFill
vla-put-TextFillColor
vla-put-TextFont
vla-put-TextFontSize
vla-put-TextFontStyle
vla-put-TextFrameDisplay
vla-put-TextGap
vla-put-TextGenerationFlag
vla-put-TextHeight
vla-put-TextInside
vla-put-TextInsideAlign
vla-put-TextJustify
vla-put-TextLeftAttachmentType
vla-put-TextLineSpacingDistance
vla-put-TextLineSpacingFactor
vla-put-TextLineSpacingStyle
vla-put-TextMovement
vla-put-TextOutsideAlign
vla-put-TextOverride
vla-put-TextPosition
vla-put-TextPrecision
vla-put-TextPrefix
vla-put-TextRightAttachmentType

vla-put-TextRotation
vla-put-TextString
vla-put-TextStyle
vla-put-TextStyleName
vla-put-TextSuffix
vla-put-TextTopAttachmentType
vla-put-TextureMapPath
vla-put-TextWidth
vla-put-TextWinBackgrndColor
vla-put-TextWinTextColor
vla-put-Thickness
vla-put-TimeServer
vla-put-Title
vla-put-TitleSuppressed
vla-put-ToleranceDisplay
vla-put-ToleranceHeightScale
vla-put-ToleranceJustification
vla-put-ToleranceLowerLimit
vla-put-TolerancePrecision
vla-put-ToleranceSuppressLeadingZeros
vla-put-ToleranceSuppressTrailingZeros
vla-put-ToleranceSuppressZeroFeet
vla-put-ToleranceSuppressZeroInches
vla-put-ToleranceUpperLimit
vla-put-ToolPalettePath
vla-put-Top
vla-put-TopHeight
vla-put-TopRadius
vla-put-TranslateIDs
vla-put-Transparency
vla-put-TrueColor
vla-put-TrueColorImages
vla-put-TurnHeight
vla-put-Turns

vla-put-Twist
vla-put-TwistAngle
vla-put-Type
vla-put-UCSIconAtOrigin
vla-put-UCSIconOn
vla-put-UCSPerViewport
vla-put-UIsolineDensity
vla-put-UnderlayLayerOverrideApplied
vla-put-UnderlayName
vla-put-UnderlayVisibility
vla-put-Units
vla-put-UnitsFormat
vla-put-UpsideDown
vla-put-URL
vla-put-URLDescription
vla-put-URLNamedLocation
vla-put-UseEntityColor
vla-put-UseLastPlotSettings
vla-put-UseStandardScale
vla-put-Value
vla-put-Verify
vla-put-VertCellMargin
vla-put-VerticalDirection
vla-put-VerticalTextPosition
vla-put-Vertices
vla-put-ViewingDirection
vla-put-ViewportDefault
vla-put-ViewportOn
vla-put-ViewToPlot
vla-put-VisibilityEdge1
vla-put-VisibilityEdge2
vla-put-VisibilityEdge3
vla-put-VisibilityEdge4
vla-put-Visible

vla-put-VisolineDensity
vla-put-VisualStyle
vla-put-Weights
vla-put-Width
vla-put-WindowLeft
vla-put-WindowState
vla-put-WindowTop
vla-put-WireframeType
vla-put-WorkspacePath
vla-put-XEffectiveScaleFactor
vla-put-XrefDemandLoad
vla-put-XRefEdit
vla-put-XRefFadeIntensity
vla-put-XRefLayerVisibility
vla-put-XScaleFactor
vla-put-XVector
vla-put-YEffectiveScaleFactor
vla-put-YScaleFactor
vla-put-YVector
vla-put-ZEffectiveScaleFactor
vla-put-ZScaleFactor
vla-PutRemoteFile
vla-Quit
vla-RealToString
vla-RecomputeTableBlock
vla-RefreshPlotDeviceInfo
vla-Regen
vla-Reload
vla-Remove
vla-RemoveAllOverrides
vla-RemoveCustomByIndex
vla-RemoveCustomByKey
vla-RemoveEntry
vla-RemoveFromMenuBar

vla-RemoveItems
vla-RemoveLeader
vla-RemoveLeaderLine
vla-RemoveMenuFromMenuBar
vla-RemoveVertex
vla-Rename
vla-RenameCellStyle
vla-RenameProfile
vla-Replace
vla-ReselectSubRegion
vla-ResetBlock
vla-ResetCellValue
vla-ResetProfile
vla-Restore
vla-Reverse
vla-Rotate
vla-Rotate3D
vla-RunMacro
vla-Save
vla-SaveAs
vla-ScaleEntity
vla-SectionSolid
vla-Select
vla-SelectAtPoint
vla-SelectByPolygon
vla-SelectOnScreen
vla-SelectSubRegion
vla-SendCommand
vla-SendModelessOperationEnded
vla-SendModelessOperationStart
vla-SetAlignment
vla-SetAlignment2
vla-SetAutoScale
vla-SetAutoScale2

