第15章　其　　他

本章收集了其他一些有用的实例，范围从使用定时器到闪烁一个窗口等。这些实例包括：

实例60　创建定时器，本例将演示如何使用 MFC 类创建定时事件。

实例61　播放声音，本例将演示在应用程序中播放 .wav文件。

实例62　创建VC++宏，本例将回顾由MFC支持的宏语法，它可作为编写小函数或者内联函数的一种选择。

实例63　使用函数地址，本例将演示如何传递类的成员函数的地址，实现间接调用。

实例64　处理二进制字符串，本例中将扩展 MFC的字符串类使其可以处理二进制的字符串。

实例65　重新启动计算机，在本例中将看到如何编写程序重新启动系统。

实例66　获得可用磁盘空间，本例将演示如何检测磁盘上的剩余空间。

实例67　闪烁窗口和文本，本例将演示如何闪烁文本信息或者窗口的标题来获得特殊效果。

15.1　实例60：创建定时器

1. 目标

了解已经过去了多少秒的时间。

2. 策略

使用CWnd::SetTime()函数创建定时器。CWnd::SetTime()函数有两个方式：第一个方式允许指定静态函数在定时器超时的情况下调用；第二个模式向调用　SetTime()的窗口发送WM_TIMER窗口消息，这个消息像其他Windows消息一样被相应处理。

3. 步骤

1) 创建定时器，指定时间后调用一静态函数

通过使用下面的语法定义定时器回调函数，这个函数将作为类的静态函数:

```
private:
static void CALLBACK EXPORT TimerProc( HWND hWnd, UINT nMsg,
   UINT nIDEvent, DWORD dwTime );
```

使用如下代码创建具有函数名的定时器：

```
SetTimer(
   3,                      // event ID, passed to time out process
   1000,                   // time out time in milliseconds
   CWzdView::TimerProc     // timeout process called
   );
```

在这个实例中，1秒以后调用CWzdView::TimerProc()函数。事件标识号“ 3 ”被发送到TimerProc()。

实现定时功能，如下所示：

```
void CALLBACK EXPORT CWzdView::TimerProc( HWND hWnd, UINT nMsg,
```

```
    UINT nIDEvent, DWORD dwTime )
{
    // hWnd will be this window
    // nMsg will be WM_TIMER
    // nIDEvent will be the event id specified in SetTimer
    // dwTime will be the current system time
}
```

2) 创建当时间超时时发送WM_TIMER消息的定时器

启动定时器，当定时终止时向窗口类发送 WM_TIMER窗口消息，可以做如下调用：

```
SetTimer(
    4,                  // event ID
    2000,               // time in milliseconds
    NULL                // causes a WM_TIMER message to be sent to this window
    );
```

在这个实例中，WM_TIMER消息在2秒钟以后被发送到窗口，它使用事件标识号“ 4 ”。

可使用ClassWizard向窗口类中添加WM_TIMER 消息。创建的消息处理函数如下所示：

```
void CWzdView::OnTimer( UINT nIDEvent )
{
    // nIDEvent will be the event id specified in SetTimer

    CView::OnTimer( nIDEvent );
}
```

3) 取消定时器

1) 要在超时之前取消定时器，可使用 CWnd::KillTimer，如下所示：

```
UINT timerID = SetTimer( 3,3000,NULL );
KillTimer(
    timerID      // timer id returned from SetTimer()
    );
```

注意本例中是通过使用 SetTimer()返回的标识号来识别要取消的定时器。

4. 注意

■ 创建一个重复计时的定时器没有简单的方法。每一次都必须在定时器超时后调用CWnd::SetTimer()函数。

■ 定时器将占用系统资源。如果应用程序需要多个定时器，最好是复用一个简单的系统定时器。首先确定希望跟踪的最小时间单元，然后当应用程序的某一部分试图创建定时器时，使它在相关列表中创建一个项目 (entry)。该项目中应该指定它所表示的系统时钟的倍数值，以及当时间超时事件发生时向何处通知该消息 (指定WM_TIMER消息的窗口句柄或者要调用的函数地址)。接下来创建以最小时间单元作为超时时限的定时器。一旦时间超时将复位该定时器，同时为列表中所有项目中的时间计数器递增一个计数单元，直到它们超时为止。当列表中某一个项目时间超时，将向该项目中指定的窗口指针发送 WM_TIMER 消息，或者调用其中指定的函数地址。

