续表

数据类型	含　义
BYTE	无符号 8 位整数
DWORD	无符号 32 位整数或段/偏移地址
UNIT	无符号 32 位整数，为将来兼容用
FAR	数据类型属性，用于创建 far 指针
FARPROC	指向函数的长指针
HANDLE	通用句柄
HINSTANCE	替换 Borland C++3.1 中的 HANDLE
HDC	显示上下文的句柄
HWIND	窗口句柄
LONG	同 long 类型
LPSTR	指向字符串的长指针
LPVOID	指向 void 类的长指针
PWORD	指向 WORD 类的指针
WORD	无符号 16 位整数

除了表 1.1 中列出的简单数据类型外，Windows 程序还包括许多结构。其中有两个简单但很重要的结构是：POINT 和 RECT. POINT 结构存储了一个点的 X 和 Y 坐标，声明如下：

```
struct POINT{
        int x;
        int y;
};
```

RECT 结构定义了矩形的左上和右下角坐标，声明如下：

```
struct RECT{
        int left;
        int top;
        int right;
        int bottom;
}
```

1.2 ObjectWindows 数据类型定义约定

ObjectWindows 使用了一种数据类型定义约定，使得更易区别数据类型。表 1.2 列出了数据类型定义约定的通用语法。表 1.3 描述了 ObjectWindows 类常用的某些数据类型，它们中间隐含了数据类型定义约定。

表 1.2 ObjectWindows 数据类型定义约定

约定	含　义
Pclass	指向 class 类的指针类型
Rclass	class 类的引用类型
RPclass	class 类指针的引用
PCclass	指向 const class 类的指针类型
RCclass	const class 类的引用类型

表 1.3 隐含数据类型定义约定的 ObjectWindows 类常用数据类型

数据类型	含　义
PCchar	指向 char 常量的指针
Pchar	指向 char 类型的指针
PCvoid	指向 void 常量的指针
Pint	指向 int 类型的指针
Pvoid	指向 void 类型的指针
RCObject	常 Object 的引用
Rint	int 类型的引用
Ripstream	ispstream 类型的引用
Ropstream	opstream 类型的引用
RPvoid	void 类型指针的引用

1.3 ObjectWindows 层次结构

ObjectWindows 库是类的层次结构，它使应用程序用户界面的创建成为一个整体。图 1.1 给出了 ObjectWindows 类的层次结构。Borland C++ 3.1 将类 TBWindow，TBButton，TBCheckBox 和其他一些未列入图 1.1 的类作为 ObjectWindows 层次结构的一部分。在 Borland C++ 3.0 中，它们也是可获得的，但是被作为 bonus 类。这个ObjectWindows 层次结构有两个基类：

类 Object，基本基类

类 TStreamable，第二基类

只有 TModule 类和它的子类 TApplication 是由 Object 类唯一派生的。其他ObjectWindows 类均继承自两个基类。有了这种多重继承，类就可以被包括在输入/输出流中以永久支持对象。

有了 ObjectWindows 层次结构，你就可以在创建 Windows 应用程序的用户界面时大大减少代码长度。本节给出各个 ObjectWindows 成员的类声明，并简要讨论每个类支持的功能。