vla-SetBackgroundColor
vla-SetBackgroundColor2
vla-SetBackgroundColorNone
vla-SetBitmaps
vla-SetBlockAttributeValue
vla-SetBlockAttributeValue2
vla-SetBlockRotation
vla-SetBlockScale
vla-SetBlockTableRecordId
vla-SetBlockTableRecordId2
vla-SetBreakHeight
vla-SetBulge
vla-SetCellAlignment
vla-SetCellBackgroundColor
vla-SetCellBackgroundColorNone
vla-SetCellClass
vla-SetCellContentColor
vla-SetCellDataType
vla-SetCellFormat
vla-SetCellGridColor
vla-SetCellGridLineWeight
vla-SetCellGridVisibility
vla-SetCellState
vla-SetCellStyle
vla-SetCellTextHeight
vla-SetCellTextStyle
vla-SetCellType
vla-SetCellValue
vla-SetCellValueFromText
vla-SetColor
vla-SetColor2
vla-SetColorBookColor
vla-SetColumnName
vla-SetColumnWidth

vla-SetContentColor
vla-SetContentColor2
vla-SetContentLayout
vla-SetControlPoint
vla-SetCustomByIndex
vla-SetCustomByKey
vla-SetCustomData
vla-SetCustomScale
vla-SetDatabase
vla-SetDataFormat
vla-SetDataType
vla-SetDataType2
vla-SetDoglegDirection
vla-SetFieldId
vla-SetFieldId2
vla-SetFitPoint
vla-SetFont
vla-SetFormat
vla-SetFormat2
vla-SetFormula
vla-SetGridColor
vla-SetGridColor2
vla-SetGridDoubleLineSpacing
vla-SetGridLineStyle
vla-SetGridLinetype
vla-SetGridLineWeight
vla-SetGridLineWeight2
vla-SetGridSpacing
vla-SetGridVisibility
vla-SetGridVisibility2
vla-SetInvisibleEdge
vla-SetLayoutsToPlot
vla-SetLeaderLineVertices
vla-SetMargin

vla-SetNames
vla-SetOverride
vla-SetPattern
vla-SetProjectFilePath
vla-SetRelativeDrawOrder
vla-SetRGB
vla-SetRotation
vla-SetRowHeight
vla-SetScale
vla-SetSnapSpacing
vla-SetSubSelection
vla-SetTemplateId
vla-SetText
vla-SetTextHeight
vla-SetTextHeight2
vla-SetTextRotation
vla-SetTextString
vla-SetTextStyle
vla-SetTextStyle2
vla-SetTextStyleId
vla-SetToolTip
vla-SetValue
vla-SetValueFromText
vla-SetVariable
vla-SetView
vla-SetWeight
vla-SetWidth
vla-SetWindowToPlot
vla-SetXData
vla-SetXRecordData
vla-SliceSolid
vla-Split
vla-StartBatchMode
vla-StartUndoMark

vla-SwapOrder
vla-SyncModelView
vla-TransformBy
vla-TranslateCoordinates
vla-Unload
vla-UnloadArx
vla-UnloadDVB
vla-UnmergeCells
vla-Update
vla-UpdateEntry
vla-UpdateMTextAttribute
vla-Wblock
vla-ZoomAll
vla-ZoomCenter
vla-ZoomExtents
vla-ZoomPickWindow
vla-ZoomPrevious
vla-ZoomScaled
vla-ZoomWindow
VLARTS-INIT