5. 使用光盘时注意

运行随书附带光盘上的工程时，在 OnTestWzd()函数中设置断点。单击 Test和Wzd菜单命令，观察两个定时器的创建和超时过程。

15.2 实例61：播放声音

1. 目标

使应用程序播放声音。

2. 策略

使用两个Windows API在程序中播放声音。其中::Beep() API创建能够设置频率和持续时间的单音(single tone)。::sndPlaySound() API允许应用程序播放.wav文件。

3. 步骤

1) 生成单音

为在应用程序中创建单音，可使用以下代码：

```
::Beep(
   1000,          // sound frequency in hertz
   1000           // sound duration in milliseconds
   );
```

第一个参数是音调的频率，赫兹数值越高音调越高 (人耳可以听到的声音的范围是从20赫兹到20000赫兹)。第二个参数是以毫秒计的持续时间。

wav文件是数字化的音频文件。为了播放. wav文件，系统必须有声卡。用户必须在应用程序中手工添加MFC的多媒体支持。

2) 播放.wav文件

为了包含MFC对.wav文件的支持，在每个将调用该API的模块中加入下面的语句：

```
#include <mmsystem.h>
```

同时为应用程序加入winmn.lib到连接设置。

从磁盘文件播放.wav文件，使用代码如下：

```
::sndPlaySound(
   "sound.wav",                  // file to play
   SND_SYNC |                    // or SND_ASYNC to play in another thread
   SND_LOOP |                    // play continually (SND_ASYNC must be set too)
   SND_NODEFAULT                 // play nothing if error in .wav file
   );
```

停止播放.wav文件，可使用以下代码：

```
::sndPlaySound(" ", 0);
```

3) 在应用程序的资源中包含.wav文件

如果希望在应用程序的资源中包含 .wav文件，单击Developer Studio的Insert和Resource菜单命令，打开Insert Resource对话框。单击Import然后找到并选择希望包含的 .wav文件。一个WAVE文件夹将自动在资源中创建，同时 .wav文件被添加和拷贝到工程的\res目录中。

为播放定义于应用程序资源中的 .wav文件，可以使用下面的代码。在这个实例中，使用标识符IDR_SOUND识别.wav文件。

```
// find .wav file name in resources and play
HRSRC hRsrc = FindResource( AfxGetResourceHandle(),
   MAKEINTRESOURCE( IDR_SOUND ), "WAV" );
HGLOBAL hglb = LoadResource( AfxGetResourceHandle(), hRsrc );
::sndPlaySound( ( LPCTSTR )::LockResource( hglb ),
```

```
    SND_MEMORY | SND_SYNC | SND_NODEFAULT );
FreeResource( hglb );
```

4. 注意

需要声卡、麦克风和多媒体软件才能创建自己的 .wav文件。

5. 使用光盘时注意

运行随书附带光盘上的工程时，在 OnTestWzd()函数中设置断点。单击 Test和Wzd菜单命令，注意生成的三种声音。

15.3 实例62：创建VC++宏

1. 目标

充分利用VC++的功能来创建宏。

2. 策略

简单回顾一下MFC支持的宏。 C++试图以单纯的内联代码来代替宏，然而许多情况下必须用宏实现而无法使用内联代码。

3. 步骤

1) 创建简单的宏定义

使用下面的语法来定义宏：

```
#define WZD1 7    // WZD1 is substituted for 7 when used in code
#define WZD2      // WZD2 is simply defined so that #ifdef WZD2 is TRUE
```

2) 创建条件宏指令

使用下面的语法，对代码行进行条件编译：

```
#ifdef WZD1                 // if defined, process next
#undef WZD1                 // undefine WZD1
#endif
#ifndef WZD1                // if WZD1 not defined, process next
#endif

#if WZD2 = 5                // if WZD2 equals 5, process next
   //
#elif WZD2 > 6              // else if WZD6 is greater then 6
   //
#else                       // else...
   //
#endif
```

