

计算机语言技术系列丛书(二)

扩展

SAMS
PUBLISHING

Extending Visual BasicTM for WindowsTM

郭爱民 译

In-depth coverage
of Dynamic Link
Libraries, custom
controls, database
add-ons, and more

Programming
techniques for
adding power
to your applications

Complete
instructions for third-
party enhancement

希望

学苑出版社

D. F. Scott

另配磁盘
disks feature
over 9M of
software from
the most popular
extension products

PROGRAMMING

计算机语言技术系列丛书(二)

Extending Visual Basic for Windows

扩展 Visual Basic for Windows

D. F. Scott 著
郭爱民 译
章 含 审校

学苑出版社

(京)新登字 151 号

内 容 简 介

本书是一本极有价值的 Visual Basic 参考用书。它深入地讲解了动态链接库、控件、数据库等内容,以及如何扩展 Visual Basic,并编写更为理想的应用程序。本书适用于各行各业的工作人员及技术人员,也适用于大专院校的师生。

需要本书的用户,请直接与北京海淀 8721 信箱书刊部联系,邮政编码 100080,电话 2562329。

版 权 声 明

Authorized translation from the English language edition published by SAMS Publishing Copyright © 1992.

Chinese language edition published by Beijing Hope Computer Company & Xue Yuan Press/Simon & Schuster (Asia) Pte Ltd Copyright © 1994.

本书英文版名为《Extending Visual Basic for Windows》,由 SAMS Publishing 公司出版。版权归 SAMS Publishing 所有。本书中文版由 Simon & Schuster (Asia) Pte Ltd 授权出版。未经出版者书面许可,本书的任何部分不得以任何形式或任何手段复制或传播。

计算机语言技术系列丛书(二)

扩展 Visual Basic for Windows

著 者: D. F. Scott

译 者: 郭爱民

审 校: 章 含

责任编辑:甄国宪

出版发行:学苑出版社 邮政编码:100036

社 址:北京市海淀区万寿路西街 11 号

印 刷:兰空印刷厂

开 本:787×1092 1/16

印 张:13 字数:299 千字

印 数:1~5000

版 次:1994 年 8 月第 1 版第 1 次

ISBN7-5077-0905-1/TP. 29

本册定价:29.00 元

学苑版图书印、装错误可随时退换

目 录

第一部分 可扩展机制

第一章 资源与库如何工作	(2)
1.1 开放式结构	(2)
1.2 调用局部 API	(4)
1.3 图形设备上下文	(6)
1.3.1 .hDC	(7)
1.3.2 .hWnd	(7)
1.4 试验上下文	(7)
1.4.1 少说,多做.....	(8)
1.4.2 指定标尺.....	(10)
1.4.3 进入不同区域之间的块传递.....	(13)
1.5 用块传递移动国际象棋.....	(18)
1.6 高级编程与低级编程.....	(27)
第二章 语言是如何扩展的	(28)
2.1 在高速路上编程.....	(28)
2.2 且慢,请先交费	(29)
2.2.1 核心中的建筑.....	(31)
2.2.2 扩展与内部 DLL 调用的比较	(31)
2.3 VB 库的形状	(33)
2.3.1 在当前项目中包括.VBX 文件.....	(33)
2.3.2 无用的礼物.....	(34)

第二部分 程序员助手与库

第三章 更新编程环境	(36)
3.1 Sheridan 系统的 VBAssist	(36)
3.1.1 “外观与感觉”的争论.....	(41)
3.1.2 培育编码文化.....	(42)
3.1.3 越过属性条.....	(43)
3.1.4 Assistant 之后是什么?	(45)
3.2 MicroHelp VBXRef	(45)
3.2.1 属性与过程显示.....	(46)
3.2.2 生成报告.....	(47)
3.2.3 逆向工程.....	(49)

第四章 在语言中增加关键字	(52)
4.1 MicroHelp Muscle	(52)
4.2 Muscle 数组	(57)
4.3 在家庭跑马场上加上跑道	(58)
第五章 探索思想过程	(63)
5.1 神经元与变量	(64)
5.2 在所有这些中窗口的作用	(67)
5.3 拧紧一个灯泡要用多少个神经元	(69)
5.4 Basic 训练	(70)
5.5 用电子显微镜观察茶叶	(73)
5.6 飞快的灰狐狸跳过了懒惰的狗	(75)
5.7 让世界更神经化	(82)
第六章 访问外部世界	(84)
6.1 打开通道	(86)
6.2 检查终端窗口	(88)
6.3 内装式文件传送协议例程	(90)
6.4 速度问题	(91)