C

VLAX 函数名称

vlax-3D-point
vlax-add-cmd
vlax-create-object
vlax-curve-getArea
vlax-curve-getDistAtParam
vlax-curve-getDistAtPoint
vlax-curve-getEndParam
vlax-curve-getEndPoint
vlax-curve-getParamAtDist
vlax-curve-getParamAtPoint
vlax-curve-getPointAtDist
vlax-curve-getPointAtParam
vlax-curve-getStartParam
vlax-curve-getStartPoint
vlax-curve-isClosed
vlax-curve-isPeriodic
vlax-curve-isPlanar
vlax-curve-getClosestPointTo
vlax-curve-getClosestPointToProjection
vlax-curve-getFirstDeriv
vlax-curve-getSecondDeriv
vlax-dump-object

vlax-ename->vla-object
vlax-erased-p
vlax-for
vlax-get
vlax-get-acad-object
vlax-get-object
vlax-get-or-create-object
vlax-get-property
vlax-import-type-library
vlax-invoke
vlax-invoke-method
vlax-ldata-delete
vlax-ldata-get
vlax-ldata-list
vlax-ldata-put
vlax-ldata-test
vlax-make-safearray
vlax-make-variant
vlax-map-collection
vlax-method-applicable-p
vlax-object-released-p
vlax-product-key
vlax-property-available-p
vlax-put-property
vlax-read-enabled-p
vlax-release-object
vlax-remove-cmd
vlax-safearray-fill
vlax-safearray-get-dim
vlax-safearray-get-element
vlax-safearray-get-l-bound
vlax-safearray-get-u-bound
vlax-safearray-put-element
vlax-safearray-type

vlax-safearray->list
vlax-tmatrix
vlax-typeinfo-available-p
vlax-variant-change-type
vlax-variant-type
vlax-variant-value
vlax-vla-object->ename
vlax-write-enabled-p

Visual LISP IDE 键盘快捷键

表 D.1 显示了在 Visual LISP 集成开发环境编辑器中的默认键盘快捷键：

表 D.1 VLIDE 默认键盘快捷键

快捷键	功能描述
F1	帮助
F3	查找 / 替换下一个
F6	LISP 控制台
F8	下一个嵌套表达式
Shift + F8	下一个表达式
Ctrl + Shift + F8	跳出
Ctrl + F8	继续
F9	切换断点
Ctrl + Shift + F9	清除所有断点
Ctrl + F9	上一断点源代码
Ctrl + W	添加监视
Ctrl + R	重置为顶层
Ctrl + Q	退出当前层
Alt + F6	缩放窗口
Alt + Q	退出 Visual LISP 集成开发环境

Visual LISP 中的小贴士和技巧

E.1 为 (autoload) 函数增加对 VLX 的支持

默认情况下，AutoLISP (autoload) 函数在搜索指定文件加载时只会查找 LSP、FAS 或 MNL 文件类型。它根本不会考虑 VLX 文件。这是 AUTODESK® 的一个很小但很重要的疏忽，不过幸运的是它很容易修复。只需打开位于 AutoCAD 安装的 Support 文件夹中的 acad2000doc.lsp 文件^①。然后找到 (defun ai_ffile) 函数定义并添加对 VLX 文件的额外检查，保存文件并关闭。你需要在希望执行此更改的每台机器上完成这项工作。

E.2 保存你的 VLIDE 配置

当你修改编辑器配置时，你的更改将保存在名为 VLIDE.DSK 的配置文件中，该文件位于 AutoCAD 本地（或网络客户端）的支持文件夹中^②。你应该在别处保留此文件的副本，以避免在重新安装 AutoCAD 或安装服务包更新时覆盖它。此文件包含你的格式首选项（颜色、制表符、缩进等）。

E.3 从 VLX 文件中恢复 DCL 代码

在第 13 章（创建 Visual LISP 应用程序）中详细讨论了如何将 LSP 和 DCL 编译成 VLX 输出。需要记住的一件事是，虽然 LSP 代码在被编译为 VLX 输出之前首先被编译为

^① 译者注：这个文件通常位于 %PROGRAMFILES%\Autodesk\AutoCAD 2020\Support\zh-cn\acad2020doc.lsp，具体路径视你的安装路径和语言版本而定。

^② 译者注：以 AutoCAD® 2020 为例，这个路径为：%USERPROFILE%\AppData\Roaming\Autodesk\AutoCAD 2020\R23.1\chs\VLIDE.DSK，具体路径视你的安装路径和语言版本而定。

但 FAS 格式，但 DCL 代码根本没有被编译。它只是附加到 VLX 输出文件的底部。因此，你可以在任何标准文本编辑器（例如 Windows® 记事本）中打开 VLX 文件，浏览到文件底部就可以完整地找到所有 DCL 代码。每当你错误地删除了一个 DCL 文件但碰巧有可用的 VLX 时，你可以从那里复制并粘贴回一个新的 DCL 文件。

E.4 通过 生成应用程序 向导使用工程和 DCL

为了充分利用工程，将与特定 VLX 所有相关文件添加到工程中是一个很好的做法。然后根据函数定义语句的顺序（加载顺序）以正确的顺序对它们进行排序。接下来，在使用 生成应用程序 向导时，选择工程 PRJ 文件而不是各个单独的 LSP 文件。

