实用编程百例丛书

PowerBuilder 9.0

实用编程 100例

Programming Practically

冉林仓 侯高岚 等编著

- 遵循理论与实践相结合的原则，学以致用，使您的编程水平更上一层楼。
- 结构清晰，按照实例说明、技术要点、实现步骤、归纳注释的顺序进行阐述。
- 本书面向具有一定PowerBuilder编程基础的读者。
- 本书附光盘一张，包含书中所有的源程序。

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

PowerBuilder 9.0 实用编程 100 例

冉林仓　侯高岚　等编著

中 国 铁 道 出 版 社

2004 • 北京

内 容 简 介

本书遵循理论与实践相结合的原则，分门别类地讲解了100个各具特色的实例，主要内容包括API函数调用、用户界面设计、辅助程序功能设计、网络和通信开发、数据窗口、报表打印、Web应用开发、ADO和报表控件处理等。

本书结构清晰，按照实例说明、技术要点、实现步骤、归纳注释的顺序进行阐述，面向具有一定编程基础的读者，通过本书的学习使读者的编程水平更上一层楼。书中另附配套光盘一张，包含了书中所有的范例程序。

图书在版编目（CIP）数据

PowerBuilder 9.0实用编程100例/冉林仓，侯高岚编著. —北京：中国铁道出版社，2004.4
（实用编程100例）
ISBN 7-113-05876-0
Ⅰ.P… Ⅱ.⑴冉…⑵侯… Ⅲ.数据库系统-软件工具，PowerBuilder 9.0 Ⅳ.TP311.56

中国版本图书馆CIP数据核字(2004)第029178号

书　　名：PowerBuilder 9.0实用编程100例
作　　者：冉林仓　侯高岚
出版发行：中国铁道出版社（100054，北京市宣武区右安门西街8号）
策划编辑：严晓舟　郭毅鹏
责任编辑：苏　茜　黄园园　卜照斌
封面设计：薛　为
印　　刷：北京兴顺印刷厂
开　　本：787×1092　1/16　印张：23　字数：539千
版　　本：2004年5月第1版　2004年5月第1次印刷
印　　数：1—5000册
书　　号：ISBN 7-113-05876-0/TP·1194
定　　价：39.00元

前　言

PowerBuilder 是第一个基于商业开发人员的面向对象编程（OOP）的应用程序，是一种“快速构建商业应用程序”的开发工具。自 1991 年被首次推出到今天，PowerBuilder 已有长达 13 年的发展历史。目前，全球已有超过上百万的 PowerBuilder 忠实追随者。由于 PowerBuilder 卓越的、高效的开发性能和倍受推崇的易用性，在国内外拥有无数的成功应用，广泛地应用在世界各地的银行、电信、医疗保健、保险等行业中。

相对于以前的版本，PowerBuilder 9.0 的发布是 PB 产品发展史上的又一个重要里程碑。它的进步是革命性的，它给开发人员带来的绝对是令人震撼的惊喜。它带来了许多最新的企业开发所需要的功能，具有划时代的意义。

PowerBuilder 9.0 是一个 4GL 平台，它集设计、建模、开发、部署、管理等各项功能为一体。可以全面支持 Internet 的开发，而不再是局限于 Client/Server 框架下的 4GL 平台。PowerBuilder 9.0 提供对 J2EE 和 Microsoft.NET 开发环境的支持功能，并且与 PowerDesigner 更紧密地结合。它不仅实现了对 XML、JSP、.NET 以及 Web Services 的支持，满足企业级应用的需求，而且可以实现对手持设备的应用开发。PowerBuilder 9.0 将使这个古老的产品焕发新的生命力，使之成为具有高度集成性的新一代开发平台。

PowerBuilder 9.0 包含以下新的特征：

支持快速应用开发的 JSP 编辑器；全面支持 XML；PBNI（PowerBuilder Native Interface）本机接口；EJB Client；Web Service 等增强功能。

为了满足各个层次开发人员的需要，本书对内容结构进行了精心编排，对实例进行了认真的筛选，力求语言朴实、层次分明、循序渐进。这些实例主要用于解决开发中的实际问题，具有一定的代表性，但是不会面面俱到，读者可以结合这些实例，参考其他资料，达到融会贯通、触类旁通的目的。