第三部分 定制控件与图形扩展

第七章 工具箱变成玩具箱	(93)
7.1 Microsoft 的 VBTools	(93)
7.2 作者的动机	(98)
7.3 Sheridan 系统的 3D Widgets	(103)
第八章 动画与功能溶合	(108)
8.1 Desaware CCF 光标	(111)
8.2 程序员如何建立定制控件	(112)

第四部分 附加数据库

第九章 数据管理理论	(115)
9.1 数据如何开发	(115)
9.2 走出随机存取的沼泽	(118)
9.3 VB/ISAM MX 源代码	(120)
9.3.1 主关键字与次关键字	(122)
9.3.2 蕴含算术算子	(123)
9.4 在字母表的下面	(125)
第十章 如何使用数据	(127)

10.1	vxBase 有限公司的 vxBase	(128)
10.1.1	内容表——如此说	(129)
10.1.2	DOS 范围之外	(130)
10.1.3	飞机代理测试应用程序	(131)
10.1.4	浏览控件	(135)
10.1.5	内部表达式	(135)
10.2	Sequiter Software 的 CodeBasic	(136)
10.2.1	数据库的通讯模式	(138)
10.2.2	人道地使用指针变量	(144)
10.2.3	创造的奇迹	(144)
第十一章	BroaderBased 数据库	(147)
11.1	Visual Basic 的 SQL 库	(147)
11.1.1	从格言知识=“”选出的项	(148)
11.2	Quadbase 的 SQL/Win	(150)
11.2.1	执行 SQL 命令	(153)
11.2.2	嵌入式 SQL	(154)
11.2.3	复合值的问题	(155)
11.2.4	取得说明光标的值	(156)
11.3	Pioneer Software 的 Q+E 数据库	(159)
11.3.1	大奖之外的驱动程序	(160)
11.3.2	Q+E 的通讯模式	(161)
11.3.3	日志文件	(163)
11.3.4	Q+E 变量取得	(164)
11.3.5	交叉表引用	(165)
第十二章	数据库功能性的唯一性	(169)
12.1	Blue Rose Software 数据库	(169)
12.1.1	B 树索引	(170)
12.1.2	散列法	(174)
12.1.3	数据库的创立	(174)
12.1.4	使用或不使用 xBase	(176)
12.2	Pioneer Software Q+E Database/VB	(177)
12.3	控制 Object Trieve 的 Coromandel	(182)
12.3.1	工作前和工作后	(185)
12.3.2	虚拟 ISAM	(187)
12.3.3	低级工具	(187)
第十三章	开发者的见解	(190)
13.1	初步印象	(190)
13.1.1	Mark Norlsoff, Microsoft 总裁	(190)
13.1.2	Dan Appleman, Desaware 总裁兼编程领导	(190)

13.1.3	Sam Patterson, MicroHelp 公司的 VBTools 开发者	(191)
13.2	某一天你也可以成为 Windows 产品管理者	(191)
13.2.1	Bill Locke, MicroHelp 的 VBTools 开发者	(191)
13.2.2	Joe Dour, sheridon 系统合伙人	(191)
13.2.3	John Rappl, Pioneer 软件编程领导人	(192)
13.2.4	Dorai Coromandel 公司编程领导人	(192)
13.2.5	Mark Novisoff	(193)
13.2.6	Sam Cohn Sofrware 公司信息部总裁	(193)
13.3	关于 Visual Basic 的正式希望清单("Ream")	(193)
13.3.1	Crai Leach, MicroHelp 通讯库开发者:	(194)
13.3.2	Dan Forest, Sequiter Software 联合开发者	(194)
13.3.3	Tim O'pry, MicroHelp VBXRef 的开发者	(194)
13.3.4	Mark Novisoff	(194)
13.3.5	Dorai Swamy	(195)
13.3.6	Joe Dour	(196)
13.3.7	San porferson	(196)
13.3.8	Dan Appleman 的结束语	(196)
附录 A		(197)
A.1	磁盘内容	(197)
A.1.1	Ward Systems Neuro Windows	(197)
A.1.2	MicroHelp VBTools, Musde 和通讯库	(197)
A.1.3	VXBase	(197)
A.1.4	Quadbase—SQL/Win	(197)
A.1.5	Software Source VB/ISAM MX	(198)
A.1.6	Seqniter CodeBasic	(198)
A.1.7	Sheridan 3D Widgets	(198)
A.1.8	Coromandel Db Controls	(198)
A.1.9	Desaware Custom Control Faactory	(198)
A.1.10	Micro System Option 3D Graphic Tools	(198)
A.2	安装	(199)
A.3	进一步了解	(199)