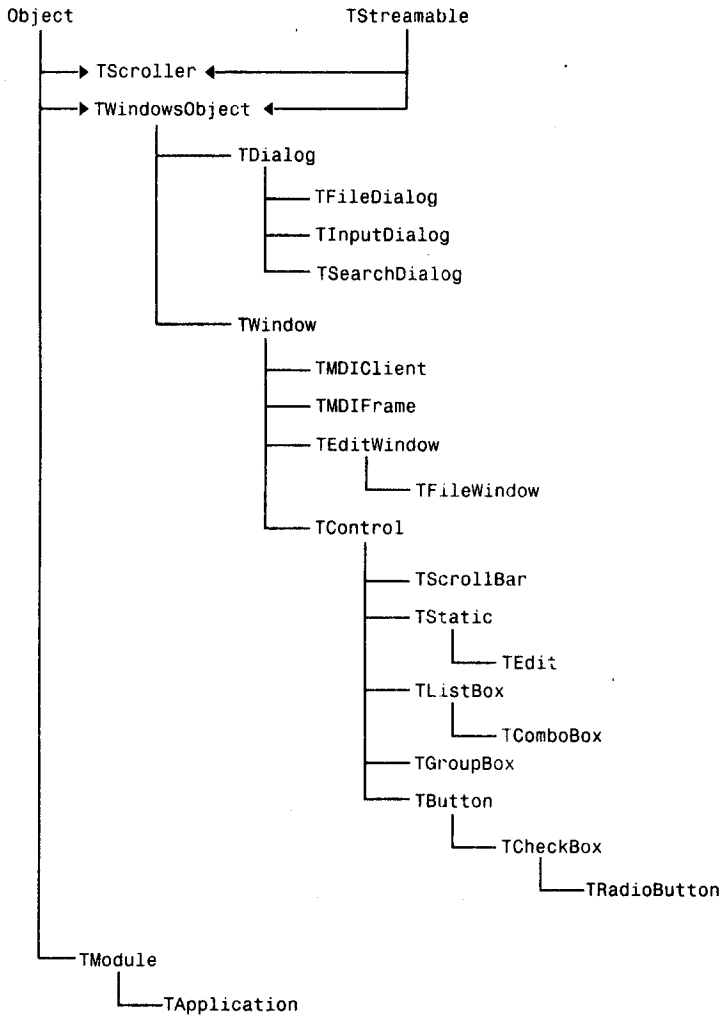

图 1.1 ObjectWindows 类层次结构

1.3.1 Object 类

Object 类是 ObjectWindows 层次结构的根。因此，Object 就是一个抽象类，它定义了通用于 ObjectWindows 类的一般结构和功能。下面给出 Object 类的声明：

```
class _CLASSTYPE Object {
public:
    virtual ~Object() {}
    virtual classType isA() const = 0;
    virtual char _FAR *nameOf() const = 0;
    virtual hashValueType hashValue() const = 0;
    virtual int isEqual(const Object _FAR &) const = 0;
```

```
    virtual int isSortable() const { return 0; }
    virtual int isAssociation() const { return 0; }
    void _FAR *operator new(size_t s);
    virtual void forEach(iterFuncType, void _FAR *);
    virtual Object _FAR & firstThat(condFuncType, void _FAR *) const;
    virtual Object _FAR & lastThat(condFuncType, void _FAR *) const;
    virtual void printOn(ostream _FAR &) const = 0;
    static Object _FAR *ZERO;
    static Object _FAR & ptrToRef(Object _FAR *p)
        { return p == 0 ? *ZERO : *p; }
    friend ostream _FAR& operator << (ostream _FAR&,
                                      const Object _FAR&);
};
```

这一声明表明 Object 类不是一个具有最少成员个数的高度抽象的对象。一个高度抽象的对象应该使用虚方式。Object 不是这样。相反，这个类定义了一些虚成员函数，它们提供了包容类的特性。这些成员函数包括 forEach，firstThat，lastThat，isSortable 和 isAssociation。静态指针类型的数据成员 ZERO 是分配失败时由新操作返回的地址。isA 和 nameOf 成员函数是抽象函数，它们返回类分类和某一类名。这些成员函数在查询类名或实例的分类时有用。

注意：_CLASSTYPE 标识符是一个特殊的宏，因为它根据正在使用的数据模型扩展为 huge，far 或 near。

注意：ObjectWindows 类声明通常包括_CLASSDEF 宏。这个宏声明了与已被声明的类有关的指针和引用 typedefs。

_CLASSDEF 宏使用一系列其他的宏定义了这些 typedefs，结果产生如下声明：

```
typedef classname _FAR * Pclassname
typedef classname _FAR & Rclassname
typedef classname _FAR * _FAR & RPclassname
typedef const classname _FAR * PCclassname
typedef const classname _FAR & RCclassname
```

其中_FAR 宏在适当的时候被自动定义为 far。如果设置了__DL __AKG _CLASSDLL 宏，则将_FAR 宏扩展为 far。

1.3.2 TModule 类

TModule 类是 Object 类的直接后继。TModule 给出了动态连接库(DDLs)的模型，并将其成员用于 ObjectWindows 应用程序(通过它的子类 TApplication)。下面是 TModule 类的声明：

```
class _EXPORT TModule : public Object {
public:
    // Lib and WinMain args
```

```
    HINSTANCE hInstance;
    LPSTR   lpCmdLine;
    int Status;
    LPSTR Name;
    TModule(LPSTR AName, HINSTANCE AnInstance, LPSTR ACmdLine);
    virtual ~TModule();
BOOL LowMemory();
void RestoreMemory();
virtual PTWindowsObject ValidWindow(PTWindowsObject AWindowsObject);
virtual PTWindowsObject MakeWindow(PTWindowsObject AWindowsObject);
virtual int ExecDialog(PTWindowsObject ADialog);
HWND GetClientHandle(HWND AnHWindow);
virtual PTWindowsObject GetParentObject(HWND ParentHandle);
virtual void Error(int ErrorCode);
// define pure virtual functions derived from Object class
virtual classType isA() const

    { return moduleClass; }
virtual Pchar nameOf() const
    { return "TModule"; }
virtual hashValueType hashValue() const
    { return hashValueType(hInstance); }
virtual int isEqual(RCObject module)  const
    { return (hInstance == ((RTModule)module).hInstance); }
virtual void printOn(Rostream outputStream) const
    { outputStream << nameOf() << "{ hInstance = "
        << hInstance << " }\n"; }
};
```

