第 1 章　ObjectWindows 基础

ObjectWindows 库为开发 Windows 应用程序提供了一个功能强大的工具。若没有这样一个库，编写 Windows 应用程序将变成一项复杂、艰苦且强度高的工作。ObjectWindows 库成功地把面向对象和事件驱动编程结合在一起，它证明了这两种程序设计方法怎样完美地一起工作。这一章将介绍 ObjectWindows 类库和一些与 Windows 有关的基本信息，包括如下内容：

- ObjectWindows 层次
- 对 Windows 消息的响应
- 发送消息
- 用户自定义消息
- 简单的基于 OWL Windows 程序的实例

1.1　ObjectWindows 层次

ObjectWindows 层次能显著地减少创建 Windows 应用程序界面所需的代码总量。这一节展示了各个 ObjectWindows 成员所选择的类声明，并简要地讨论了每一层次的类所支持的功能。ObjectWindows 支持如下类的层次：

- 模块和应用程序类
- Windows 类
- 控件类
- 对话框类
- 文档及浏览类
- 打印类
- 图形类

ObjectWindows 库的层次相当精致而且包含有多重继承性。这个分层有两个基根类，称为 TEventHandler 和 TStreamableBase。这些类支持各种 Windows 类的事件处理功能和流功能(streamability)。

1.1.1　模块和应用程序类

这一层包括 TModule 类和它的子层 TApplication 类。TModules 类是 TEventHandler 类和 TStreamableBase 类的子类。这个很普通的 OWL 子层有非常重要的任务，就是初始化 Windows 并管理 Windows 消息流，使它们到达各自正确的接受者。下面是 TApplication 类的声明：

```
class _OWLCLASS TApplication : public TModule {
  public:
    class _OWLCLASS TXInvalidMainWindow : public TXOwl {
      public:
      TXInvalidMainWindow();
      TXOwl * Clone();
      void Throw();
    };
    HINSTANCE     hPrevInstance;
    int           nCmdShow;

    TDocManager *   DocManager;
    TFrameWindow *  MainWindow;
    HACCEL          HAccTable;

    TApplication(const char far * name = 0);
    TApplication(const char far * name,
                 HINSTANCE   hInstance,
                 HINSTANCE   hPrevInstance,
                 const char far * cmdLine,
                 int         cmdShow);

    ~TApplication();

    static void     SetWinMainParams(HINSTANCE     hInstance,
                                     HINSTANCE     hPrevInstance,
                                     const char far * cmdLine,
                                     int           cmdShow);

    void            GetWinMainParams();

    virtual BOOL  CanClose();
    virtual int   Run();

#if defined(__WIN32__)
    TMutex& GetMutex();

    class _OWLCLASS TAppLock : public TMutex::Lock {
      public:
        TAppLock(TApplication &app);
    };

    //
    // override TEventHandler::Dispatch() to handle multi-thread
    // synchronization
    //
    virtual   LRESULT     Dispatch(TEventInfo& info, WPARAM wp, LPARAM lp = 0);
#endif

    BOOL                  PumpWaitingMessages();    // pumps all waiting msgs
    virtual int           MessageLoop();            // Loops until break or WM_QUIT
    virtual BOOL          ProcessAppMsg(MSG& msg);
    void                  SuspendThrow(xalloc& x);  // saves xalloc exception info
    void                  SuspendThrow(xmsg& x);    // saves xmsg exception info
    void                  SuspendThrow(TXOwl& x);// saves copy of TXOwl exception
    void                  SuspendThrow(int);        // set bit flag to log exception
    void                  ResumeThrow();  // checks and rethrows suspended exceptions
    int                   QueryThrow() {return XState;}  // return suspend flags
```

```
    enum{
    xsUnknown      = 1,
    xsBadCast      = 2,
    xsBadTypeid    =4,
    xsMsg          = 8,