第一部分

可扩展机制

第一章 资源与库如何工作

大约自一九七七年在个人计算机上开始配备 BASIC 语言用以编写程序。这时,打开计算机,就可以进入 BASIC。光标闪烁之后,输入 BASIC 命令。如果输入的命令有错误,计算机指出,可以再试。如果 BASIC 程序存储在称为盒式录音机的流上,在开机时,就可以引导进入 BASIC。一九七八年夏天,我开始用 BASIC 编写程序,当时我用一台 4K 的 TRS-80 微机替换了一个四声道仿真装置。

六月的一个夏天,我和我的朋友帕特(Pat)正着迷于在这种银白色塑料灯上运行的星际旅行游戏。我终于拯救了银河系,这里的“银河系”是指 8×8 的方格,每格为一个 3 位的数,但是在机器把我得的高分存到录音带上时,机器产生了一个语法错误? SYNTAX ERROR。在错误信息下面是代码,向我说明如何调用 BASIC。从那天起我能够改正这一错误,把我的高分存起来。

十四年后我仍然在使用它。使用代码的概念,程序可以做很多事情。在早期,程序员使用很多宝贵的资源。铅笔是个人工具库中最常用到的工具之一,当磁带用完后需要重绕时,就需要拿起铅笔。另一个有用的工具是一本精巧的书,它是由一位名叫唐·兰开斯特(DON LANCASTER)的优秀工程师设计的,书名为《十六进制数汇编》,由几百页的表组成,用于把十六进制数转换为十进制数或反过来。在可以用袖珍计算器做这项工作之前,这本书是非常有用的。

也许我要说的是,我们没有人预测编程艺术将如何发展。但是这是在说谎。我们确实预测编程的未来。我们要跟上加利福尼亚的保罗·阿尔托研究中心(PALO ALTO RESEARCH CENTER)和另一边海岸 MIT 与贝尔实验室的研究工作。我们看到的不是产品的增长,而是概念的增加。实际上,这些概念导致了计算机工业的诞生和发展。

1.1 开放式结构

计算机中的开放性原理是:计算机的每一个工作元素是整个系统的一部分,而且每一部分本身提供一项服务。这些服务主要是为计算机提供的,而不是为用户提供的。计算机的“对话”部分提供确定服务的等级和水平。计算机的这些部分组成了从用户到计算机并再返回的互相联接的链,就像在救火时,市民从水井向火场运送水桶那样。

只要确定出每一部分在哪里,可以很容易用类似的部分替换链中的一部分,当系统的其它部分参与其工作时,它可以替换或升级。没有开放性就没有现代软件产业。用户的应用程序是链中的另一元素;今天计算是一种产业,因为,可以选择想要运行的软件。由于开放性,你可以选择最适合你的工作与生活方式的指令集。

图 1.1 表明一个简单的计算链中的元素。为计算机工作的那些部分在堆的底部;为用户的那些部分在顶部。在最底部是最具逻辑性的部分——中央处理单元(CPU)。它可以为任何协处理器享用。CPU 是整个计算运算的逻辑开关板,其作用就像对话的接受方。

基本输入/输出系统(BIOS)是系统的神经中枢。BIOS 可以感觉到任何时候向计算机进行

的任何一种形式的输入(键盘,鼠标,光笔等)并作出反应,把字符或图像像素像快速莫尔斯电码那样发送出去。

磁盘操作系统(DOS)负责把那些字符放到内存或磁盘文件中去。这些文件提供给 Microsoft Windows,Microsoft Windows 的作用是为显示提供环境,并管理这些文件。通过 Windows,可以看到人的作用,比特流和无关的文件现在变成了实在的设备,可以移动和发出命令。

Windows 上面一个层次一般是用户应用程序,链在此通常转向,掉头向下。但是在图 1.1 中链还在继续,后面加上了 Visual Basic。VB 解释程序的作用是从链的底层取出基本资源,对它们命名。这样,可以访问它们,用于构造复杂指令。

VB 结构中的下一个层次也就是本书的题目:扩展 Visual Basic。它使程序员可以更方便地取得链的底部提供的资源。数据库链,256 色图形框控制,动画键和 BIOS 状态分析程序是提供的开放的资源的例子。因为开放性,可以随时把这些提供的程序加到个人计算系统中,而不会造成破坏。