由于 DCL 文件不能包含在项目列表中，你通常必须将它们单独添加到 生成应用程序 向导的资源文件列表中（仅限专家模式）。然而，另一种方法可以是将所有 DCL 文件连接成一个 DCL 文件，这必须添加一个 PRJ 和一个 DCL 文件来制作 VLX 应用程序。

要将多个 DCL 文件连接成一个 DCL 文件，请使用古老的 DOS 命令 COPY，如下所示：

```
$ >copy *.dcl all.dcl
```

这会将所有 DCL 文件复制到一个名为 all.dcl 的文件中。请记住要考虑相对路径，或从同一路径位置运行所有内容。

E.5 基于团队的 VLX 开发

如果你正在与其他几个开发人员一起开发基于网络的项目，那么应该记住一些事情。首先，PRV 格式在编译 VLX 应用程序时会存储源文件的路径/驱动器信息。其次，共享 PRV 设置文件仅在相对路径可以移植到所有预期用户时才有效。便携是指相对路径/驱动器信息必须同样适用于所有目标用户。有一个好办法，那就是让所有开发人员将整个源代码集复制到他们的本地硬盘驱动器并为所有 PRV 文件设置使用本地路径，然后在构建 VLX 应用程序后，再将它们复制到预期的网络服务器。

此外，如果你有某种版本管理软件，例如 Visual Source Safe、StarBase、PVCS-DOORS 或其他软件^①，你应该使用它来管理所有各种文件的签出、签入和版本控制以免互相倾轧而造成混乱。

^① 译者注：这些软件我都没用过，git 才是王道。

有用的网络资源

与我在 2003 年版中包含的列表不同，有些已经从互联网上消失了。仍然可用的如下：^①

- <http://download.rhino3d.com/McNeel/1.0/doslib>
- <http://support.autodesk.com>
- <http://www.augi.com>
- <http://www.vbdesign.net>
- <http://www.myitforum.com>
- <http://www.upfront.com>
- <http://www.microsoft.com/scripting>
- <http://www.cadinfo.net>
- <http://www.adminscripts.net>
- <http://sites.google.com/site/skatterbrainz>
- <http://scriptzilla.blogspot.com>

^① 译者注：现在是 2023 年了，剩下的这些链接也有的不可用了，处于对作者原文的尊重，我们将网址罗列在下面，已经失效的没有用蓝色高亮。

致谢

我真的很想感谢一些人，他们给了我知识、渴望和毅力，令我尝试再次让这本书重现生命。特别感谢 PHILL ASH 在本书付梓之前进行的校对工作。

GENE STRAKA —感谢你写了我读过的，无论从语言还是从技术上来说都是最好的编程书籍之一（《AutoLISP 编程实例》^①）。那本书启发了我。

PETER SEIBEL —感谢你写了《实用 Common Lisp 编程》^②，这本书实用到每个人都可以照着书来实践。

PHILL ASH, JON SZEWCZAK and BRAD HAMILTON —感谢你们分享想法和挑战极限，无论是否大量饮用啤酒和咖啡。

JERRY MILANA, FRANK MOORE, SHAAN HURLEY —AUTODESK® 的员工，一群伟大的人，我为能在这些年与你们相遇而感到荣幸。

BOBBY JOHNSON, EVERETT BROWNING, PAUL PERUSSE, STEPHEN CORREIA —感谢你们给我提供了令人难以置信的成长和探索空间，即使有时违反了规则。

感谢我的妻子 KATHY，和我们的四个不可思议的孩子，你们忍受了我那些疯狂的想法。

JOE SUTPHIN, KENNY RAMAGE, SHERKO SHARIF, FRANK OQUENDO, BILL KRAMER, SCOTT MCFARLANE, OWEN WENGARD, RHEINI URBAN, RANDALL RATH（无论身处何方），感谢他们为了我们所有人的利益而突破极限。

以上或本书中某些地方可能会提及某些知名公司工作的人员，他们之所以被提及，是因为他们的慷慨、为人称道的专业知识和愿意帮助他人更好地理解利用本文讨论的软件技术的整体同情心。这绝不是代表他们的雇主或他们自己表达承认、批准或宽恕的声明。其中一些人甚至不知道我在这里给他们点名了。哈哈。

^① 译者注：原著英文名称 *AutoLISP Programming by Example*

^② 译者注：原著英文名称 *Practical Common Lisp*，国内有正式出版的译作，田春译，2011 年 10 月第 1 版，人民邮电出版社出版，ISBN: 978-7-115-26374-2。

词汇表

ActiveX (哥们儿, 你去问 Microsoft® 吧) .