3) 创建带参数的宏指令

带参数的简单宏的语法如下：

```
#define WZD3( arg1,arg2 ) \
   arg1 + arg2;          // WZD3( 2,5 ) is substituted for 2+5 in the code
```

当使用续行符(“\”)时，本行内续行符的后面必须没有任何字符，即使是空格也不行。

为使宏的参数带双引号出现，可在宏参数之前田间加符号“ #”:

```
#define WZD4( arg1,arg2 ) \
   arg1 + #arg2;    // WZD4( 1,test ) becomes 1 + "test" in the code
```

为使宏的参数带单引号出现，可在宏参数之前加符号“ #@ ”：

```
#define WZD5( arg1,arg2 ) \
   arg1 + #@arg2;                    // WZD5( 1,t ) becomes 1 + 't'
```

为了在宏的标记(token)中添加宏参数，可在标记之前加符号“ ## ”：

```
#define WZD6( arg1,arg2 ) \
   arg1 + arg2##3;                   // WZD6( 1,2 ) becomes 1 + 23
```

4) 使用预定义宏

VC++编译器预定义四个宏，允许在代码中插入有关源文件的信息。当处理到包含源文件名的文本字符串时， _FILE_宏指令展开(FILE前后各有一个下划线)。例如，在WzdView .cpp文件的下列代码：

```
Print( __FILE__ );
```

当编译时，展开为：

```
Print( "WzdView.cpp" );
```

_LINE_展开为源文件中的当前行。 _TIME_和_DATE_宏展开为源文件编译时的日期和时间。

4. 注意

_FILE_和_LINE_宏用于更有效地确定出错信息。例如，与其仅仅在应用程序里显示下列出错信息：

```
"File not found"
```

倒不如在代码中包含发生错误的确切位置。如下面一行代码所示：

```
"File not found(_FILE_@_LINE_)"
```

在编译时，将扩展成：

```
"File not found(WzdView.cpp@1232)"
```

5. 使用光盘时注意

随书附带的光盘上没有相应的实例。

15.4 实例63：使用函数地址

1. 目标

向其他应用程序传递成员函数的地址，用于执行或保存。

2. 策略

ANSI C允许像对待其他变量一样对待函数的地址。但是在C语言中的C++对象中的函数是非静态的，这意味着不但必须保持函数的地址，而且必须保存函数所在对象的地址。

在这个实例中，将向其他应用程序传递 CWzdView类的成员函数的地址。其他的函数将间接地调用这个函数。

```
int CWzdView::ViewFunc( int i,LPCSTR s,BOOL b )
{
   i;s;b;      // process values
   return 0;   // result is returned
}
```

如上所示，ViewFunc()函数包含三个参数并返回整数值。

3. 步骤

为成员函数定义函数指针类型，如下所示：

```
typedef int ( CWzdView::*PMFUNC)(int, LPCSTR, BOOL );
```

在这个实例中，PMFUNC成为函数指针类型。将该定义放在容易加入到代码的地方，例如每个代码模块所包含的文件中。

当向这个函数传递指针的时候，还需要包含它指向其所在类的实例的指针：

```
void Perform( CWzdView *pClass, PMFUNC pFunc );
```

在这个实例中，PMFUNC驻留在CWzdView类中。因此可以使用下面的参数调用Perform()。

```
wzdUtil.Perform(
   xxx,        // a pointer to the object that ViewFunc() sits in
   pFunc       // the pointer to ViewFunc()
   );
```

为间接调用这个成员函数，需要如下结合对象和成员函数指针：

```
// pClass is the object pointer;
pFunc is the function pointer i =
   ( pClass -> *func )( 1,"test",TRUE );
```

4. 注意

■ 可以在C++类中封装指针对(例如：Object::Function)，如下所示：

```
class CFunc
{
   CFunc( CWzdView *pClass, PMFUNC func )
      {m_pClass = pClass;m_pFunc = pFunc;};
   classPtr *m_pClass;
   funcPtr *m_pFunc;
   Perform( int i,LPCSTR s,BOOL b ){( PClass -> *func )( i,s,b );};
}
```