在链的顶部是用户编写的应用程序。注意,链的每一级都有完善提高的余地。对于 CPU 逻辑,没有多少扩充的余地。但是到了 Windows,其中有文件、结构和图形对象。进入 Visual Basic 领域后,会遇到指令和真实的、可编程的设备;同时可以取得 VB 应用程序。

链中元素从上到下共享同样的资源,但是当资源向上处理时,改进了“包装”。例如,输入数据的字符是资源。通过链向上时,字符变成了数据域或命令的一部分。字符是由 BIOS 提供的资源。一种灰色的小东西——鼠标上的按钮是来自链的上面部分 Windows 的资源。剥去这一键的外壳,它实际上是一种在屏幕上提供命令选择的图形方法。命令主要由字符组成,这样就回到了链的下面的 BIOS 提供的资源。

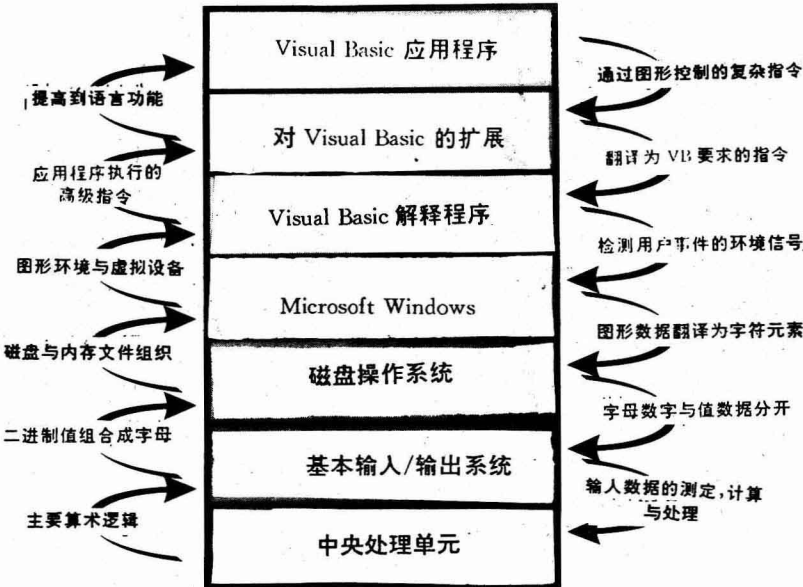


图 1.1 计算系统中的资源组成

1.2 调用局部 API

Windows 的应用程序接口(API)是一系列过程,使应用程序可以直接取得 Microsoft Windows 的操作,对话与图形函数与过程。API 构成了 Windows 与上面那些元素之间的链,向应用程序提供资源,以及应用程序的环境,如 Visual Basic。API 是以动态链库(DLLs)的形式给出的。DLL 文件可以是一系列图形资源,如:图标或 Windows 内容。然而,大部分 DLL 含有小的程序文件,可以与应用程序联接,如预先编制好的构造部分。

用 C 语言编写程序的人对库的概念是熟悉的。C 程序设计语言本身没有用户对话或数据的图形显示方面的指令。作为替代方法,C 使用函数库,这些函数由预处理器集中在一个程序中。例如,标准输入和输出函数由称为 STDIO.H 的库提供,它由预先汇编或编译的过程组成。对每一台计算机或 C 编译程序,每一过程的语法是相同的;只有处理过程本身在不同的编译程序中是不同的。在从一种编译程序换到另一种编译程序时,C 程序设计员不必重新学习使用标准 I/O 函数。

Windows API 在同样的概念下工作。对每一次顺序释放 Windows,库过程本身可以改变。重要的是 Windows 提供了,或者至少提出了,在 Windows 环境中对每一种编程或应用程序的标准化接口。例如,如果理解了 API,就可以改变光标闪烁的速率,不论是用 BORLANDC++、Windows 背景“宏”语言或 Visual Basic 都一样。

然而,与使用 Visual Basic 的 API 过程不同的是,每一过程都像 VB 关键字一样是可以直接使用的。只要在 VB 应用程序的全局模块中包括适当的说明,就可以把调用 Windows 操作系统特有资源的关键字加到 Visual Basic 的词汇表中。

下面是可由 Visual Basic 调用的动态链库中关键字:

- Kernel 一组过程,表示当 Windows 联接到计算机的操作系统并与其交换信息时的 Windows 的关键操作。
- Keyboard 处理键盘输入的过程。
- User 通常由 Microsoft 提供的一组过程,允许用户(程序员)额外调用 Windows 环境中的函数,包括表示多重联接的 Windows 过程的复合函数。
- GDI 提供直接调用 Windows 图形设备接口的过程,使用户可以更容易执行画轮廓,处理图形设备及处理动画。