如果创建了 DDL(设置了__DL__),由将 EXPORT 宏扩展为_EXPORT。否则,_EXPORT宏将扩展为_CLASSTYPE。

类构造函数创建了 Tmodule 的一个实例,其参数为应用程序/DDL 名、应用程序/DDL 句柄和命令行。

成员函数 LowMemory 和 RestoreMemory 完成内存管理任务。MakeWindow 成员函数创建一个窗口,该过程包括使对象有效的 ValidWindow 成员函数。在成员函数 ValidWindow 验证了被激活对象的合法性之后,ExecDialog 成员函数执行一个模式对话框。ExecDialog 函数与被激活对话框的 Execute 成员函数一起工作。还要注意 isA 和 nameOf 成员函数分别返回 moduleClass 的枚举值和"TModule"串。

1.3.3 TApplication 类

TApplication 类是 TModule 类的子类,也是你所开发的每个 ObjectWindows 应用程序的父类。TApplication 的成员和由 TModule 继承来的成员一起提供了基本的数据成分以及操作以支持最小的 Windows 应用程序。这种应用程序给出一个窗口的特征,并具有一个最小的命令集。

下面清单给出 TApplication 类的声明:

```
class _EXPORT TApplication : public TModule {
public:
    // WinMain arguments
    HINSTANCE hPrevInstance;
    int    nCmdShow;
    PTWindowsObject MainWindow;

    HINSTANCE HAccTable;
    PTWindowsObject KBHandlerWnd;
    TApplication(LPSTR AName, HINSTANCE AnInstance,
                 HINSTANCE APrevInstance, LPSTR ACmdLine, int ACmdShow);
    ~TApplication();
    virtual void Run();
    virtual BOOL CanClose();
    void SetKBHandler(PTWindowsObject AWindowsObject);
    // define pure virtual functions derived from Object class
    virtual classType isA() const
        { return applicationClass; }
    virtual Pchar nameOf() const
        { return "TApplication"; }
protected:
    virtual void InitApplication();  // "first"-instance initialization
    virtual void InitInstance();     // each-instance initialization
    virtual void InitMainWindow();   // init application main window
    virtual void MessageLoop();
    /* IdleAction may be redefined in derived classes to do
    some action when there are no messages pending. */
    virtual void IdleAction() {}
    virtual BOOL ProcessAppMsg(LPMSG PMessage);
    virtual BOOL ProcessDlgMsg(LPMSG PMessage);
    virtual BOOL ProcessAccels(LPMSG PMessage);
    virtual BOOL ProcessMDIAccels(LPMSG PMessage);
};
```

类构造函数根据所提供的应用程序名、实例句柄、前一实例的句柄(如果不存在则为0)、命令行串和一个决定如何显示窗口(正常,最小化或最大化)的整数码,创建一个TApplication 的实例。多重成员函数初始化 TApplication 的实例:

- InitMainWindows 函数初始化 ObjectWindows 应用程序的主窗口。
- InitApplication 函数初始化 ObjectWindows 应用程序的第一个实例。
- InitInstance 函数初始化 ObjectWindows 应用程序的每一个实例。

TApplication 实例和它们的后继使用一套成员函数处理各种消息。Windows 消息由MessageLoop 函数处理,正如其名所示,它监督着应用程序的消息循环。MessageLoop 函数和 Process 成员函数一起处理即将到来的消息。IAction 函数使应用程序在不忙时能进行后台处理(如数据排序)。ProcessDlgMsg 管理某些非模式对话框和窗口消息处理以操作键盘输入。Run 成员函数初始化并执行 ObjectWindows 应用程序的实例。