    xsAlloc        = 16,
    xsOwl          = 32,
  };
  //
  // begin and end of a modal window's modal message loop
  //
  int              BeginModal(TWindow* window, int flags=MB_APPLMODAL);
  void             EndModal(int result);
  virtual void     PreProcessMenu(HMENU);   // called from MainWindow
  //
  // Dead TWindow garbage collection
  //
  void             Condemn(TWindow* win);

  //
  // Call this function after each msg dispatch if TApplication's message
  // loop is not used.
  //
  void             PostDispatchAction();

  //
  // TApplication has no event table itself, defers event handling to
  // DocManager if it has been installed.
  //
  BOOL Find(TEventInfo&, TEqualOperator = 0);
  // Obsolete
  static HINSTANCE GetLibInstance(TModule* module) {return *module;}
  void             EnableBWCC(BOOL enable = TRUE,UINT language = 0);
  BOOL             BWCCEnabled() const {return BWCCOn;}
  TModule*         GetBWCCModule() const {return BWCCModule;}
  void             EnableCtl3d(BOOL enable = TRUE);
  void             EnableCtl3dAutosubclass(BOOL enable);
  BOOL             Ctl3dEnabled() const {return Ctl3dOn;}
  TModule*         GetCtl3dModule() const { return Ctl3dModule;}
protected:
  BOOL             BreakMessageLoop;
  int              MessageLoopResult;
  virtual void     InitApplication();   // "first"-instance initialization
  virtual void     InitInstance();      // each-instance initialization
  virtual void     InitMainWindow();    // init application main window
  virtual int      TermInstance(int status); // each-instance termination
  //
  // (re)set a new main window sometime later, after construction
  //
  TFrameWindow*    SetMainWindow(TFrameWindow * window);
```

```
    //
    // called each time there are no messages in the queue. idle count is
    // incremented each time, & zeroed when messages are pumped. Return
    // whether or not more processing needs to be done.
    //
    // default behavior is to give the main window an opportunity to do idle
    // processing by invoking its IdleAction() member function when
    // "idleCount"is 0
    //
    virtual BOOL        IdleAction (LongidleCount);
  private:
    BOOL                BWOCOn;
    TModule*            BWCCModule;

    BOOL                Ctl3dOn;
    TModule*            Ctl3dModule;

#if defined(_WIN32_)
    TMutex              Mutex;
#endif

    static HINSTANCE    InitHInstance;
    static HINSTANCE    InitHPrevInstance;
    static const char far* InitCmdLine;
    static int          InitCmdShow;

    //
    // exception handling state
    //
    int             XState;
    string          XString;
    size_t          XSize;
    TXOwl* XOwl;

    //
    // Condemned TWindow garbage collection
    //
    void            DeleteCondemned();
    TWindow*        CondemnedWindows;
    //
    // hidden to prevent accidental copying or assignment
    //
    TApplication(const TAppliacation&);
    TApplication& operator =(const TApplication&);
};
```