这种方法的好处是成员函数可以作为简单的参数被传递和存储，缺点是需要一个用于各种函数类型的封装类。还需要考虑删去多余的类实例。有时候隐藏间接调用的内部细节可能比其所具有的简单性更易使人混淆。

■ 本实例与其说是MFC的练习倒不如说是与C++的联系。既然它属于C++中比较晦涩难懂的一部分内容，这里解释一下还是有必要的。

5. 使用光盘时注意

当运行随书附带光盘的工程时，在OnTestWzd()函数中设置断点。单击Option和Wzd菜单命令，观察一个类实例的函数被间接调用。

15.5 实例64：二进制字符串

1. 目标

使用MFC的字符串类操作二进制字符串。

2. 策略

MFC的CString类允许操作以零字符终止的文本字符串。为了使它也能对可能带有许多零字符的二进制字符串进行操作，本例将把二进制字符串转化且保存为等值的十六进制文本字符串。换句话说，字节 0x23、0x43、0xa4在CString中将被保存为文本字符串“ 2343a4 ”。

从CString类派生出自己的类，在其中添加两个新的成员函数，它们将转换并保存二进制字符串为文本字符串。

3. 步骤

如本实例结尾的程序清单 —— 字符串类所示，手工创建 CString的派生类。ClassWizard不能创建从CString派生的类。

使用文本编辑器在该类中添加 PutBinary()函数。由于CString只能够在内部保存为字符串文本，因此需要将二进制字符串转化为文本表示。为此使用 CString的Format()函数，将二进制字节转化为一个双字符的十六进制数值:

```
void CWzdString::PutBinary( LPBYTE pByte,int len )
{
   Empty();
   CString hex;
   for ( int i = 0;i < len;i++ )
   {
      hex.Format( "%02X",pByte[i] );
      *this += hex;
   }
}
```

该类中的另外一个成员函数是 GetBinary()，用于从CString类重新生成二进制字符串。使用sscanf()可将十六进制字符串重新转换二进制字符串，如下所示:

```
// returned value is actual binary length
int CWzdString::GetBinary( LPBYTE pByte,int maxlen )
{
   // make sure the string contains only valid hex characters
   if ( SpanIncluding( "0123456789aAbBcCdDeEfF" ) != *this )
   {
      return 0;
   }

   // make sure less then max bytes
   int len = GetLength();
   if ( len > maxlen*2 )
   {
      len = maxlen*2;
   }

   // pad HEX to even number
   CString hex = *this;
   if ( ( len % 2 ) != 0 )
   {
      len++;
      hex += "0";
   }
```

```
    // convert to binary
    len/ = 2;
    for ( int i = 0; i < len; i++ )
    {
        int b;
        sscanf( hex.Mid( ( i * 2 ), 2 ), "%02X", &b );
        pByte[i] = ( BYTE )b;
    }
    return( len );

}
```

4. 注意

使用sscanf()函数时应小心。在上面的实例中，即使所希望的是字节，也需要在整型变量中扫描搜索。这是因为无论如何 sscanf()都返回整数值。如果希望在字节中搜索，当 sscanf()销毁堆栈时应用程序可能将不时地发生莫名其妙地崩溃。