书签 放置在文档中的位置标记, 让用户能够快速返回到该位置.

断点 放置在程序代码中的标记, 指示编译器或解释器在运行时暂停执行并等待用户执行调试任务或继续执行.

回调 对给定操作或事件的请求响应。例如, 单击按钮的回调可能是"Accept", 然后触发对特定函数或表达式的调用.

集合 具有共同父对象和相关属性或方法的一组对象, 这些属性或方法支持以逻辑方式将对象作为一个组进行处理.

COM 组件对象模型。Microsoft® 的一项技术, 它定义了软件组件和服务的分层组织, 并为组件和服务提供了内在属性、方法和事件, 以实现更高效的编程操作并促进组件化的功能重用。其他类型包括分布式 COM 或 DCOM 以及 Windows® 2000 和 Windows® XP 平台中包含的较新的 COM+.

常量 具有静态赋值的变量或符号.

使用方 导入或使用另一个软件组件或服务的公开组件服务的任何软件组件或应用程序。导入的组件或服务的来源称为提供方.

控件 一个 ActiveX DLL 组件.

数据类型 与特定值表示的数据形式有关的内在性质。ActiveX 数据类型的示例包括整数型 (Integer)、长整数型 (Long)、双精度浮点数型 (Double)、字符串型 (String) 和数组型 (Array) .

DCL 对话框控制语言, 基于 C 语言构造, 用于在 Auto/Visual LISP 环境中定义对话框形式.

调试 隔离、诊断和纠正程序代码或编程逻辑中错误的过程.

词典 一种集合类型, 它通过为每个对象使用唯一标识符来提供对成员对象的直接访问.

DLL Dynamic Link Library, 动态链接库, 基于 Windows® 的 ActiveX 组件, 通常公开函数、属性、方法和常量以供其他应用程序使用。它类似于一个打包的工具库, 应用程序可以加载这些工具来执行专门的任务。

元素 数组或安全数组构造的单个成员。

枚举 (这个需要官方定义)。

求值 执行 LISP 表达式 或从 LISP 符号中提取关联值并返回结果的过程。

事件 程序中发生某些动作的时刻。可以是单击按钮或移动实体。事件通常提供可以使用反应器或回调检测和响应的编程通知。

表达式 Auto/Visual LISP 解释器环境上下文中的程序语句。

焦点 DCL 对话框中给定项的状态或者由活动光标位置控制。如果编辑框的光标处于活动状态并且是可编辑的, 则称其具有焦点。当光标移出给定项目时, 称其已失去焦点。

函数 在软件开发的上下文中, 这是一个或一组表达式, 用于处理某种类型的输入并返回结果。在 Visual LISP 的上下文中, 子例程 (subroutine) 和函数 (function) 是同义词。在 Visual Basic 或 C/C++ 等其他语言的上下文中, 子例程不返回结果, 而函数返回结果。

全局 在同一名称空间中运行的, 可由所有其他变量、符号或表达式公开以供读取或写入操作的任何变量、符号或表达式。其他表达式无法访问的符号或表达式被称为局部于其父函数或表达式。

堆 一堆需要清理的垃圾, 或者为一组特定的相关表达式定义和/或其结果分配的逻辑内存地址空间。

接口 一个软件组件或服务可以连接到另一个软件组件或服务以便从另一个组件或服务请求服务或值的任何方式。这也可能涉及通过公共逻辑程序引用在任一方向传递信息。

迭代 在一组相关项目中按顺序访问项目的循环过程。迭代函数的示例包括有: (while)、(foreach)、(repeat) 和 (vlax-for)。

局部 在其父函数或表达式之外不共享或不可访问的任何变量、符号或表达式。

方法 给定对象的内置函数, 可以自动检索或修改该对象。直线 (LINE) 对象提供的方法例子包括 Move、Rotate 和 Copy。

模式 指的是在启动环境中如何显示和控制对话框表单的性质。如果对话框可以保持可见, 而用户可以继续与父应用程序的其他方面进行交互, 则称该窗体是无模式。如果可见表单在关闭之前阻止与父应用程序的其他方面的交互, 则该表单在本质上被称为模式。

无模式 参见 模式.