先前的声明说明了 TApplication 并不是一个粗框架类。相反，它包含有功能强大的操作：用来运行应用程序类的重要 Run 成员函数。一般来说，应从 TApplication 类中派生出实际的应用程序类，TApplication 类支持以下操作：

- 初始化 Windows 应用程序及其实例
- 处理消息
- 管理模态(modal)窗口的消息循环
- 收集废弃的内存

1.1.2　Windows 类

Windows 类层次使用 TWindow 类作为它的根。TWindow 是 TEventHandler 和 TStreamableBase 的子类。下面是 Windows 类子层次的一个轮廓：

```
+ TWindow
    +TFrameWindow
        – TFloatingFrame (also a descendant of TTinyCaption)
        * TMDIFrame
        * TDecoratedFrame
            –TDecoratedMDIFrame
        –TMDIChild
```

* 号指示这个类作为其下所列子类的合作父类。

1.1.3　TWindow 类

TWindow 类是所有 Windows(包括控件，对话框，MDI 窗口等)的基类，它封装了所有这些类所共享的基本操作和公用操作。下面是 TWindow 类的声明：

```
class _OWLCLASS TWindow: virtual public TEventHandler,
                         virtual public TStreamableBase {
  public:
    class _OWLCLASS TXWindow : public TXOwl {
      public:
        TXWindow(TWindow* win = 0, UINT resourceId = IDS_INVALIDWINDOW);
        TXWindow(const TXWindow& arc);
        int Unhandled(TModule * app,unsigned promptResId);
        TXOwl * Clone();
        void Throw() {throw * this;}
        TWindow *      Window;
        static string Msg(TWindow * wnd,UINT resourceid);
};

TStatus            Status;
HWND               HWindow;  // handle to associated MS-Windows window
char far *         Title;
TWindow *          Parent;
TWindowAttr        Attr;
WNDPROC            DefaultProc;
TScrollerBase * Scroller;

TWindow(TWindow *      parent,
        const char far * title = 0,
        TModule *      module = 0);

TWindow(HWND hWnd,TModule * module = 0);

virtual ~TWindow();

//
// two iterators that take function pointers
//
TWindow *          FirstThat(TCondFunc test, void * paramList =0);
void               ForEach(TActionFunc action, void * paramList =0);
```

```
//
// two iterators that take pointers to member functions
//
TWindow *       FirstThat(TCondMemFunc test,void * paramList = 0);
void            ForEach(TActionMemFunc action,void * paramList =0);

//
// other functions for iteration
//
TWindow *       Next() {return SiblingList;)
void            SetNext(TWindow * next) {SiblingList = next;}
TWindow *       GetFirstChild()
                    {return ChildList ? ChildList ->SiblingList : 0;}
TWindow *       GetLastChild() {return ChildList;}
TWindow *       Previous();
unsigned        NumChIldren();  // number of child windows

//
// query and set the flags
//
void            SetFlag(TWindowFlag mask) {Flags |= DWORD(mask);}
void            ClearFlag(TWindowFlag mask) {Flags &= DWORD(-mask);}
BOOL            IsFlagSet(TWindowFlag mask) {return (Flags & mask) ? 1 : 0;}

//
// sets/clears flag which indicates that the TWindow should be
// created if a create is sent while in the parent's child list
//
void            EnableAutoCreate() {SetFlag(WB_AUTOCREATE);}
void            DisableAutoCreate() {ClearFlag(WB_AUTOCREATE);}

//
// sets flag which indicates that the TWindow can/will transfer data
// via the transfer mechanism
//
void            EnableTransfer() {SetFlag(WB_TRANSFER);}
void            DisableTransfer() {ClearFlag(WB_TRANSFER);}

//
// Window's default module access functions
//
TModule *       GetModule() const {return Module;}
void            SetModule(TModule * module) {Module = module;}

// Obsolete library id functions
//
HINSTANCE       GetLibInstance(const TModule * modul=0)
                  const{return module? * module; * GetModule();}

TApplication *  GetApplication() const{return Application;}
WNDPROC         GetThunk() const {return Thunk;}
virtual BOOL    Register();

//
// create/destory an MS_Windows element to be associated with an OWL window
//
virtual BOOL    Create();
virtual void    PerformCreate(int menu OrId);
```

```
    BOOL            CreateChildren();
    virtual void    Destory(int retVal = 0);

    //
    // suggest an Owl window to close itself
    virtual void    CloseWindow(int retVal = 0);

    //
    // This function is obsolete. Destroy() should be called directly,& then
    // the window destructed (using delete,etc).
    //
    void            ShutDownWindow(int retVal =0);

    #if defined(__WIN32__)
    //
    // override TEventHandler::Dispatch() to handle multi-thread
    // synchronization
    //
    virtual LRESULT Dispatch(TEventInfo& info, WPARAM wp, LPARAM lp = 0);
#endif

    //
    // called from TApplication::ProcessAppMsg() to give the window an
    // opportunity to perform preprocessing of the Windows message
    //
    // if you return TRUE,further processing of the message is halted
    //