书中介绍时只提供关键代码，详细代码可以参考本书的配套光盘。

参与本书编写的人还有尹建民、冯冬梅、薛年喜、于丙超、徐日强、刘旭、梁斌、张俊岭、李志伟、刘伟、李子婷、张海霞、李龙、周松建、于华芸、赵磊、杨龙、韩涛等，在此一并表示感谢。

由于水平有限，加上时间仓促，书中难免存在不妥之处，恳请读者指正。光盘中的范例程序已经过仔细测试，都能够正常运行，如果您在编译时遇到了什么问题，欢迎垂询 microran2002@sohu.com 或 yujinshan@263.net。

编　者

2004 年 4 月

目　录

第一部分　API 函数调用

第二部分　用户界面设计

第三部分　辅助程序功能设计

第四部分　网络和通信开发

第五部分　数据窗口

第六部分 报表打印

第七部分 Web 应用开发

第八部分 ADO 和报表控件处理

第九部分 其他应用

第一部分　API 函数调用

本章主要介绍如何在 PowerBuilder 中调用 Windows SDK API 函数，并给出几个比较简单的调用实例。

实例 1

使用动态链接库扩展 PowerBuilder

实例说明

PowerBuilder 主要用于开发面向 Windows 平台的窗口应用程序，但是 PowerBuilder 本身采用的是一种开发效率高但是运行效率低的 PowerScript 脚本语言，这种语言和 Visual Basic 采用的解释语言一样，对系统底层的调用往往不尽人意或者是力不从心。很多方面需要调用 Windows 系统提供的 API 函数进行扩展，熟悉动态链接库 API 函数调用，对增强程序的功能十分有益。

这些 API 函数即可以是操作系统提供，也可以是其他开发工具生成的动态链接库。在调用这些函数之前，必须对这些函数进行声明，声明的格式类似于 Visual Basic 语言，主要在声明中指出采用动态链接库的文件名称、函数名称、接口规范（输入参数的类型列表、返回值的类型）等等。

下面这个实例是采用 Windows API 函数调用实现的一个播放声音的实例，程序运行界面如图 1-1 所示。

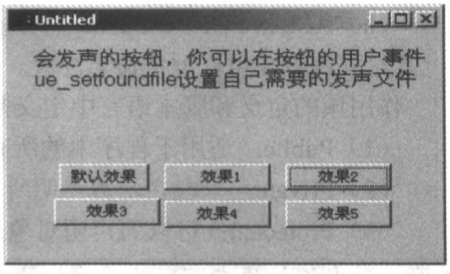

图 1-1　会发声的按钮

技术要点

- 主要了解如何声明和调用动态链接库函数，即介绍调用动态链接库函数的主要步骤。
- 根据声明函数的使用范围，动态链接库可以大致分为全局有效和局部有效两种情况。这非常类似于 Visual Basic。在 Visual Basic 模块中定义的 Public 类型的 API 函数，可以在程序代码的任何部分使用。而在个别窗体中采用 private 声明的 API 函数则只能在所在窗体的代码中使用。同样在 PowerBuilder 中也有类似的情况，全局作用域的 API 函数可以在应用程序的任何代码中使用。而局部作用域的函数则只能在其所在的窗口、菜单和用户定义的对象中使用。

外部函数是这样声明的。

```
{ access } FUNCTION returndatatype name ( { { REF } datatype1 arg1,
..., { REF } datatypen argn } ) LIBRARY "libname"
ALIAS FOR "extname"
```

外部过程声明语法如下。

```
{ access } SUBROUTINE name ( { { REF } datatype1 arg1, ...,
{ REF } datatypen argn } ) LIBRARY "libname"
ALIAS FOR "extname"
```

学过 Visual Basic 的用户都知道过程和函数的区别，过程没有返回值，实际上它们对应

于 C++中 void 返回值类型的函数。

在这个声明中具体关键字的定义是这样的：

（1）access：（可选）：只能用于局部的外部函数调用。它通过使用 Public、Protected、Private 三个关键字中的一个来指定局部外部函数的作用域。缺省时采用 public 关键字。

（2）FUNCTION /SUBROUTINE：这个关键字用于指定调用的类型。它决定着能否使用返回结果。一般地，如果一个函数返回了一个具体值，应该声明 FUNCTION；如果没有返回任何值或者返回 void，则应声明为 SUBROUTINE。