1.3.4 TWindowsObject 类

TWindowsObject 是一个从 Object 和 TStreamable 得来的非常的抽象类。

TWindowsObject 类是 ObjectWindows 层次结构中各种界面类的父类。这包括窗口、对话框和控制。下面清单是 TWindowsObject 的声明。正如声明所示，TWindowsObject 是一个具有公共、保护和私有数据成员和成员函数的复杂类。

```
class _EXPORT TWindowsObject : public Object, public TStreamable { public:
    int Status;
    HWND HWindow;  // handle to associated MS-Windows window
    LPSTR Title;
    PTWindowsObject Parent;
    TWindowsObject(PTWindowsObject AParent, PTModule AModule = NULL);
    virtual ~TWindowsObject();
    void SetFlags(WORD Mask, BOOL OnOff);
    /* Determines whether the flag whose mask is passed has been set,
       returning a BOOL indicator --  True = On, False = Off. */
    BOOL IsFlagSet(WORD Mask) { return((Flags & Mask) == Mask); }
    PTWindowsObject FirstThat(TCondFunc Test, Pvoid PParamList);
    void ForEach(TActionFunc Action, Pvoid PParamList);
    PTWindowsObject FirstThat(
          BOOL (TWindowsObject::* _FAR Test)(Pvoid, Pvoid),
          Pvoid PParamList);
    void ForEach(
          void (TWindowsObject::* _FAR Action)(Pvoid, Pvoid),
          Pvoid PParamList);
    PTWindowsObject Next();
    PTWindowsObject GetFirstChild() { return ChildList->SiblingList;}
    PTWindowsObject GetLastChild() { return ChildList; }
    PTWindowsObject Previous();
    void EnableKBHandler();
    void EnableAutoCreate();
    void DisableAutoCreate();
    void EnableTransfer();
    void DisableTransfer();
    PTModule GetModule() { return Module; }
    PTApplication GetApplication() { return Application; }
    FARPROC GetInstance() { return Instance; }
    virtual BOOL Register();
/* Pure virtual function, placeholder for derived classes */
/* to redefine to create an MS_Windows element to be associated */
/* with an OWL window object */
    virtual BOOL Create() = 0;
    virtual void Destroy();
    virtual int GetId();
    PTWindowsObject ChildWithId(int Id);
    virtual PTMDIClient GetClient();
    virtual void SetParent(PTWindowsObject NewParent);
    void Show(int ShowCmd);
    void SetCaption(LPSTR ATitle);
    virtual BOOL CanClose();
    void SetTransferBuffer(Pvoid ATransferBuffer)
        { TransferBuffer = ATransferBuffer; }
    virtual WORD Transfer(Pvoid DataPtr, WORD TransferFlag);
    virtual void TransferData(WORD Direction);
    virtual void DefWndProc(RTMessage Msg);
```

```
    virtual void BeforeDispatchHandler() {}
    virtual void AfterDispatchHandler() {}
    virtual void DispatchAMessage(WORD AMsg, RTMessage AMessage,
        void (TWindowsObject::* _FAR)(RTMessage));
    void CloseWindow();
    void GetChildren(Ripstream is);
    void PutChildren(Ropstream os);
    BOOL CreateChildren();
    virtual void ShutDownWindow();
    virtual void DrawItem(DRAWITEMSTRUCT far & DrawInfo);
    virtual void ActivationResponse(WORD Activated, BOOL IsIconified);
    // define pure virtual functions derived from Object class
    virtual classType isA() const = 0;
    virtual Pchar nameOf() const = 0;
    virtual hashValueType  hashValue() const
        { return hashValueType(HWindow); }
    virtual int isEqual(RCObject testwin) const
        { return this ==  &(RTWindowsObject)testwin; }
    virtual void printOn(Rostream outputStream) const
        { outputStream << nameOf() << "{ HWindow = "
                       << HWindow << " }\n"; }
    static PTStreamable build();
protected:
    FARPROC DefaultProc;
    Pvoid TransferBuffer;
    virtual void GetWindowClass(WNDCLASS _FAR & AWndClass);
    virtual LPSTR GetClassName() = 0;
    void RemoveClient()
       { RemoveChild((PTWindowsObject)GetClient()); }
    void GetChildPtr(Ripstream is, RPTWindowsObject P);
    void PutChildPtr(Ropstream os, PTWindowsObject P);
    void GetSiblingPtr(Ripstream is, RPTWindowsObject P);
    void PutSiblingPtr(Ropstream os, PTWindowsObject P);
    virtual void DefCommandProc(RTMessage Msg);
    virtual void DefChildProc(RTMessage Msg);
    virtual void DefNotificationProc(RTMessage Msg);
    virtual void SetupWindow();
    virtual void WMVScroll(RTMessage Msg) = [WM_FIRST + WM_VSCROLL];
    virtual void WMHScroll(RTMessage Msg) = [WM_FIRST + WM_HSCROLL];
    void DispatchScroll(RTMessage Msg);
    virtual void WMCommand(RTMessage Msg) = [WM_FIRST + WM_COMMAND];
    virtual void WMDrawItem(RTMessage Msg) = [WM_FIRST + WM_DRAWITEM];
    virtual void WMClose(RTMessage Msg) = [WM_FIRST + WM_CLOSE];
    virtual void WMDestroy(RTMessage Msg) = [WM_FIRST + WM_DESTROY];
    virtual void WMNCDestroy(RTMessage Msg) =
                [WM_FIRST + WM_NCDESTROY];
    virtual void WMActivate(RTMessage Msg) = [WM_FIRST + WM_ACTIVATE];
    virtual void WMQueryEndSession(RTMessage Msg) =
                [WM_FIRST + WM_QUERYENDSESSION];
    virtual void CMExit(RTMessage Msg) = [CM_FIRST + CM_EXIT];
    TWindowsObject(StreamableInit) {};
    virtual void write (Ropstream os);
    virtual Pvoid read (Ripstream is);
```

```
private:
    FARPROC Instance;
    PTApplication Application;
    PTModule Module;
    WORD Flags;
    WORD CreateOrder;
    BOOL OrderIsI(Pvoid P, Pvoid I);
    BOOL CreateZeroChild(Pvoid P, Pvoid I);
    void AssignCreateOrder();
    PTWindowsObject ChildList, SiblingList;
    void AddChild(PTWindowsObject AChild);
    void RemoveChild(PTWindowsObject AChild);
    int IndexOf(PTWindowsObject P);

    PTWindowsObject At(int APosition);
    virtual const Pchar streamableName() const
                { return "TWindowsObject"; }
};
```