    // if you override this method in a derived class,make sure to call this
    // routine because it handles translation of accelerators...
    //
    virtual BOOL    PreProcessMsg(MSG& msg);
    virtual BOOL    IdleAction(long idleCount);
    virtual BOOL    HoldFocusHWnd(HWND hWndLose, HWND hWndGain);

    int             GetId() const(return Attr.Id;)
    TWindow *       ChildWithId(int id) const;

    virtual void    SetParent(TWindow* newParent);
    virtual BOOL    SetDocTitle(LPCSTR docname,int index);

    void            Show(int showCmd);
    void            SetCaption(const char far* title);
    void            GetWindowTextTitle();
    void            GetHWndState();
    BOOL            SetCursor(TModule * module, TResId resId);

    void            SetBkgndColor(DWORD color) {BkgndColor = color;}

    virtual BOOL    CanClose();

    //
    // forwards the current event to "hWnd" using either PostMessage() or
    // SendMessage(). Owl window version calls directly to window proc on send.
    //
    LRESULT         ForwardMessage(HWND hWnd, BOOL send = TRUE);
    LRESULT         ForwardMessage(BOOL send = TRUE);

    //
    // send message to all children
    //
    void        ChildBroadcastMessage(UINT msg,WPARAM wParam=0,LPARAM lParam=0);
```

```
//
// Called from StdWndProc to allow exceptions to be caught and suspended.
// Calls HandleMessage from within try block. Catches and suspends all
// exceptions before returning to Windows (Windows is not exception safe).
//
LRESULT         ReceiveMessage(UINT   msg,
                               WPARAM wParam = 0,
                               LPARAM lParam = 0);

//
// Call a Window's window proc to handle a message. Similar to SendMessage
// but more direct.
//
LRESULT         HandleMessage(UINT   msg,
                              WPARAM wParam = 0,
                              LPARAM lParam = 0);

//
// virtual functions called to handle a message,and to deal with an
// unhandled message in a default way.
//
virtual LRESULT  WindowProc(UINT msg,WPARAM wParam,LPARAM lParam);
virtual LRESULT  DefWindowProc(UINT msg,WPARAM wParam,LPARAM lParam);

//
// called by WindowProc() to handle WM_COMMANDs
//
// "id"                 -specifies the identifier of the menu item or control
//
// "hWndCtl"            -specifies the control sending the message if the message
//                      is from a control;otherwise it is 0
//
// "notifyCode"   -specifies the notification message if the message is from
//                      a control. if the message is from an accelerator,it is 1.
//                      if the message is from a menu,it is 0
//
virtual LRESULT  EvCommand(UINT id, HWND hWndCtl, UINT notifyCode);

//
// called by WindowProc() to handle WM_COMMAND_ENABLE
//
virtual void         EvCommandEnable(TCommandEnabler& ce);
//
// default processing,deals with special cases or calling defWindowProc
//
LRESULT         DefaultProcessing();

//
// Paint function called by base classes when responding to WM_PAINT
//
virtual void         Paint(TDC& dc,BOOL erase, TRect& rect);

//
// tranfer buffer
//
void                 SetTransferBuffer(void * transferBuffer)
                         {TransferBuffer = transferBuffer;}
virtual UINT         Transfer(void * buffer = TTransferDirection direction);
```

```
virtual void        TransferData(TTransferDirection direction);

//
// installs the thunk as the window function and saves the previous window
// function in "DefaultProc"
//
void                SubclassWindowFunction();

//
// Encapsulated HWND functions inline
//
//
// allow a TWindow& to be used as an HWND in Windows API calls
//
operator            HWND() const {return HWindow;}
BOOL                IsWindow() const {return :: IsWindow(HWindow);}

//
// messages
//
LRESULT             SendMessage(UINT   msg,
                                WPARAM wParam =0,
                                LPARAM lParam =0);
LRESULT             SendDlgItemMessage(int     childId,)
                                       UINT   msg,
                                       WPARAM wParam = 0,
                                       LPARAM lParam =0);
BOOL                PostMessage(UINT   msg,
                                WPARAM wParam = 0,
                                LPARAM lParam = 0);
static HWND         GetCapture();
HWND                SetCapture();
static void         ReleaseCapture();
static HWND         GetFocus();
HWND                SetFocus();
BOOL                IsWindowEnabled() const;
BOOL                EnableWindow(BOOL enable);