（3）returndatatype：指定函数返回值所属的数据类型。

（4）name：动态链接库中函数或者过程的名称。

（5）REF：指示采用引用方式来传递入口参数，这意味着 PowerBuilder 后面的脚本可以使用函数调用过程中存储在这个参数中的值。

（6）datatype arg：函数和过程调用中的参数类型和参数名称，这个列表要求和 DLL 中的函数实现完全匹配，这里的完全匹配指的是参数顺序和类型。每一个数据类型的前面可以加上 REF 前缀。

（7）LIBRARY "libname"：这个 LIBRARY 关键字通过后面跟的字符串来指示调用函数所在的动态链接库名称。

（8）ALIAS FOR "extname" (可选)：这个关键字用于给调用的函数起一个别名。通过这种方法可以解决命名冲突。

作用域的定义和脚本语言中定义是一样的。

（1）Public：适用于程序中的所有脚本。

（2）Private：只能在函数声明所在对象事件代码中使用。对象的派生类也不能使用。

（3）Protected：可以为声明对象及其派生类使用。

注意：动态链接库在下列位置才能被正常调用：
应用程序所在的当前目录；Windows 目录；Windows 系统目录；系统环境变量指示的目录。

实现步骤

1. 首先用户需要一个工作区，并在工作区中添加应用程序。
2. 然后在应用程序中添加一个窗体，不妨设窗体的名称为 w_PlaySound。
3. 从 commandbutton 派生一个用户对象 uo_command，在这个对象中定义一个字符串类型的变量，用于存放播放的声音文件，并在这个对象中声明调用的 API 函数。

```
// playsound  函数声明
 FUNCTION boolean sndPlaySoundA (string SoundName,
    uint Flags) LIBRARY "WINMM.DLL"
 FUNCTION uint waveOutGetNumDevs () LIBRARY "WINMM.DLL"
```

对象的单击响应代码为。

```
This.TriggerEvent("ue_setsoundfile")
//开始播放 Wav
ulong lul_numdevs
lul_numdevs = WaveOutGetNumDevs()
If lul_numdevs > 0 Then
```

```
sndPlaySoundA(is_soundfile, 1)
End If
```

4. 然后向窗体添加多个用 uo_command 实现的对象。并在这个对象的 ue_setsoundfile 事件中设置播放的对应文件，如图 1-2 所示。

5. 最后设置应用程序的 open 事件，使它运行以下脚本。

```
open(w_playsound)
```

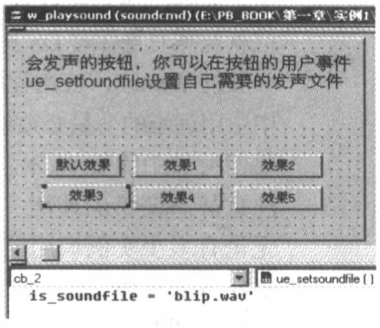

图 1-2　创建用户界面

归纳注释

在 PowerBuilder 的 script 中调用 DLL 中的函数，默认情况下以传值方式来传递参数，即 PowerBuilder 作一个备份，然后通过堆栈将这份拷贝传递给函数。如果用户希望 DLL 中的函数可以改变调用参数的原值，就可以通过传递引用（passed by reference）的办法来传递参数，即在参数类型前面加 REF 关键字。

大多数 DLL 都是采用 C 或者 C++编写的，PowerBuilder 和 C/C++存在很多差异，所以用户使用时要了解以下规则：

在 MS Windows 中，一个 DLL 在被载入内存后，只会产生一个实例，不会因为多个程序使用同一个 DLL 而在内存中产生多个 DLL 副本。每个 DLL 只有一个最大为 64KB 的数据段。缺省情况下，PowerBuilder 都是使用传值法来传递参数。

在 PowerBuilder 中使用的数据类型与 C 语言支持的数据类型不尽相同，C 中不支持的数据类型应在调用前先行转换。

对于结构，要在 C 和 PowerBuilder 中做相等的说明。

PowerBuilder 不支持函数指针的传递，因此在 PowerBuilder 中不能使用回调函数（callback funcion）。如果 DLL 的参数需要空指针（NULL），用户可以向函数传递一个值为 0 的长整型。

Windows 中使用的有些数据类型 PowerBuilder 中并不支持，但一般在 C 的预编译器中用 TYPEDEF 作预定义，同时 PowerBuilder 接口也应当作适当转换。