TWindowsObject 类所提供的操作包括处理 ObjectWindows 应用程序的各种实例，使即将到来的消息成为一个整体并关闭窗口。

> **注意**：消息处理成员函数的声明后面跟了一个特殊的语法(如=[WM_FIRST+WM_VSCROLL])，它定义了由那个函数操作的消息。WM_XXX 标识符是无符号整常数，代表消息号和偏移地址。WM_XXX 常数和消息将在本章后面部分讨论。

1.3.5 TDialog 类

TDialog 类是 TWindowsObject 类的子类，既是用户界面类也是更特定的对话框类的父类(即 TFileDialog 文件选择、TInputDialog 行输入和 TSearchDialog 查找对话框)。TDialog 类支持表示模式和非模式对话框的实例。TDialog 的实例包括控制类的实例，如单选按钮和复选框。下面是 TDialog 类的声明：

```
class _EXPORT TDialog : public TWindowsObject {
public:
    TDialogAttr Attr;
    BOOL IsModal;
    TDialog(PTWindowsObject AParent, LPSTR AName,
            PTModule AModule = NULL);
    TDialog(PTWindowsObject AParent, int ResourceId,
            PTModule AModule = NULL);
    virtual ~TDialog();
    virtual BOOL Create();
    virtual int Execute();
    virtual void CloseWindow(int ARetValue);
    virtual void CloseWindow();
    virtual void ShutDownWindow(int ARetValue);
```

```
    virtual void ShutDownWindow();
    virtual void Destroy(int ARetValue);
    virtual void Destroy();
    void SetCaption(LPSTR ATitle);
    // Returns the handle of the dialog's control with the passed Id
    HWND GetItemHandle(int DlgItemID)
      { return GetDlgItem(HWindow, DlgItemID); }
    // Sends the passed message to the dialog box's control
    // that has the Id DlgItemID.
    DWORD SendDlgItemMsg(int DlgItemID, WORD AMsg,
                         WORD WParam, DWORD LParam)
    { return SendDlgItemMessage(HWindow, DlgItemID,
                                AMsg, WParam, LParam); }
    virtual classType isA() const { return dialogClass; }
    virtual Pchar nameOf() const { return "TDialog"; }
    static PTStreamable build();
protected:
    virtual void GetWindowClass(WNDCLASS _FAR & AWndClass);
    virtual LPSTR GetClassName();
    virtual void SetupWindow();
    virtual void Ok(RTMessage Msg) = [ID_FIRST + IDOK];
    virtual void Cancel(RTMessage Msg) = [ID_FIRST + IDCANCEL];
    virtual void WMInitDialog(RTMessage Msg) =
                [WM_FIRST + WM_INITDIALOG];
    virtual void WMQueryEndSession(RTMessage Msg) =
                [WM_FIRST + WM_QUERYENDSESSION];
    virtual void WMClose(RTMessage Msg) = [WM_FIRST + WM_CLOSE];
    TDialog(StreamableInit) : TWindowsObject(streamableInit) {};
    virtual void write (Ropstream os);
    virtual Pvoid read (Ripstream is);
private:
    virtual const Pchar streamableName() const { return "TDialog"; }
};
```

TDialog类有三个构造函数。前两个构造函数除第二个参数不同外均相同。共同的参数是指向父窗口对象的指针和指向相关模块的指针。第一个构造函数包括用于创建对话框的资源名。第二个构造函数中，第二个参数是一个与整数类型等价的资源ID。第三个类构造函数从流所提供的数据创建了TDialog的实例。

TDialog的模式实例是由Execute成员函数创建(更好地，应由更安全的TModule::ExecuteDialog成员函数创建)。模式对话框被创建和执行。只有在用户以它们结束时，它们才返回对其他窗口的控制。你用TModule::MakeWindow成员函数创建非模式对话框的实例。TDialog类使用Destroy成员函数去掉对话框。

1.3.6 TFileDialog类

TFileDialog类给对话框带来了一个实用的而且是非常需要的限制。TFileDialog的实例使你能够为各种输入和输出操作选择一个文件。下面是TFileDialog的声明：

```
class _EXPORT TFileDialog : public TDialog {
public:
    LPSTR FilePath;
    char PathName[MAXPATH];
    char Extension[MAXEXT];
    char FileSpec[FILESPEC];
    TFileDialog(PTWindowsObject AParent, int ResourceId,
                LPSTR AFilePath, PTModule AModule = NULL);
    virtual BOOL CanClose();
    void SelectFileName();
    void UpdateFileName();
    BOOL UpdateListBoxes();
    static PTStreamable build();
protected:
    virtual void SetupWindow();
    virtual void HandleFName(RTMessage Msg) = [ID_FIRST + ID_FNAME];
    virtual void HandleFList(RTMessage Msg) = [ID_FIRST + ID_FLIST];
    virtual void HandleDList(RTMessage Msg) = [ID_FIRST + ID_DLIST];
    TFileDialog(StreamableInit) : TDialog(streamableInit) {};
private:
    virtual const Pchar streamableName() const
                { return "TFileDialog"; }
};
```