用于在全局模块中把 DLL 过程说明为 VB 应用程序的一部分的表达式自然是调用 Declare。可以用下面的方法表示这一表达式:

语法一:

```
Declare Sub process $ Lib library $ [Alias secondname $ ]  
  
([ [ByVal] argument1 As type $ 1, [ByVal] argument2  
As type $ 2... [ByVal] argumentn As type $ n ])
```

语法二:

```

[ Declare Function process $ Lib library $ [Alias secondname $ ]
[
[   ([[byVal] argument1 As type $1, [ByVal] argument2
[
[   As type $2... [ByVal] argumentn As type $n])
[   As funtype $

```

只要一经说明, DLL 过程就有了名字, 或句柄, 把它与 VB 程序链接在一起。如果 DLL 过程向 VB 程序返回一个值, 这个过程就说明为一个 Function; 否则将它说明为 Sub 过程。

在前面的语法例子中, process \$ 是在动态链接库中的过程的名字。这是在 Visual Basic 过程中使用的名字。另一方面, 可以通过 Alias 选择项将 secondname \$ 赋予那个过程。Alias 经常用于对 API 过程改名, 因为在 API 过程的名字中含有下划线字符(—), 而 VB 解释程序不能正确处理下划线字符。程序员选择 secondname \$, 虽然不能在程序的其它地方都不使用 process \$。

Lib 项旁边的 library \$ 串可以设置为动态链接库的文件名, 这个库的说明过程为局部的; 也可以说明为 VB 解释程序假定为 Windows API 系统的一部分的库。Kernel、Keyboard、User 及 GDI 就是例子。这个串应用引号括起来。只要 VB 扩展库一经安装, VB 应用程序中使用的每一扩展 DLL 过程的说明语句中 Lib 项可以用于引用其文件名。下面是从应用程序 Quadbase—SQL/Win 的说明表中取出的一个例子:

```
Declare Function DqExit Lib "wqbsql.dll" () As Integer
```

其中没有参数, 因为这个函数设计要做的全部工作是“退出”主数据库管理函数。在这个例子中也不是完全 DOS 路径名; 说明假定 Quadbase 所在的目录列在启动设备的根目录中的 AUTOEXEC.BAT 文件中的 PATH 语句中。

通常扩展包中含有非常多的 DLL 过程(可能有几百个), 只要写一个 VB 应用程序, 就必须花大量时间说明每一个过程。因此, 程序员通常提供一个文本文件, 可以用 VB 控制窗口的 Code|Load 文本菜单命令把它加到 VB 应用程序的全局模块中。在这个文本文件中尽可能包括对包中的每一个 DLL 过程的说明, 在 VB 应用程序中, 其中有很多可能用不上。可能希望去掉没有用到没有什么坏处。

例子中的 Quadbase 的说明没有把自变量传给过程 DqExit()。在 VB 说明语句中, 自变量和参数必须列表说明, 表用括号括起来, 之间用逗号分开。下面是一个更详细的例子, 它用 Quadbase 过程把一个结构化 Query 语言命令传递给 Quadbase 数据库的查询处理程序:

```

[ Declare Function DqSend Lib "wqbsql.dll" (ByVal TaskID As Long,
[   ByVal Cmd $, Hostv As Any, Handle As Any) As Integer

```

如果 ByVal 项出现在被说明变量的前面, 只把这个项的值传递给 DLL 过程而不是作为 Visual Basic 变量传递其指针或其本身。因为 DLL 过程不能识别 VB 变量名, 这些过程不能改变工作时作为参数接受的变量的值。通过 ByVal 说明传递给过程的大部分值也是如此, 因

此,不能期望参数的值或内容改变或由过程来改变。

在前面的例子中,当 TaskID 传递给 DqSend() 函数 ByVal 时,函数接受四个字节(因为它被说明为 As Long),其中第一个表示与 VB 变量 TaskID 相同的值。变量本身不传递。当 Cmd \$ 传递 ByVal 时,Cmd \$ 内容的一个副本在内存中的地址传递给函数。然后从那个地址开始读字符,一次读一个字符,直到见到一个空字符,或其 ASCII 代码值为 0(零)的字符。这个字符是“结束标志”,它表示一行的结束。