void                SetRedraw(BOOL redraw);

//
// window coordinates, dimensions...
//
void                ScreenToClient(TPoint& point) const;
void                MapWindowPoints(HWND   hWndTo,
                                    TPoint *  points,
                                    int       count) const;
void                GetClientRect(TRect& rect) const;
TRect               GetClientRect() const;
static HWND         WindowFromPoint(const TPoint& point);
HWND                ChildWindowFromPoint(const TPoint& point) const;
void                ClientToScreen(TPoint& point) const;
void                GetWindowRect(TRect& rect) const;
TRect               GetWindowRect() const;
static void         AdjustWindowRect(TRect& rect,DWORD style,BOOL menu);
static void         AdjustWindowRectEx(TRect& rect,DWORD style,
                                       BOOL menu, DWORD exStyle);
```

```
    //
    // window and class Words and longs, window properties
    //
    long            GetClassName(char far * className, int maxCount) const;
    long            GetClassLong(int index) const;
    long            SetClassLong(int index, long newLong);
    WORD            GetClassWord(int index) const;
    WORD            SetClassWord(int index,WORD newWord);
    long            GetWindowLong(int index) const;
    long            SetWindowLong(int index,long newLong);
    WORD            GetWindowWord(int index) const;
    WORD            SetWindowWord(int index,WORD newWord);
    int             EnumProps(PROPENUMPROC proc);
    HANDLE          GetProp(WORD atom,HANDLE data) const;
    HANDLE          RemoveProp(WORD atom const;
    BOOL            SetProp(WORD atom,HANDLE data) const;
    HANDLE          GetProp(const char far * str) const;
    HANDLE          RemoveProp(const char far * str) const;
    BOOL            SetProp(const char far * str, HANDLE data) const;
    //
    // window placement (X,Y) and display
    //
    BOOL            MoveWindow(int x,int y, int w,int h,BOOL repaint = FALSE);
    BOOL            MoveWindow(const TRect& rect,BOOL repaint = FALSE);
    BOOL            ShowWindow(int cmdShow);
    void            ShowOwnedPopups(BOOL show);
    BOOL            IsWindowVisible() const;
    BOOL            IsZoomed() const;
    BOOL            IsIconic() const;
    int             GetWindowTextLength() const;
    int             GetWindowText(char far * str,int maxCount) const;
    void            SetWindowText(const char far * str);
    BOOL            GetWindowPlacement(WINDOWPLACEMENT * place) const;
    BOOL            SetWindowPlacement(const WINDOWPLACEMENT * place);
    //
    // window positioning(Z),sibling relationships
    //
    void            BringWindowToTop();
    static HWND     GetActiveWindow();
    HWND            SetActiveWindow();
    static HWND     GetDesktopWindow();
#if !defined(__WIN32__)
    static HWND     GetSysModalWindow();
    HWND            SetSysModalWindow();
#endif
    HWND            GetLastActivePopup() const;
    HWND            GetNextWindow(UINT dirFlag) const;
    HWND            GetTopWindow() const;
    HWND            GetWindow(UINT cmd) const;
    BOOL            SetWindowPos(HWND       hWndInsertAfter,
                                 const TRect& rect,
                                 UINT       flags);
    BOOL            SetWindowPos(HWND hWndInsertAfter,
```

```
                        int x,int y, int w, int h,
                        UINT flags);

//
// window painting: invalidating, validating & updating
//
void            Invalidate(BOOL erase = TRUE);
void            InvalidateRect(const TRect& rect,BOOL erase = TRUE);
void            InvalidateRgn(HRGN hRgn,BOOL erase = TRUE);
void            Validate();
void            ValidateRect(const TRect& rect);
void            ValidateRgn(HRGN hRgn);
void            UpdateWindow();
BOOL            FlashWindow(BOOL invert);
BOOL            GetUpdateRect(TRect& rect,BOOL erase = TRUE) const;
BOOL            GetUpdateRgn(TRegion& rgn,BOOL erase = TRUE) const;
BOOL            LockWindowUpdate();
BOOL            RedrawWindow(TRect * update,
                             HRGN hUpdateRgn,
                             UINT redrawFlags = RDW_INVALIDATE|
RDW_UPDATENOW | RDW_ERASE);
//
// scrolling and scrollbars
//
int             GetScrollPos(int bar) const;
int             SetScrollPos(int bar,int pos,BOOL redraw = TRUE);
void            GetScrollRange(int bar,int& minPos,int& maxPos) const;
void            SetScrollRange(int bar,