TFileDialog 类有两个构造函数。第一个构造函数使你能够从头开始创建一个实例。AFilePath 参数提供包含查找文件的路径。ResourceID 参数使你能够指定对话框是操作文件输入还是输入。第二个类构造函数使你能够从流读入数据来创建一个实例。

1.3.7 TInputDialog 类

TInputDialog 类实现一个通用目的的对话框，你可以向其中打入文本。下面清单是 TInputDialog 类的声明：

```
class _EXPORT TInputDialog : public TDialog {
public:
    LPSTR Prompt;
    LPSTR Buffer;
    WORD BufferSize;
    TInputDialog(PTWindowsObject AParent, LPSTR ATitle,
                 LPSTR APrompt, LPSTR ABuffer, WORD ABufferSize,
                 PTModule AModule = NULL);
    void TransferData(WORD Direction);
    static PTStreamable build();
protected:
    virtual void SetupWindow();
    TInputDialog(StreamableInit) : TDialog(streamableInit) {};
    virtual void write (Ropstream os);
    virtual Pvoid read (Ripstream is);
private:
    virtual const Pchar streamableName() const
        { return "TInputDialog"; }
};
```

公共的类构造函数具有参数 ATitle，APrompt 和 ABuffer，它们分别指定了对话框标题，提示信息和缺省输入。使用保护的类构造函数，你可以用从流读入的方法来创建实例。

1.3.8 TSearchDialog 类

TSearchDialog 类创建了常用的文本查找和替换对话框。这个类是其父类 TDialog 的实用限定的又一个例子。下面是 TSearchDialog 类的声明。

```
class  EXPORT TSearchDialog: public TDialog {
public:
    TSearchDialog(PTWindowsObject AParent, int ResourceId,
                  TSearchStruct _FAR &SearchStruct,
                  PTModule AModule = Null);
};
```

注意：TSearchDialog 的声明只包括一个类构造函数。这一短小的声明证明了使用面向对象编程方法可以减少代码长度。TSearchDialog 类继承了所有所需的数据成员和成员函数。TSearchDialog 使用 TSearchStruct 结构与查找对话框之间传送数据。这一特性使你能够保留最近的输入。

1.3.9 TWindow 类

TWindow 类是 TwindowsObject 的子类，它实现了类属窗口。这些窗口包括标题、控制菜单(也叫系统菜单)和最大化/最小化图符。你可以移动、重新定义大小、最小化和最大化 TWindow 的实例。下面是 TWindow 类的声明：

```
class _EXPORT TWindow : public TWindowsObject {
public:
    TWindowAttr Attr;
    PTScroller Scroller;
    HINSTANCE FocusChildHandle;
    TWindow(PTWindowsObject AParent, LPSTR ATitle,
            PTModule AModule = NULL);
    TWindow(HWND AnHWindow, PTModule AModule = NULL);
    virtual ~TWindow();
    virtual BOOL AssignMenu(LPSTR MenuName);
    virtual BOOL AssignMenu(int MenuId);
    virtual BOOL Create();
    virtual void ActivationResponse(WORD Activated, BOOL IsIconified);
    virtual classType isA() const { return windowClass; }
    virtual Pchar nameOf() const { return "TWindow"; }
    static PTStreamable build();
protected:
    virtual LPSTR GetClassName()
        { return "OWLWindow"; }
```

```
    virtual void GetWindowClass(WNDCLASS _FAR & AWndClass);
    virtual void SetupWindow();
    virtual void WMCreate(RTMessage Msg) = [WM_FIRST + WM_CREATE];
    virtual void WMMDIActivate(RTMessage Msg) =
                [WM_FIRST + WM_MDIACTIVATE];

    virtual void WMHScroll(RTMessage Msg) = [WM_FIRST + WM_HSCROLL];
    virtual void WMVScroll(RTMessage Msg) = [WM_FIRST + WM_VSCROLL];
    virtual void WMPaint(RTMessage Msg) = [WM_FIRST + WM_PAINT];
    virtual void Paint(HDC PaintDC, PAINTSTRUCT _FAR & PaintInfo);
    virtual void WMSize(RTMessage Msg) = [WM_FIRST + WM_SIZE];
    virtual void WMMove(RTMessage Msg) = [WM_FIRST + WM_MOVE];
    virtual void WMLButtonDown(RTMessage Msg) =
                [WM_FIRST + WM_LBUTTONDOWN];
    TWindow(StreamableInit) : TWindowsObject(streamableInit) {};
    virtual void write (Ropstream os);
    virtual Pvoid read (Ripstream is);
private:
    virtual const Pchar streamableName() const { return "TWindow"; }
};
```