名称空间 分配给给定应用程序或进程的独立内存地址范围。地址范围受到保护，不会被其他地址范围访问，从而为进程在其中执行创建了一个受保护的环境。

对象 在软件开发的上下文中指的是类的实例，它提供诸如属性、方法或事件之类的内在功能，可用于与其他服务、组件或对象交互以执行某些编程任务。

对象模型 上层软件应用程序或进程中对象的逻辑分层组织。例如，Windows® 2000 有一个由 Win32 和 DCOM 或 COM+ 类环境提供的对象模型。AutoCAD® 2002 有自己的对象模型，由 ActiveX 框架提供并通过 ObjectDBX、ObjectARX、VBA 和 Visual LISP 环境公开。

工程 在 Visual LISP 中表示给定名称的程序源代码文件集合。

属性 给定对象的固有特性，可以以某种方式对该对象进行唯一标识。

提供方 公开某些功能以供其他应用程序或组件使用的软件应用程序或组件。当它实际被使用方应用程序、组件或服务使用时，它就变成了提供方。

反应器 Visual LISP 提供了一种特殊类型的软件服务，它充当 AutoCAD 应用程序会话中特定事件的侦听设备，并在截获的事件满足指定条件时选择性地执行某些任务。

递归 评估给定函数的输入，依此为达到预期结果而进行的重复处理，对同一函数进行一次或多次自我调用，直到满足最终的终止条件。

RPC Remote Procedure Call，Windows® 操作系统提供的一种远程过程调用进程，允许本地进程请求在另一台机器上创建的进程以远程执行某些操作。

安全数组 一个元素数组，其中该数组具有固定长度，不能修改以增加或减少长度（可以存储在其中的元素数量）。之所以说它是“安全的”，是因为它不能改变长度，从而减少了由于尝试从越界（超出数组末尾）的索引输入或检索元素而导致出错的可能性。

堆栈 一个可以收集对象或值，并允许以有序或顺序的方式系统地添加或删除成员的逻辑容器。

分步执行 停止程序执行并允许用户一次手动执行一条语句或一行的过程。分步执行有几种类型：下一嵌套表达式、下一表达式和 跳出。.

类型库 一种专用软件组件，用于向请求编程交互的任何其他应用程序或进程标识该对象模型的对象模型和成员对象、属性、方法和常量。

变体 一种定义为能够存储所有其他数据类型的数据类型，从而避免在从给定变量或符号分配或检索给定数据类型之前验证给定数据类型的问题。

工作空间 在编程开发环境中打开的一组活动的相关程序文件。

WSH Microsoft's Windows Scripting Host, 一种为在机器上执行脚本代码文件提供运行时支持的软件服务。**WSH** 允许脚本在 **Windows®** 名称空间中运行, 或者在调用应用程序或进程的名称空间中从编程接口运行。默认的 **WSH** 启用服务是 **CSCRIPT** (命令行界面) 和 **WSCRIPT** (图形用户界面)。

活动目录 Microsoft® Windows® Server 中, 负责架构中大型网络环境的集中式目录管理服务。

ARX 参见 **ObjectARX**。

COM+ 参见 **COM**。

DBX 参见 **ObjectARX**、**ObjectDBX**。

DCOM 参见 **COM**。

DOSLib 一个 **LISP** 可调用函数库, 提供基于 **CAD** 的 **LISP** 编程语言所不具备的功能。

导出 原文用词为 **dump**, 译者对这个词的翻译不是很确定, 一概使用导出, 当然, 并不是所有的**导出**均是对这个词的翻译, 如果你点击那个**导出**不能跳到这个页面, 那么那个**导出**就不是对 **dump** 的翻译。

Express Tools 一组可扩展 **AutoCAD** 功能的生产力工具。这些工具是出于礼节而提供, 并不受支持。**AUTODESK®** 不对其成功运行承担任何责任。

组策略 Microsoft® Windows® NT 家族操作系统的一个特性, 它可以控制用户帐户和计算机帐户的工作环境。

Jet Microsoft® Jet, 是微软针对档案型资料库所发展的资料库引擎, 目前的 **Jet** 引擎最新版本为 4.0, 并且未来在 x64 平台上将不再支持。

.NET 一种适用于 **Windows®**、**Linux** 和 **macOS** 操作系统的免费开源托管计算机软件框架。

ObjectARX 用于自定义和扩展 **AutoCAD** 的 **API**, **ObjectARX SDK** 由 **AUTODESK®** 发布, 并在 **AUTODESK®** 许可下免费提供。

ObjectDBX 参见 **ObjectARX**。

面向对象 一种具有对象概念的程序设计方法, 同时也是一种程序开发的抽象方针。它可能包含数据、特性、代码与方法。

TechNet 是面向 **Information Technology (IT)** 专业人员的 **Microsoft®** 网络门户和网络服务。

缩略语

ADO ActiveX Data Object, ActiveX 数据对象.