                               int minPos,
                               int maxPos,
                               BOOL redraw = TRUE);
void            ShowScrollBar(int bar, BOOL show = TRUE);
void            ScrollWindow(int          dx,
                             int          dy,
                             const TRect far * scroll=0,
                             const TRect far * clip =0);
void            ScrollWindowEx(int          dx,
                               int          dy,
                               const TRect far * scroll =0,
                               const TRect far * clip = 0,
                               HRGN         hUpdateRgn = 0,
                               TRect far *   update = 0,
                               UINT         flags = 0);

//
// parent/child with Ids
//
int             GetDlgCtrlID() const;
HWND            GetDlgItem(int chlidId) const;
UINT            GetDlgItemInt(int  childId),
                              BOOL * translated =0,
                              BOOL  isSigned = TRUE) const;
void            SetDlgItemInt(int childId),
                              UINT value,
                              BOOL isSigned = TRUE) const;
```

```
int             GetDlgItemText(int      childId,
                               char far * text,
                               int        max) const;
void            SetDlgItemText(int childId,const char far * text) const;
UINT            IsDlgButtonChecked(int buttonId) const;
HWND            GetParent() const;
BOLL            IsChild(HWND) const;
HWND            GetNextDlgGroupItem(HWND hWndCtrl,
                                    BOOL previous = FALSE) const;
HWND            GetNextDlgTabItem(HWND hWndCtrl,
                                  BOOL previous = FALSE) const;
void            CheckDlgButton(int buttonId,UINT check);
void            CheckRadioButton(int firstButtonId,
                                 int lastButtonId,
                                 int checkButtonId);

//
// menus and menubar
//
HMENU           GetMenu() const;
HMENU           GetSystemMenu(BOOL revert = FALSE) const;
BOOL            SetMenu(HMENU hMenu);
BOOL            HiliteMenuItem(HMENU hMenu,UINT,idItem,UINT hilite);
void            DrawMenuBar();

//
// clipboard
//
TClipboard&     OpenClipboard();

//
// timer
//
BOOL            KillTimer(UINT timerId);
UINT            SetTimer(UINT timerId,UINT timeout,TIMERPROC proc = 0);

//
// caret,cursor, font
//
void            CreateCaret(HBITMAP hBitmap);
void            CreateCaret(BOOL isGray,int width,int height);
static UINT     GetCaretBlinkTime();
static void     GetCaretPos(TPoint& point);
void            HideCaret();
static void     SetCaretBlinkTime(WORD milliSecs);
static void     SetCaretPos(int x,int y);
static void     SetCaretPos(const TPoint& pos);
void            ShowCaret();
static void     DestroyCaret();
static void     GetCursorPos(TPoint& pos)
void            SetWindowFont(HFONT font,BOOL redraw);
HFONT           GetWindowFont();

//
// hot keys
//
#if defined(__WIN32__)
```

```
    BOOL            RegisterHotKey(int idHotKey,
                                   UINT modifiers,
                                   UINT virtKey);
    BOOL            UnregisterHotKey(int idHotKey);
#endif
    //
    // Misc
    //
    BOOL            WinHelp(const char far * helpFile,
                            UINT          command,
                            DWORD         data);
    int             MessageBox(const char far * text,
                               const char far * caption = 0,
                               UINT          type = MB_OK);
#if defined(__WIN32__)
    HANDLE          GetWindowTask() const;
#else
    HTASK           GetWindowTask() const;
#endif
    void            DragAcceptFiles(BOOL accept);

protected:
    //
    // these events are processed by TWindow
    //
    void            EvClose();
    int             EvCreate(CREATESTRUCT far& createStruct);
    void            EvDestroy();
    LRESULT         EvCompareItem(UINT ctrlId, COMPAREITEMSTRUCT far&
compareInfo);
    void            EvDeleteItem(UINT ctrlId,DELETEITEMSTRUCT far& deleteInfo);
    void            EvDrawItem(UINT ctrlId,DRAWITEMSTRUCT far& drawInfo);
    void            EvMeasureItem(UINT ctrlId,MEASUREITEMSTRUCT far& measureInfo);
    void            EvHScroll(UINT scrollCode,UINT thumbPos,HWND hWndCtl);
    void            EvVScroll(UINT scrollCode,UINT thumbPos,HWND hWndCtl);
    void            EvMove(TPoint & ClientOrigin);