重要的是,传递给 DLL 过程的每个自变量的变量类型在那个过程的 Declare 语句中必须提到。Visual Basic 支持的最常用的类型是:Integer、Long(integer)、Single、Double、String、和 Currency。在有的例子中,可以使用复杂变量类型,它们要在全局模块前用 Type 说明符说明。在有些情况下,过程的输入变量可以不是任何特定类型;因此,在这种情况下 Visual Basic 支持一个特殊的类型说明 As Any。例如,DLL 过程从数据库返回到数据域的元素可以是人名(String),美元数量(Currency),或一个值(3.1415927)。在这种情况下,不能预知将是什么类型,但是至少可以事先把这种情况通知 VB 解释程序。其它任何 VB 语句不支持 Any 类型。然而,它支持传递给 DLL 过程的自变量,但不支持返回值。

1.3 图形设备上下文

在 Visual Basic 的词汇库中,接受输出文本的图形对象通常是一个文本框、一个标号、或偶尔为一台打字机。在 Windows 图形设备接口的范围内,每一个图形设备都以某种形式作为数据输出或显示的元素。由图形设备接口(GDI)调用的所有设备都以统一的方式调用,并且只要可能,输出到某一设备的过程一般都给予同样的名字和语法,这样,程序员就不会混淆。GDI 知道如何区分各种输出,如输出到窗口,输出到通常的屏幕,输出到打印机,以及输出到可能联接的 LCD 彩色投影机。

Windows 的 GDI 对所有由 Windows 调用的图形设备维护一个独立的上下文。屏幕与窗是两个不同的设备;这可能与实际不符,但是在 Windows 中,这样的区分是必须的。这就是为什么要把那些上下文维护放在首要位置上。

设备的上下文是内存中数据结构段、定义段用途的编码指令与对其在图形环境中作用的操作的组合。

在 Visual Basic 编程中,上下文这个术语通常是指一个过程的数据元素的与其它程序的元素有关的部分。术语设备上下文也同样;每一个潜在的图形输出接受者,不论是窗口还是绘图仪,都由 Windows GDI 给出设备上下文。这一上下文通过其句柄调用。在这种情况下,“设备上下文的句柄”的缩写是 hDC,其中 h 为小写。虽然设备与其内容是常数,但是随着其次序的不断变化,句柄号也可能在变化,句柄代表的内存区域也在变化。可以看作“虚拟终端”的 Visual Basic 的图形对象有两个与之相关的参数:.hDC 和 .hWnd。(因为框架背景和图像框可以通过 .Print 方法接受文本,这样的语句原来用于终端输出,因此这些对象称为“虚拟终端”。)

1.3.1 .hDC

作为虚拟终端的图形对象,.hDC 参数是一个由 GDI 自动给对象设定的数。这个数是对象设备上下文的句柄。这一参数用于引用图形对象,其中调用 DLL 过程,这个过程的输出区域就是它所引用的图形对象——它的前件。这一参数的值不作为变量传递给 DLL 过程;原因在于,变量赋值与调用发生之间前件对象的 hDC 可能会改变。这一参数任何时候都不能由程序员设定。

1.3.2 .hWnd

同样,对每一个出现在屏幕的窗口,GDI 为其维护一个窗口识别号。这个号也可以认为是窗口的句柄,因为 Visual Basic 维持.hWnd 参数。

框架的.hWnd 参数自动设置为框架窗口的窗口识别句柄,窗口识别句柄是在窗口建立时由 GDI 给出的。这一参数的值不应作为变量传递给 DLL 过程,因为在变量赋值与调用发生之间前件对象的 hDC 可能会改变。这一参数任何时候都不能由程序员设定。

1.4 试验上下文

为了理解如何利用这一过程合理地共享资源,考虑几个使用非扩展 Visual Basic 调用 Windows API 的例子。大量地使用 VB 扩展的例子以后会看到,这里假定清楚 API 的工作,至少是对次要的扩展。首先看一个简单的例子,其结果马上就可以看到。在 Windows GDI 中,库是在窗口画一个椭圆的过程。在全局模块中有这个过程的说明:

```
Declare Function Ellipse Lib "GDI" (ByVal hDC As Integer, ByVal X1 As Integer, ByVal Y1 As Integer, ByVal X2 As Integer, ByVal Y2 As Integer) As Integer
```

这个 GDI 过程是一个函数,因为它返回一个整数值;这个值在 Declare 指令的后面说明为整型。函数的标题在关键字 Function 之后。它带三个参数,第一个是设备上下文的句柄 hDC。这个句柄告诉 GDI,在 Windows 的工作空间中的哪一个窗口包括这个椭圆。其余的参数是两对坐标,表示包含这个椭圆的一个不可见矩形的左上角和右下角;即矩形的边与椭圆相切。