使用 DLL 的常见错误和需要注意的地方：

1. 导致保护性错（general protection fault）

   在 Windows 中，如果用户企图访问不是属于用户的应用程序的内存将导致保护性错。导致保护性错的原因可能有以下几点。

   1）向 DLL 中的函数传递了不正确的参数。这种错误是比较难调试的，因为 PowerBuilder 的调试器不能跟踪到 C 程序中。用户可以通过在 PowerBuilder 中使用 MessageBox 函数显示调用参数的方法来检查参数传递的正确性。或者采用 WritePrivateProfileString 函数把运行的中间结果写入到一个配置文件中。

   2）C 中对数组的访问超出了 PowerBuilder 中申请的边界。在 C 中是不作数组边界检查的，这可能是导致保护性错的最常见的原因。

   3）使用了已经释放的内存指针。用户最好把已经释放的内存指针置为 NULL，以便在使用前进行判断。

2. 使用远指针

在C中，所有的静态变量和全局变量都是在程序的数据堆中分配的，其他变量都是在栈中分配的。DLL可以有自己的数据段，但是它没有堆栈段，使用的是调用程序的堆栈。这就意味着寄存器DS指向的是DLL数据段，SS指向PowerBuilder应用程序的堆栈。而一般的Windows应用程序中，DS和SS是相同的，用户可以使用近指针，但在调用DLL中引用远堆的变量必须使用32位的远指针。如果使用任何与内存寻址有关的C函数，都要使用C中的far版本。例如字符串拷贝函数，应该用_fstrcpy而不要用strcpy。

3. 注意静态变量的使用

无论有多少实例调用同一个DLL，在内存中只有一份DLL代码。由于Windows是多任务的环境，因此DLL中的静态变量可能由于其他实例对此DLL的调用而改变。

4. 不要试图共享文件句柄

在Windows环境下，不可能在应用程序和DLL间共享文件句柄。每个应用有各自的文件句柄表，如果两个应用通过一个DLL来访问同一个文件，它们必须分别打开这个文件。

5. 及时释放使用过的资源

在API函数调用的过程中，会得到大量的句柄，这些句柄包括文件句柄、窗口句柄、设备对象句柄，这些句柄不再使用时，一定要把它们关闭，否则就会造成内存泄漏。比如用户的DLL中使用了GDI对象，一定要及时释放它们，避免内存泄漏。否则会使Windows因申请GDI资源失败而死机；例如用户建立了一个逻辑字体或逻辑笔，在使用完后，要用DeleteObject来删除它。

API函数调用和COM组件编程是本书的重点，贯穿本书的始终，用户应该仔细阅读本节内容。

任务状态区图标的创建

实例说明

任务状态区常常通过一个小图标来标识程序的当前运行状态，同时这个图标还可以显示一个快捷菜单，用于激活和显示活动窗口。

程序运行结果如图 2-1 所示。

图 2-1　任务状态区图标的显示

技术要点

利用动态链接库的输出函数 Shell_NotifyIcon 实现任务状态区图标的创建、修改和添加。

实现步骤

1．　新建项目

在创建项目时，用户首先准备一个用于在任务状态区中显示的图标文件(.ICO)，并创建一个主窗体。这个窗体构成应用程序的主窗口。用户可以在主窗体中加入任何控件。比如设置几个按钮和图片。如图 2-2 所示。