API Application Programming Interface, 应用程序接口.

ASCII American Standard Code for Information Interchange, 美国信息交换标准代码.

ASP Active Server Pages, Microsoft® 的第一个服务器端脚本语言和动态网页引擎.

BIOS Basic Input/Output System, 基本输入输出系统.

CAD Computer Aided Design, 计算机辅助设计.

CIM Common Information Model, 一种开放标准, 它定义了 IT 环境中的托管元素如何表示为一组通用对象以及它们之间的关系.

CLSID CLasS IDentifer, 标识 COM 类对象的全局唯一标识符.

COBOL COmmon Business-Oriented Language, 面向商业的通用语言.

DMTF Desktop Management Task Force, 硬件和软件制造商联盟.

DOS Disk Operating System, 一种古老的操作系统.

DXF Drawing eXchange Format, AUTODESK® 开发的用于 AutoCAD 与其它软件之间进行 CAD 数据交换的数据文件格式.

EED Extended Entity Data, 实体扩展数据.

FSO FileSystem Object, 文件系统功能对象.

GUI Graphical User Interface, 图形用户界面.

GUID Globally Unique IDentifier, 全局唯一标识符.

HTML Hypertext Markup Language, 超文本标记语言.

IT Information Technology, 信息技术.

JVM Java Virtual Machine, Java 虚拟机.

LISP LISt Processor language, 表处理语言, Auto/Visual LISP 是其方言.

MSDN MicroSoft Developer Network, 微软开发者网络.

MTS Microsoft Transaction Server, 一种为COM组件提供服务的软件, 可以更轻松地创建大型分布式应用程序.

OCX Object Linking and Embedding (OLE) Control Extension, 对象类别扩充组件.

OLE Object Linking and Embedding, 对象链接与嵌入.

SDK Software Development Kit, 软件开发工具包.

UCS User Coordinate System, 用户坐标系.

UNC Universal Naming Convention, 通用命名规则.

VB Visual Basic.

VBA Visual Basic for Applications.

VBAIDE Visual Basic for Applications Integrated Development Environment, VBA 集成开发环境.

VBE VBScript Enclosed.

VBS VBScript.

VLIDE Visual LISP Integrated Development Environment, Visual LISP 集成开发环境.

WBEM Web-Based Enterprise Management, 基于 Web 的企业管理, 为统一分布式计算环境的管理而开发的一组系统管理技术.

WCS International Organization for Standardization, 国际标准化组织.

WCS World Coordinate System, 世界坐标系.

WMI Windows Management Instrumentation, Windows®中用于提供共同的界面和对象模式以便访问有关操作系统、设备、应用程序和服务的管理信息.

XML eXtensible Markup Language, 可扩展的标记语言.