    void            EvNCDestroy();
    BOOL            EvQueryEndSession();
    void            EvSize(UINT sizeType,TSize& size);
    void            EvLButtonDown(UINT modKeys, TPoint& point);
    BOOL            EvEraseBkgnd(HDC);
    void            EvPaint();
    void            EvSysColorChange();
    LRESULT         EvWin32CtlColor(WPARAM,LPARAM);
    void            CmExit();  // CM_EXIT
    //
    // input validation message handler
    //
    void            EvChildInvalid(HWND hWnd);
    //
    // system messages
    //
    void            EvCompacting(UINT compactRatio);
```

```
    void            EvDevModeChange(char far * devName);
    void            EvEnable(BOOL enabled);
    void            EvFontChange();
    int             EvPower(UINT powerEvent);
    void            EvSysCommand(UINT cmdType,TPoint& point);
    void            EvSystemError(UINT error);
    void            EvTimeChange();
    void            EvTimer(UINT timerId);
    void            EvWinIniChange(char far * section);

    //
    // window manager messages
    //
    void            EvActivate(UINT active,
                               BOOL minimized,
                               HWND hWndOther /* may be 0 */);
#if defined(__WIN32__)
    void            EvActivateApp(BOOL active, HANDLE threadId);
#else
    void            EvActivateApp(BOOL active,HTASK hTask);
#endif
    void            EvCancelMode();
    void            EvGetMinMaxInfo(MINMAXINFO far& minmaxinfo);
    void            EvIconEraseBkgnd(HDC hDC);
    void            EvKillFocus(HWND hWndGetFocus/* may be 0 */);
    UINT            EvMouseActivate(HWND hWndTopLevel,
                                    UINT hitTestCode,
                                    UINT msg);
#if defined(__WIN32__)
    void            EvInputFocus(BOOL gainingFocus);
    void            EvOtherWindowCreated(HWND hWndOther);
    void            EvOtherWindowDestroyed(HWND hWndOther);
    void            EvPaintIcon();
#endif
    void            EvParentNotify(UINT event,
                                   UINT childHandleOrX,
                                   UINT childIDOrY);
    HANDLE          EvQueryDragIcon();
    BOOL            EvQueryOpen();
    BOOL            EvSetCursor(HWND hWndCursor,
                                UINT hitTest,
                                UINT mouseMsg);
    void            EvSetFocus(HWND hWndLostFocus /* may be 0 */);
    void            EvShowWindow(BOOL show,UINT status);
    void            EvWindowPosChanged(WINDOWPOS far& windowPos);
    void            EvWindowPosChanging(WINDOWPOS far& windowPos);

    //
    // keyboard input
    //
    void            EvChar(UINT key,UINT repeatCount,UINT flags);
    void            EvDeadChar(UINT deadKey,UINT repeatCount,UINT flags);
    void            EvKeyDown(UINT key,UINT repeatCount,UINT flags);
    void            EvKeyUp(UINT key,UINT repeatCount,UINT flags);
    void            EvSysChar(UINT key,UINT repeatCount,UINT flags);
```

```
    void            EvSysDeadChar(UINT key,UINT repeatCount,UINT flags);
    void            EvSysKeyDown(UINT key,UINT repeatCount,UINT flags);
    void            EvSysKeyUp(UINT key,UINT repeatCount,UINT flags);

    //
    // hot keys
    //
#if defined(__WIN32__)
    void            EvHotKey(int idHotKey);
#endif

    //
    // controls
    //
    HBRUSH          EvCtlColor(HDC hDC,HWND hWndChild,UINT ctlType);

    //
    // mouse input
    //
    void            EvLButtonDblClk(UINT modKeys,TPoint& point);
    void            EvLButtonUp(UINT modKeys,TPoint& point);
    void            EvMButtonDblClk(UINT modKeys,TPoint& point);
    void            EvMButtonDown(UINT modKeys,TPoint& point);
    void            EvMButtonUp(UINT modKeys,TPoint& point);
    void            EvMouseMove(UINT modKeys,TPoint& point);
    void            EvRButtonDblClk(UINT modKeys,TPoint& point);
    void            EvRButtonDown(UINT modKeys,TPoint& point);
    void            EvRButtonUp(UINT modKeys,TPoint& point);

    //
    // menu related messages
    //
    void            EvInitMenu(HMENU hMenu);
    void            EvInitMenuPopup(HMENU hPopupMenu,
                                    UINT index,
                                    BOOL sysMenu);
    UINT            EvMenuChar(UINT nChar,UINT menuType,HMENU hMenu);
    void            EvMenuSelect(UINT menuItemId,UINT flags,HMENU hMenu);

    //
    // dialog messages
    //
    void            EvEnterIdle(UINT source,HWND hWndDlg);