Attr 数据成员存储 TWindow 实例的属性。另一个有趣的数据成员是 Scroller，它是一个指向帮助滚动窗口内容的 TScroller 类的指针。你可以通过赋予 Attr。Style 成员以特殊的风格来创建可视的垂直和水平滚动条。这些属性由 TWindowAttr 结构给出，其声明如下：

```
struct _CLASSTYPE TWindowAttr {
    DWORD Style;
    DWORD ExStyle;
    int X, Y, W, H;      // dimensions
    LPSTR Menu;          // Menu name
    int Id;              // Child identifier
    LPSTR Param;
};
```

这个结构定义了风格、扩展风格、位置、大小、菜单句柄和控制 ID。

TWindow 类包括一些成员函数，它们响应有关创建、绘制、移动、定义大小、水平滚动、垂直滚动、按下鼠标和激活-MDI-窗口的消息。

所幸登记是一个“透明”的过程——即，ObjectWindows 类是在后台中处理这一过程的。第三章“创建基本窗口”将向你演示如何细微调整窗口的登记。

注意：Create 成员函数管理窗口的登记和创建。登记是一个重要的过程，它要求以下步骤：

1. 声明一个 WNDCLASS 类型的变量。
2. 给你声明过的这个 WNDCLASS 类型的变量的数据域赋值。
3. 将指向这个 WNDCLAS 类型的变量的指针作为唯一的参数调用 RegisterClass 函数。

1.3.10 TControl 类

TControl 类是 TWindow 类的子类。它为它自己的派生类提供了共同的数据成员和成员函数。这些派生类实现了特定的视觉控制(如组合框、按钮、列表框、复选框和单选按钮)。下面是 TControl 类的声明:

```
class _EXPORT TControl : public TWindow {
public:
    TControl(PTWindowsObject AParent, int AnId, LPSTR ATitle, int X,
            int Y, int W, int H, PTModule AModule = NULL);
    TControl(PTWindowsObject AParent, int ResourceId,
            PTModule AModule = NULL);
    virtual int GetId() { return Attr.Id; }
protected:
    TControl(StreamableInit) : TWindow(streamableInit) {};
    virtual void WMPaint(RTMessage Msg) = [WM_FIRST + WM_PAINT];
    virtual void WMDrawItem(RTMessage Msg) = [WM_FIRST + WM_DRAWITEM];
    virtual void ODADrawEntire(DRAWITEMSTRUCT far & DrawInfo);
    virtual void ODAFocus(DRAWITEMSTRUCT far & DrawInfo);
    virtual void ODASelect(DRAWITEMSTRUCT far & DrawInfo);
};
```

TControl 类声明的成员函数相对来说较少。这些函数处理绘制和绘画消息。在用户可画的控制被选、被作为焦点、未被作为焦点或需要画时,这些函数管理该控制的外观。

1.3.11 TScrollBar 类

TScrollBar 是 TControl 的子类,给窗口提供了垂直和水平滚动条。每个滚动条都定义了一定范围的值,并使用视觉滚动条的姆指框来显示当前被选的值。你可以以小的增量(行增量)或大的增量(页增量)来移动滚动条的姆指框。下面清单声明了 TScrollBar 类:

```
class _EXPORT TScrollBar : public TControl {
public:
    int LineMagnitude, PageMagnitude;
    TScrollBar(PTWindowsObject AParent, int AnId, int X, int Y, int W,
              int H, BOOL IsHScrollBar, PTModule AModule = NULL);
    TScrollBar(PTWindowsObject AParent, int ResourceId,
              PTModule AModule = NULL);
    void GetRange(Rint LoVal, Rint HiVal);
    int GetPosition();
    void SetRange(int LoVal, int HiVal);
    void SetPosition(int ThumbPos);
    int DeltaPos(int Delta);
    virtual WORD Transfer(Pvoid DataPtr, WORD TransferFlag);
    static PTStreamable build();
protected:
    virtual LPSTR GetClassName() { return "SCROLLBAR"; }
    virtual void SetupWindow();
    virtual void SBLineUp(RTMessage Msg) = [NF_FIRST + SB_LINEUP];
    virtual void SBLineDown(RTMessage Msg) = [NF_FIRST + SB_LINEDOWN];
    virtual void SBPageUp(RTMessage Msg) = [NF_FIRST + SB_PAGEUP];
```

```
    virtual void SBPageDown(RTMessage Msg) = [NF_FIRST + SB_PAGEDOWN];
    virtual void SBThumbPosition(RTMessage Msg) =
                                    [NF_FIRST + SB_THUMBPOSITION];
    virtual void SBThumbTrack(RTMessage Msg) =
                                    [NF_FIRST + SB_THUMBTRACK];
    virtual void SBTop(RTMessage Msg) = [NF_FIRST + SB_TOP];
    virtual void SBBottom(RTMessage Msg) = [NF_FIRST + SB_BOTTOM];
    TScrollBar(StreamableInit) : TControl(streamableInit) {};
    virtual void write (Ropstream os);
    virtual Pvoid read (Ripstream is);
private:
    virtual const Pchar streamableName() const { return "TScrollBar"; }
};
```