在对这个函数进行了说明之后,就可以发出下列指令:

```
c% = Ellipse(Form1.hDC, 50, 50, 100, 100)
```

由于设备上下文的句柄随时可能改变,把它的值赋给一变量,然后把它作为第一个参数传递是不安全的。因此,直接传递 Form1 的参数 hDC。然后传递两个(相对屏幕点)坐标对,50,50 和 100,100 做为椭圆的左上角和右下角。这个函数和值在 c 中返回。GDI 图形函数返回无符号真/假的整型值,表示函数的工作是否成功;所以,如果椭圆正确画出,c=1。

这时可能要问,为什么要用 Windows API 编写程序。首先,不论是用 Visual Basic 编写程序,还是用某种 C++ 编写程序,API 都以同样的方式访问。只要选择高级语言,就有机会使用 API 语句。

第二,除了 Visual Basic 中的外,API 中还有很多函数和过程。一个在构造上与画椭圆相关的 GDI 函数是 RoundRect,它对指定的上下文画一个圆角矩形。在 Kernel 库中可以调用 Windows 的低级过程。当作为用户与 Windows 通讯时,采取的每一个动作都会启动一个标准事件系列。例如,当激活一个滚动条时,向系统的一部分发出信息,希望改变逻辑值;向另一部分发出信息,屏幕上的某一灰色框应该移动位置;并通知另一部分,与滚动条有关的窗口中的内容应该更新。使用 Visual Basic,程序员能够建立很多这种标准事件序列的例子。

然而,如果希望应用程序窗口做其它动作,程序员使用 Windows API 可以改变这一序列。例如,在移动滚动框时,滚动条改变颜色,进一步,在装载 Windows DLL 时向某个声音库发出信息,使滚动条可以发出声音。对于 API,不必局限于编程语言所加的限制,借用一个容易引起误解的词——像初学者那样。

使用 Windows API 有利于编程的第三个原因是,其它应用程序同样可以调用 API。例如,可以从 Windows 的 Microsoft WORD 和 Microsoft EXCEL 调用 API 函数。API 可以像一个众所周知的结那样,把几个 Windows 应用程序和 Visual Basic 应用程序连接起来,让它们共同完成重要的工作任务。这就是超级应用程序的目的:把几个程序连接在一起共同处理一项工作。

1.4.1 少说,多做

下面是一个 User 库中的函数,可以从中发现某种用途:

```
Declare Function IsWindowVisible Lib "User" (ByVal hWnd  
As Integer) As Integer
```

如果在心里想有一个框架,其 .FormName 是 Stars 和一个过程(如果没有看到这个框架,千万不可以向 Stars 输入任何东西),现在,可以调用下面这个函数:

```
vis%=IsWindowVisible(Stars.hWnd)
```

然后可以根据 vis% 的结果值执行一个过程。然而,由于 IsWindowVisible() 的函数值不是 1 就是 0,用它作为语句中条件表达式的值可能更好,如:

```
If IsWindowVisible(Stars.hWnd) then  
.  
.  
.  
End If
```

在比较中不需要加入数学算子。如果窗口在当时是不可见的,函数返回 0,语句不执行。但

是,如果窗口是可见的,函数返回无符号真值 1,这应该是一个带减号的负值,或者布尔值的 NOT,从 Visual Basic(-1)的观点看导出逻辑真值。

现在假设 VB 应用程序需要向 Windows 安装一组文件,或调用 WIN.INI 文件(这是一个描述 Windows 副本的标准操作参数的文本文件)。但是,不是人人都用相同的名字把 Windows 副本存储在目录中;因为我对其它版本的 Windows 感兴趣,我的 Windows 3.1 存储在目录 C:\WIN31 中。存储 Windows 文件的缺省目录是 C:\Windows,尽管不是都选择使用它。

在 Kernel 库中有两个函数,其目的是得到 Windows 在宿主计算机中的位置。可以用下列方法在 VB 应用程序的全局模块中说明它们:

```
Declare Function GetWindowsDirectory Lib "Kernel"  
(ByVal lpBuffer As String, ByVal nSize As Integer)  
As Integer  
Declare Function GetSystemDirectory Lib "Kernel" (ByVal  
lpBuffer As String, ByVal nSize As Integer)As  
Integer
```

接下来的工作是建立两个字符串,其内容将传递给刚刚说明的两个 API 函数。这两个字符串将作为缓冲区。由于 API 不能建立 Visual Basic 变量,必须自己做这项工作。建立两个空白字符串后,就为前面的几个函数留出了地方。下面是预备指令:

```
wd$ = String$(255," ")  
sd$ = String$(255," ")
```