5. 使用光盘时注意

当运行随书附带光盘的工程时，在 WzdView.cpp里的OnWzdType()函数中设置断点。单击Option和Wzd菜单命令，观察二进制字符串被操作。

6. 程序清单 —— 字符串类

```
// CWzdString.h
//
//////////////////////////////////////////////////////////////////////////

#if !defined WZDSTRING_H
#define WZDSTRING_H

class CWzdString : public CString
{
public:
    CWzdString::CWzdString() : CString(){;};
    CWzdString::CWzdString( const CString& str ) : CString( str ){;};
    CWzdString::CWzdString( LPCTSTR lpsz ) : CString( lpsz ){;};
    CWzdString::CWzdString( LPCWSTR lpsz ) : CString( lpsz ){;};

// Attributes
public:

// Operations
public:
    void PutBinary( LPBYTE pByte,int len );
    int GetBinary( LPBYTE pByte,int maxlen );

};

//////////////////////////////////////////////////////////////////////////

#endif
```

```
// WzdString.cpp
//

#include "stdafx.h"
#include "WzdString.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////////////////////////////////////
// CWzdString

// extracts characters from beg to end
CString CWzdString::Extract( int beg,int end)
{
   return Mid( beg - 1,end - beg + 1 );
}

// stores binary as HEX ascii string
void CWzdString::PutBinary( LPBYTE pByte,int len )
{
   Empty();
   CString hex;
   for ( int i = 0;i < len;i++ )
   {
      hex.Format( "%02X",pByte[i] );
      *this+ = hex;
   }
}

// retrieves HEX ascii string as binary
// returned value is actual binary length
int CWzdString::GetBinary( LPBYTE pByte,int maxlen )
{
   // make sure the string contains only valid hex characters
   if ( SpanIncluding( "0123456789aAbBcCdDeEfF" ) != *this )
   {
      return 0;
   }

   // make sure less then max bytes
   int len = GetLength();
   if ( len > maxlen*2 )
   {
      len = maxlen*2;
   }

   // pad HEX to even number
```

```
    CString hex = *this;
    if ( ( len % 2 ) != 0 )
    {
        len++;
        hex += "0";
    }

    // convert to binary
    len/ = 2;
    for ( int i = 0; i < len; i++ )
    {
        int b;
        sscanf( hex.Mid( ( i * 2 ), 2 ), "%02X", &b );
        pByte[i] = ( BYTE )b;
    }
    return( len );

}
```

15.6 实例65：重新启动计算机

1. 目标

编写程序重新启动计算机系统，无论是本地系统或者是其他远程系统:

2. 策略

为关闭自己的系统，将使用 ::ExitWindowsEx() API。关闭其他的系统则使用 ::Initiate System ShutDown () API。事实上，该API也会关闭自己的系统，甚至允许设置关闭的延迟时间。在关闭系统时要考虑的另一种情况是运行 NT的计算机需要 SE_SHUTDOWN_PRIVILEDGE。本例将使用::AdjustTokenPriviledge()来获取此特权。

3. 步骤

1) 改变NT的特权

为重启动运行NT的计算机，需要给予应用程序关闭系统的特权:

```
static HANDLE hToken;
static TOKEN_PRIVILEGES tp;
static LUID luid;
if ( ::OpenProcessToken( GetCurrentProcess(),
    TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken ) )
{
    ::LookupPrivilegeValue( NULL, SE_SHUTDOWN_NAME, &luid );
    tp.PrivilegeCount          = 1;
    tp.Privileges[0].Luid      = luid;
    tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
    ::AdjustTokenPrivileges( hToken, FALSE, &tp,
        sizeof( TOKEN_PRIVILEGES ), NULL, NULL );
}
```

注意 用户的密码帐户必须能够赋予用户关闭系统的特权。

注意 建议添加以上代码，即使应用程序运行在其他的系统之上，例如 Windows 98。

(不会损害系统)。

2) 重启动本地计算机

为重启动本地计算机，使用以下代码：

```
::ExitWindowsEx(
   // EWX_LOGOFF            // logs user off
   // EWX_SHUTDOWN          // shuts down system
   EWX_REBOOT               // reboots system
   // EWX_POWEROFF          // shuts down and turns system off
                            //    (if system has power-off feature)
   // |EWX_FORCE,           // forces processes to terminate  --
                            //    use only in emergency
   0                        // reserved
   );
```

3) 重启动本地或者远程计算机

重启动本地或者远程计算机，使用以下代码：

```
::InitiateSystemShutdown(
   "\\\\OtherMachine",      // computer to shut down - -
                            //    NULL = local system
   "Goodbye!",              // a message to user
   60,                      // time to display message (in seconds)
   FALSE,                   // TRUE = force processes to terminate
   TRUE                     // TRUE = reboot after shutdown
   );
```