    UINT            EvGetDlgCode();

    //
    // print manager messages
    //
    void            EvSpoolerStatus(UINT jobStatus,UINT jobsLeft);

    //
    // clipboard messages
    //
    void            EvAskCBFormatName(UINT bufLen,char far * buffer);
    void            EvChangeCBChain(HWND hWndRemoved,HWND hWndNext);
    void            EvDrawClipboard();
    void            EvDestroyClipboard();
    void            EvHScrollClipboard(HWND hWndCBViewer,
```

```
                                    UINT scrollCode,
                                    UINT pos);
    void            EvPaintClipboard(HWND hWnd,HANDLE hPaintStruct);
    void            EvRenderAllFormats();
    void            EvRenderFormat(UINT dataFormat);
    void            EvSizeClipboard(HWND hWndViewer,HANDLE hRect);
    void            EvScrollClipboard(HWND hWndCBViewer,
                                      UINT scrollCode,
                                      UINT pos);

    //
    // palette manager messages
    //
    void            EvPaletteChanged(HWND hWndPalChg);
    void            EvPaletteIsChanging(HWND hWndPalChg);
    BOOL            EvQueryNewPalette();

    //
    // drag-n-drop messages
    //
    void            EvDropFiles(TDropInfo dropInfo);

    //
    // list box messages
    //
    int             EvCharToItem(UINT key,HWND hWndListBox,UINT caretPos);
    int             EvVKeyToItem(UINT key,HWND hWndListBox,UINT caretPos);

    //
    // non-client messages
    //
    BOOL            EvNCActivate(BOOL active);
    UINT            EvNCCalcSize(BOOL calcValidRects, NCCALCSIZE_PARAMS far&
params);
    BOOL            EvNCCreate(CREATESTRUCT far& createStruct);
    UINT            EvNCHitTest(TPoint& point);
    void            EvNCLButtonDblClk(UINT hitTest,TPoint& point);
    void            EvNCLButtonDown(UINT hitTest,TPoint& point);
    void            EvNCLButtonUp(UINT hitTest,TPoint& point);
    void            EvNCMButtonDblclk(UINT hitTest,TPoint& point);
    void            EvNCMButtonDown(UINT hitTest,TPoint& point);
    void            EvNCMButtonUp(UINT hitTest,TPoint& point);
    void            EvNCMouseMove(UINT hitTest,TPoint& point);
    void            EvNCPaint();
    void            EvNCRButtonDblClk(UINT hitTest,TPoint& point);
    void            EvNCRButtonDown(UINT hitTest,TPoint& point);
    void            EvNCRButtonUp(UINT hitTest,TPoint& point);

protected:
    void *          TransferBuffer;
    HACCEL          hAccel;
    TModule *       CursorModule;
    TResId          CursorResId;
    HCURSOR         HCursor;
    DWORD           BkgndColor;

    //
    // Constructor & subsequent initializer for use with virtual derivations
```

```
  // Immediate derivatives must call Init() before constructions are done.
  //
  TWindow();
  void                Init(TWindow * parent,const char far * title,TModule * module);
  virtual void        GetWindowClass(WNDCLASS& wndClass);
  virtual char far *  GetClassName();
  virtual void        SetupWindow();
  virtual void        CleanupWindow();
  void                DispatchScroll(UINT scrollCode,UINT thumbPos,HWND hWndCtrl);
  void                LoadAcceleratorTable();
  virtual void        RemoveChild(TWindow * child);
private:
  WNDPROC             Thunk;        // Thunk that load 'this' into registers
  TApplication *      Application;  // Application that this window belongs to
  TModule *           Module;       // default module used for getting resources
  DWORD               Flags;
  WORD                CreateOrder;
  TWindow *           ChildList;
  TWindow *           SiblingList;
  DWORD               UniqueId;
  static DWORD        LastUniqueId;
  void                Init(TWindow * parent,TModule * module);
  BOOL                OrderIsI(TWindow * win,void * position);
  BOOL                CreateZeroChild(TWindow * win,void * );
  void                AssignCreateOrder();
  void                AddChild(TWindow * child);
  int                 IndexOf(TWindow * child);
  TWindow *           At(int position);
  void                SetUniqueId();
  //
  // hidden to prevent accidental copying or assignment
  //
  TWindow(const TWindow&);
  TWindow& operator =(const TWindow&);
  DECLARE_RESPONSE_TABLE(TWindow);
  DECLARE_STREAMABLE(_OWLCLASS,TWindow,1);
};  // end of class TWindow
```

TWindow 类有一个庞大的声明，它的成员函数能完成下列操作：

- 通过部分或全部子窗口来重复操纵这些窗口。
- 设置并查询窗口的标志。
- 允许或禁止自动创建诸如控件之类的子窗口。
- 向子窗口发送或从子窗口接收数据。
- 访问本应用程序类的实例。
- 创建或销毁窗口元素。
- 管理向窗口发送 Windows 消息的过程。
- 控制窗口的颜色外观及光标。