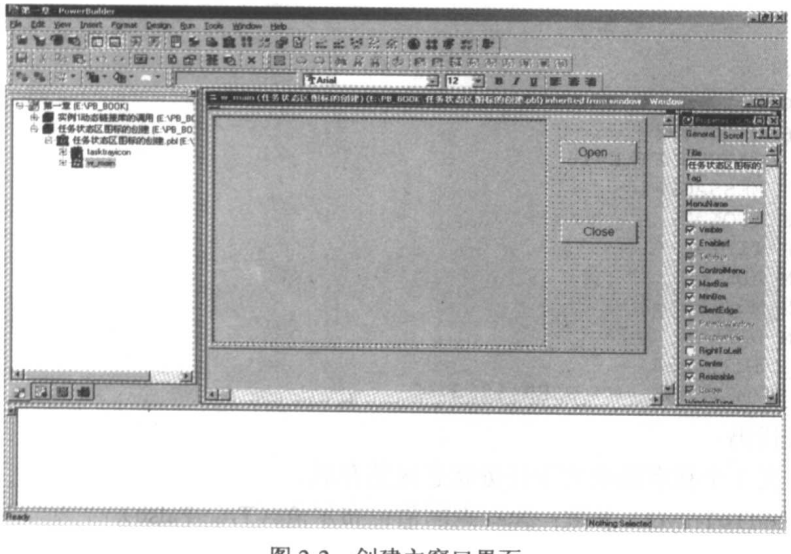

图 2-2　创建主窗口界面

2. 函数声明

由于 PowerBuilder 本身并不提供用于在任务状态区中显示图标的函数，但用户可以通过 Windows API 调用来实现，有关在 PowerBuilder 调用相应的 API 函数如下（定义在全局外部函数中）。

```
Public Function Integer Shell_NotifyIcon (Long dwMessage, Any lpData) Library
"shell32" Alias For "Shell_NotifyIconA"
```

这个函数用于维护任务状态区的图标。

```
Public Function Long LoadImage (Long hInst, String lpsz, Long un1, Long n1,
Long n2, Long un2)
Library "user32" Alias For "LoadImageA"
```

这个函数用于从磁盘或者程序资源中加载一个图像。

```
Public Function Long DestroyIcon (Long hIcon) Library "user32" Alias For
"DestroyIcon"
```

回收图标资源。

```
Public Function Long SetForegroundWindow (Long hwnd) Library "user32" Alias
For "SetForegroundWindow"
```

设置把指定的窗口显示到前台桌面中来。

```
Public Function Long OpenIcon (Long hwnd)  Library "user32" Alias For
"OpenIcon"
```

然后定义一个结构类型如下。

创建一个名称为 notifyicondata 的结构，定义如下。

项目	数据类型
cbSize	Long
hWnd	Long
uID	Long
uFlags	Long
uCallbackmessage	Long
hIcon	Long
szTip	any

常量声明。

```
long hIcon
long IMAGE_ICON=1
long LR_LOADFROMFILE=16
long NIF_MESSAGE=1
long NIF_ICON=2
Long NIF_TIP=4
long NIM_ADD =0
long NIM_MODIFY =1
long NIM_DELETE   =2
long WM_USER=1024
long WM_MYMESSAGE =WM_USER+1000
```

3. 编写代码

先定义 3 个私有函数实现任务状态区的存取。

1）AddToTray()，参数：无；返回值：布尔值（True/False）

```
Any nid
```

```
if hIcon = 0 then
   //加载图标
   hIcon = LoadImage(0, "Halloween.ico", IMAGE_ICON, 0, 0, LR_LOADFROMFILE)
end if
if hIcon = 0 then
   MessageBox ("错误", "不能加载图标!")
   Return FALSE
else
   long flags=NIF_ICON+NIF_TIP+NIF_MESSAGE
   nid = SetNotifyIconData (Handle (This), 0,flags, WM_MYMESSAGE, hIcon,
"TestTip")
   Shell_NotifyIcon (NIM_ADD, nid)
   Return TRUE
end if
```

2）RemoveFromTray ()，参数：无；返回值：无

```
Any nid
nid = SetNotifyIconData (Handle (This), 0,NIF_MESSAGE + NIF_ICON + NIF_TIP, 0, hIcon, "")
Shell_NotifyIcon (NIM_DELETE, nid)
if hIcon <> 0 then DestroyIcon (hIcon)
hIcon = 0
```

3）SetNotifyIconData (Long hWnd, Long ID, Long Flags, Long CallbackMessage, Long Icon, String Tip)

```
Char MyTip [64]
NotifyIconData NidTemp
NidTemp.cbSize = 88 // 结构的长度
NidTemp.hWnd = hWnd
NidTemp.uID = ID
NidTemp.uFlags = Flags
NidTemp.uCallbackMessage = CallbackMessage
NidTemp.hIcon = Icon
MyTip = Tip + Char (0)
NidTemp.szTip = MyTip
return NidTemp
```

4. 最后，在窗口的有关事件中输入函数。

Open 事件。

```
if AddToTray () then Visible = False
```

Close 事件。

```
RemoveFromTray ()
```

CloseQuery 事件。

```
if CanClose then
    Return 0
else
    Visible = False
    Return 0
End if
```