取消由::InitiateSystemShutdown()引发的重启动行为，使用以下代码：

```
::AbortSystemShutdown(
   "\\\\OtherMachine"       // computer NOT to shut down - -
                            //    NULL = local system
   );
```

4. 注意

远程API允许当远程服务器挂起或者安装新软件时，重新启动远程计算机。

5. 使用光盘时注意

当运行随书附带光盘上的工程时，单击 Test菜单项。第一个菜单命令立即启动计算机。第二个命令在一秒钟以后重启动计算机，最后的命令则取消由第二条命令引发的重启动。

15.7 实例66：获得可用磁盘空间

1. 目标

检查磁盘驱动器上剩余的空间。

2. 策略

使用::GetDiskFreeSpaceEx() API。

3. 步骤

获取可用磁盘空间

使用下面的调用来获取可用磁盘空间:

```
ULARGE_INTEGER freeToCaller;
```

```
ULARGE_INTEGER diskCapacity;
ULARGE_INTEGER freeSpace;

::GetDiskFreeSpaceEx(
   "c:\\",          // pointer to the directory name
   &freeToCaller,   // bytes available to caller
   &diskCapacity,   // total bytes on disk
   &freeSpace       // free bytes on disk
   );
```

4. 注意

对于Windows 95系统，需要OEM2或者更高版本以使用该函数。如果应用程序不能使用该函数，仍然可以使用::GetDiskFreeSpace()，但是必须根据返回的值计算可用空间。

5. 使用光盘时注意

当运行随书附带光盘上的工程时，单击 Test和Wzd菜单项，打开对话框。按下按钮获得 C 盘的当前可用空间。

15.8 实例67：闪烁窗口和文本

1. 目标

在对话框中闪烁文本信息或者闪烁对话框的标题以吸引用户的注意力。

2. 策略

有两种方法可实现闪烁，并且都涉及到使用定时器来间歇地修改窗口。以文本为例，只需要隐藏和显示它所在的窗口。以弹出窗口和重叠窗口为例，使窗口有效或者无效便可闪烁标题栏。这种方式通常在出错或者提示消息时使用，以此吸引用户的注意。

3. 步骤

1) 创建闪烁文本

启动定时器(参考实例60)。如果是在对话框类中，则在 OnInitDialog消息处理函数中启动定时器：

```
SetTimer(
   1,              // event ID
   250,            // time in milliseconds
   NULL            // causes a WM_TIMER message to be sent to this window
   );
```

添加WM_TIMER消息处理函数，交替隐藏和显示文本：

```
void CWzdDialog::OnTimer( UINT nIDEvent )
{
    if ( nIDEvent == 1 )
    {
        static BOOL t = TRUE;
        m_ctrlStatic.ShowWindow( ( t = !t )?SW_SHOW:SW_HIDE );
    }

    CDialog::OnTimer( nIDEvent );
}
```

注意通过使用nIDEvent 标识号，仍然可以在对话框类中将定时器消息处理函数用于其他

目的。

关闭定时器时，确保文本可见。

2) 创建闪烁窗口

如前面所示启动定时器。

添加WM_TIMER消息处理函数，使用 FlashWindows()使窗口标题栏在活动和非活跃两种状态之间切换:

```
void CWzdDialog::OnTimer( UINT nIDEvent )
{
   FlashWindow(
      TRUE    // if FALSE, forces caption bar to be active- -
            //    useful when flashing stops and
            //    you want bar to be active
      );

   CDialog::OnTimer( nIDEvent );
}
```

当停止闪烁时，确保调用 FlashWindow(FALSE)，以保证窗口标题条处于活动状态的 (标题加亮显示)。

4. 注意

可以使用这种方法闪烁主窗口，如果应用程序在后台出了问题而用户正忙于其他的事情，甚至如果主窗口在桌面上不可见，任务栏中表示该应用程序的项目也会闪烁。

5. 使用光盘时注意

当运行随书附带光盘上的工程时，单击 Test和Wzd菜单项打开对话框。按下文本闪烁按钮使文本闪烁或者按下窗口闪烁按钮使窗口标题栏闪烁。