这样就建立了两个字符串变量区域,每个为 255 个空格,空格用“ ”表示。顺便提一下, DLL 函数返回的字符串的最大长度是 255 个字符。现在就可以调用这几个过程了:

```
s1% = GetWindowsDirectory(wd$,len(wd$))  
s2% = GetSystemDirectory(wd$,len(wd$))
```

每个 DLL 函数的第一个参数是空白字符串,第二个参数是字符串的长度(现在是 255)。由于这一长度是个变量,而 DLL 函数的操作是“盲目的”——应该记住,DLL 函数在开始时不知道 Visual Basic 的存在——因此 DLL 函数必须知道其工作空间有多大。在这里,wd\$ 的内容是 C:\WIN31,SD\$ 的是 C:\WIN31\SYSTEM。

对下面的几段你可能感到困惑。通常,在写 Sub 或 Function 过程的说明时,括号中的参数前用 ByVal 项说明时,这一般意味着将值或字符串内容传递给过程的变量不应由这个过程修改或改变。换一种方式说,ByVal x As Integer 意味着变量 x 可以传递它的整数值,但不是 VB 变量本身。如果 q 具有 x 传递过来的值,q 在调用函数的说明中代表 x,那么函数完成后,x 值的改变通常在 q 的值中反应出来。若在调用说明中使用 ByVal x,通常 q 的值变得不受影响。

这一例外与用 ByVal 把值传递给 DLL 过程有关,特别是那些说明为 Function 的过程。这是因为动态链接库不能识别这见到的是 VB 变量本身,还是传递的值。(然而,下面就可以看到,DLL 能够识别复合结构)。DLL 过程可以用一种 Visual Basic 做不到的方法改变变量的值或内容。因此,虽然经常把值传递给过程,Declare 语句可以确保传递的变量,如前面用到的缓冲字符串,可以由 DLL 过程的结果改变。那么为什么使用 ByVal? 因为 DLL 过程不能使用 VB 变量,只能使用值或字符串内容。DLL 过程不能识别变量,但是变量将能识别过程提供的最后结果。

ByVal 是最常用来识别哪个数据元素作为输入传递给函数过程。不带 ByVal 的任何其它变量都不能作为 Function 过程的输出变量,除了 Function 本身的变量值之外。下面是一个例子,同样是取自 Quadbase—SQL/Win:

```
Declare Function DqGetColumnDef Lib "wqbsql.dll" (ByVal TaskID  
As Long, ByVal TblName $, ByVal ColumnNumber As Integer,  
ColumnName As Any, Dtype As Any, Length As Any, Decimal  
As Any) As Integer
```

这个 Function 的用途是识别数据表中指定列数据的信息。本书的后面将会看到这个过程的细节。在任何情况下都可以发现用哪一个变量作为过程的输入,因为它们接在 ByVal 项的后面。输出变量(非 ByVal)列在自变量表的后面。但是,仍然不知道这些变量的内容或值将是什么,但是,在这里对它们做了说明之后,就为它们保留了空间。因为它们没有值,就不能用值把它们传递给 DLL 过程,只能传递它们本身,所以这个过程知道用以传送返回值和内容的适当地址。这就是 ByVal 为什么如此关键。

1.4.2 指定标尺

本节含有一个简单的使用 Windows API 说明的 VB 应用程序。Visual Basic 自己不能确定 VB 所驻留的计算机的参数,如显示器的每英寸中点的当前比率。假定希望在框架中放一个标尺,并且希望标尺显示真实的英寸(或使 Windows 尽可能接近实际)。可以使用 1,440 个缇等于“一逻辑英寸”的比率,但是考虑到 Visual Basic 不知道显示器究竟有多大,这样可能非常不精确。Windows 自己更希望知道系统中安装的元素的相关尺寸,因为它正在处理屏幕和其它设备的驱动程序。假如未设置这一“逻辑英寸”,Windows 可以确定每英寸有多少个点。这时需要将框架中的对象的 ScaleMode 设置为 3—Pixel,但这不是问题。

对于 Windows 中安装的每一图形输出设备,不论是屏幕,屏幕中的窗口还是打印机,图形设备接口都在内存中维护一个设备功能表。可以用 API 函数修改这个表。实现这一功能的最方便的方法是,在全局模块中加上几行程序:

```
Global Const DRIVERVERSION = 0           'Device driver version  
Global Const TECHNOLOGY = 2             'Device classification
```