数据成员 LineMagnitude 和 PageMagnitude 分别存储了滚动条姆指框行和页增量的移动距离。成员函数 GetRange 和 GetPosition 分别返回值的范围和当前姆指框值。SetRange,SetPosition 和 DeltaPos 成员函数分别设置了值的范围、新的姆指框绝对位置和新姆指框相对位置。SBXXX 成员函数族按行、按页上下移动姆指框(对水平滚动条来说是左右移动),或直接移到滚动条的末尾。

1.3.12 TStatic 类

TStatic 类是 TControl 的子类,它为窗口和对话框控制提供了静态文本对象。请始终记住静态文本不是刻在石头上的文字。如果 Windows 应用程序允许,你可以在内部改变静态文本。下面是 TStatic 类的声明:

```
class _EXPORT TStatic : public TControl {
public:
    WORD TextLen;
    TStatic(PTWindowsObject AParent, int AnId, LPSTR ATitle,
            int X, int Y, int W, int H, WORD ATextLen,
            PTModule AModule = NULL);
    TStatic(PTWindowsObject AParent, int ResourceId, WORD ATextLen,
            PTModule AModule = NULL);
    /* Returns the length of the control's text. */
    int GetTextLen() { return GetWindowTextLength(HWindow); }
    /* Fills the passed string with the text of the associated text
       control.  Returns the number of characters copied. */
    int GetText(LPSTR ATextString, int MaxChars)
        { return GetWindowText(HWindow, ATextString, MaxChars); }
    /* Sets the contents of the associated static text control to
       the passed string. */
    void SetText(LPSTR ATextString)
        { SetWindowText(HWindow, ATextString); }
    /* Clears the text of the associated static text control. */
    void Clear() { SetText(""); }
    virtual WORD Transfer(Pvoid DataPtr, WORD TransferFlag);
    virtual Pchar nameOf() const { return "TStatic"; }
    static PTStreamable build();
protected:
```

```
    virtual LPSTR GetClassName() { return "STATIC"; }
    TStatic(StreamableInit) : TControl(streamableInit) {};
    virtual void write (Ropstream os);
    virtual Pvoid read (Ripstream is);
private:
    virtual const Pchar streamableName() const { return "TStatic"; }
};
```

头两个类构造函数创建了与 TWindowsObject 的后继(包括 TWindow,TDialog 和它们的后继)相关的静态文本。这些构造函数使你能够将这一文本赋给 TStatic 的实例,SetText 成员函数使你能够改变这一文本。TStatic 实例附属于(或属于)TWindowsObject 的后继。GetText 和 GetTextLen 成员函数分别返回 TStatic 实例的文本及其长度。

1.3.13 TEdit 类

TEdit 类是 TStatic 的子类,它实现了一种编辑控制,可以编辑窗口的内容。编辑控制可以处理单行和多行(在多行中,行之间是由连续的 ASCII 10 和 ASCII 13 字符分隔的)。下面是 TEdit 类的声明:

```
class _EXPORT TEdit : public TStatic {
public:
    TEdit(PTWindowsObject AParent, int AnId, LPSTR AText, int X,
          int Y, int W, int H, WORD ATextLen, BOOL Multiline,
          PTModule AModule = NULL);
    TEdit(PTWindowsObject AParent, int ResourceId, WORD ATextLen,
          PTModule AModule = NULL);
    void Undo();
    BOOL CanUndo();
    void Paste();
    void Copy();
    void Cut();
    int GetNumLines();
    int GetLineLength(int LineNumber);
    BOOL GetLine(LPSTR ATextString, int StrSize, int LineNumber);
    void GetSubText(LPSTR ATextString, int StartPos, int EndPos);
    BOOL DeleteSubText(int StartPos, int EndPos);
    BOOL DeleteLine(int LineNumber);
    void GetSelection(Rint StartPos, Rint EndPos);
    BOOL DeleteSelection();
    BOOL IsModified();
    void ClearModify();
    int GetLineFromPos(int CharPos);
    int GetLineIndex(int LineNumber);
    void Scroll(int HorizontalUnit, int VerticalUnit);
    BOOL SetSelection(int StartPos, int EndPos);
    void Insert(LPSTR ATextString);
    int Search(int StartPos, LPSTR AText, BOOL CaseSensitive);

    static PTStreamable build();
protected:
```