函数索引

ACET-STR-TO-LIST, 77

vl-acad-defun, 86

vl-acad-undefun, 86

vl-arx-import, 82

vl-bb-ref, 84

vl-bb-set, 83

vl-catch-all-apply, 41, 207

vl-catch-all-error-message, 43

vl-catch-all-error-p, 41, 43

vl-directory-files, 63, 67

vl-doc-export, 81

vl-doc-import, 81

vl-doc-ref, 83

vl-doc-set, 82

vl-every, 71

vl-exit-with-error, 44

vl-exit-with-value, 44

vl-file-copy, 64

vl-file-delete, 64

vl-file-directory-p, 64

vl-file-rename, 64

vl-file-size, 63

vl-file-systime, 65

vl-filename-base, 63, 65

vl-filename-directory, 65

vl-filename-extension, 65

vl-filename-mktemp, 66

vl-get-resources, 73

vl-list-exported-functions, 84

vl-list-loaded-vlx, 81

vl-load-all, 83

vl-load-com, 7, 92

vl-position, 70

vl-propagate, 83

vl-registry-delete, 88

vl-registry-descendents, 89

vl-registry-read, 87

vl-registry-write regkey, 88

vl-remove, 72

vl-string->list, 73

vl-string-elt, 73

vl-string-left-trim, 74

vl-string-mismatch, 75

vl-string-position, 75

vl-string-right-trim, 74

vl-string-search, 72

vl-string-subst, 76

vl-string-translate, 76

vl-string-trim, 73

vl-unload-vlx, 81, 112

vl-vlx-loaded-p, 81

vla-get-Layer, 8

vla-get-Toolbars, 193

`vla-getInterfaceObject`, 116
`vla-InsertInMenuBar`, 191
`vla-item`, 43
`vla-load`, 189
`vla-RemoveFromMenubar`, 191
`vlax-add-cmd`, 84
`vlax-create-object`, 59
`vlax-dump-object`, 11, 18, 22, 129, 197
`vlax-ename->vla-object`, 8, 100
`vlax-for`, 70, 171, 172
`vlax-get`, 23
`vlax-get-object`, 59, 208
`vlax-get-or-create-object`, 60
`vlax-get-property`, 23, 133
`vlax-import-type-library`, 28
`vlax-invoke`, 24
`vlax-invoke-method`, 24
`vlax-make-safearray`, 53
`vlax-make-variant`, 49
`vlax-map-collection`, 69
`vlax-object-erased-p`, 61
`vlax-property-available-p`, 22
`vlax-put-property`, 133
`vlax-release-object`, 61
`vlax-remove-cmd`, 85
`vlax-safearray->list`, 53
`vlax-safearray-fill`, 54
`vlax-safearray-get-dim`, 57
`vlax-safearray-get-element`, 55
`vlax-safearray-get-l-bound`, 57
`vlax-safearray-get-u-bound`, 58
`vlax-safearray-put-element`, 56
`vlax-safearray-type`, 54
`vlax-typeinfo-available-p`, 30
`vlax-variant-change-type`, 52
`vlax-variant-type`, 50
`vlax-variant-value`, 52
`vlax-write-enabled-p`, 60
`vlr-acdb-reactor`, 92
`vlr-add`, 92
`vlr-added-p`, 92
`vlr-beep-reaction`, 92
`vlr-CommandCancelled`, 106
`vlr-current-reaction-name`, 92, 102
`vlr-data`, 92, 102
`vlr-data-set`, 92
`vlr-deepclone-reactor`, 92
`vlr-docmanager-reactor`, 92
`vlr-dwg-reactor`, 92
`vlr-dxf-reactor`, 92
`vlr-editor-reactor`, 92
`vlr-linker-reactor`, 93
`vlr-miscellaneous-reactor`, 93
`vlr-mouse-reactor`, 93
`vlr-notification`, 93, 105
`vlr-object-reactor`, 93, 99, 101
`vlr-owner-add`, 93
`vlr-owner-remove`, 93
`vlr-owners`, 93, 102
`vlr-pers`, 93, 103
`vlr-pers-list`, 93, 103
`vlr-pers-p`, 93, 103
`vlr-pers-release`, 93, 103
`vlr-reaction-name`, 93
`vlr-reaction-names`, 96
`vlr-reaction-set`, 93
`vlr-reactions`, 93, 102
`vlr-reactors`, 93

vlr-remove, 93
vlr-remove-all, 93
vlr-set-notification, 93, 105, 106
vlr-sysvar-reactor, 94
vlr-toolbar-reactor, 94
vlr-trace-reaction, 94
vlr-type, 94, 97, 102
vlr-types, 94, 96
vlr-undo-reactor, 94
vlr-wblock-reactor, 94
vlr-window-reactor, 94
vlr-xref-reactor, 94

dos_exewait, 131
dos_getstring, 82
registry-tree-dump, 89
s::startup, 182

AcadProfiles, 184
AllDocs-Audit, 175
AllDocs-Purge, 175
AllDocs-Save, 175
DBX-Doc-Open, 117, 120
DBX-GetTableList, 117
DBX-TableGet, 117
DLLRegister, 117
DLLUnRegister, 117, 213
Documents-ListAll, 172
DSX-Item, 126
dump, 14
DWGSCAN, 119
errortest, 44
Excel-Get-Cell, 30
Excel-Put-CellColor, 30
excel-test, 201

fubar, 44
get-item, 43, 193
List->IntList, 126
List->VariantArray, 126
Profile-Delete, 184
Profile-Exists-p, 184
Profile-Import, 184
Profile-Reload, 184
Profile-Rename, 184
Profiles-ListAll, 186
ProgID->ClassID, 117
RegGet, 89
RegSet, 89
Toolbar-Make, 196
UpdateTemplatePath, 182
vbStrCat, 211
vbStrRev, 211
vbStrVer, 211
Xrecord-Add, 124
Xrecord-Delete, 124
Xrecord-Get, 124
Xrecord-Rebuild, 124

责任编辑：张晨南

封面设计：张晨南



定价：334.00 元