第 6 章　视

SDI和MDI应用程序的视是用户与应用程序(特别是应用程序正在编辑的文档)交互的主要方式。本章的所有实例都涉及到视，从创建属性表的视、打印视到向视拖放文件等等。这些实例包括：

实例15　创建标签窗体视，本例将创建具有属性表的视。

实例16　创建一个简单组合框的视，作为视的通用控件，本例中将看到如何创建控件视(按钮、列表框和编辑框都是控件)。

实例17　打印报表，使用MFC CView类的内置功能打印报表。

实例18　打印视，自动地捕获应用程序的屏幕图像然后打印。

实例19　绘制MDI客户视(MDI Client View)，在本例中将看到一种为MDI应用程序的普通背景添加颜色和模式的方法。背景是MDI的客户视，是MDI应用程序的视驻足的地方。

实例20　拖放文件到视，然后打开拖放到视中的文件。

6.1　实例15：创建标签窗体视

1. 目标

创建一个标签窗体视，使其包含几个对话框模板，如图 6-1所示。

2. 策略

从CScrollView派生自己的视类，在这个视类中简单地创建表单控件窗口。

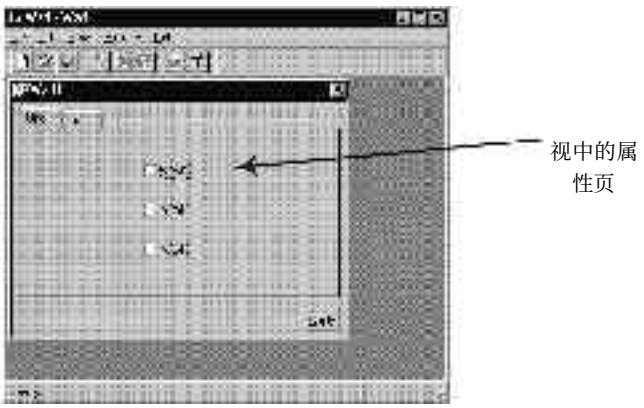

图6-1　标签窗体视

3. 步骤

1) 为标签窗体视创建页面

用Dialog Editor创建一个或者一个以上的对话框模板，使之成为标签视的页面。并将它们的风格设置为细边框的子窗口。这些页面的窗口标题将成为出现在其标签中的名字。

使用Class Wizard为这些页面创建类并将其从CPropertyPage类派生。

2) 创建标签窗体视类

用Class Wizard创建派生于CScrollView的新类。

在该新类中嵌入属性表变量，如下所示：

```
// WzdTabbedView.h
private:
   CPropertySheet m_sheet;
```

在该类中再嵌入一个按钮控件，它将成为 Apply按钮：

```
CButton  m_button;
```

属性表一般创建自己的Apply按钮，但这仅在创建有模式对话框的情况下。因此必须动态地为该视类增加自己的Apply按钮。

用Class Wizard为该类添加WM_CREATE消息处理函数。在此创建该属性表和Apply按钮，并将先前创建的属性页类添加到该属性表之中:

```
int CWzdTabbedView::OnCreate( LPCREATESTRUCT lpCreateStruct )
{
   if ( CScrollView::OnCreate( lpCreateStruct ) == -1 )
      return -1;

   // create property sheet
   m_sheet.AddPage( &m_pageOne );
   m_sheet.AddPage( &m_pageTwo );
   m_sheet.Create( this,WS_CHILD|WS_VISIBLE );

   // create apply button
   CRect rect ( 0,0,10,10 );
   CFont *pFont =
      CFont::FromHandle( ( HFONT )::GetStockObject( ANSI_VAR_FONT ) );
   m_button.Create( "&Apply", WS_VISIBLE | WS_CHILD, rect, this,
      IDC_WZD_APPLY );
   m_button.SetFont( pFont );

   return 0;
}
```

注意，现在不必关心这些控件放在哪儿或者有多大，以后将移动它们。

用Class Wizard重载该新View类的OnInitUpdate()函数。首先移动属性表和Apply按钮并且改变其大小。然后用 CScrollView::ResizeParentToFit()函数缩小包围它们的视，在此首先用SetScrollSize()函数告诉CScollView视的大小:

```
void CWzdTabbedView::OnInitialUpdate()
{
   CScrollView::OnInitialUpdate();

   // home the property sheet
   CRect rect;
   m_sheet.GetClientRect( &rect );
   m_sheet.MoveWindow( rect );
   rect.bottom += BUTTON_HEIGTH + 10;

   // move apply button into place
   CRect brect( rect.right-BUTTON_WIDTH-5,
      rect.bottom-BUTTON_HEIGTH-5,
      rect.right-5,
      rect.bottom-5 );
   m_button.MoveWindow( brect );

   // size child frame around property sheet
   SIZE size = { rect.Width(), rect.Height() };
```

```
    SetScrollSizes( MM_TEXT, size );
    ResizeParentToFit( FALSE );

    // make sure the scroll bars are gone
    SetScrollSizes( MM_TEXT, CSize( 20,20 ) );

    // copy document into property pages
    m_pageOne.m_bWzd1 = ( ( CWzdDoc* )GetDocument() ) -> m_bWzd1;
    m_pageTwo.m_sEdit = ( ( CWzdDoc* )GetDocument() ) -> m_sEdit;

}
```

最后要完成的工作是用文档中的值初始化属性页的值，这将涉及许多其他的工作。

由于属性表和按钮控件并不覆盖所有的视，所以需要绘制视的背景。为此，用 ClassWizard为该类添加WM_ERASEBKGND消息处理函数，并用系统按钮的表面颜色如下绘制背景：

```
BOOL CWzdTabbedView::OnEraseBkgnd(CDC* pDC)
{
    CPen pen( PS_SOLID, 0, GetSysColor( COLOR_BTNFACE ) );
    CPen *pPen = pDC -> SelectObject( &pen );
    CBrush brush( GetSysColor( COLOR_BTNFACE ) );
    CBrush *pBrush = pDC -> SelectObject( &brush );

    CRect rect;
    GetClientRect( &rect );
    pDC -> Rectangle( rect );
    pDC -> SelectObject( pPen );
    pDC -> SelectObject( pBrush );
    return TRUE;
}
```

为Apply按钮添加消息处理函数。这可以通过手工来完成，或者使用与按钮相同的命令标识符创建Apply菜单选项，然后用ClassWizard自动地为自己的视类增加消息处理函数，并通知属性表更新其页面。即使属性表以无模式对话框的方式使用 (在这种情况下没有Apply按钮)，也可以通过CPropertySheet的PressButton()函数按照下述方法访问该按钮的功能:

```
void CWzdTabbedView::OnWzdApply()
{
    m_sheet.PressButton( PSBTN_APPLYNOW );

    ( ( CWzdDoc* )GetDocument() ) -> m_bWzd1 = m_pageOne.m_bWzd1;
    ( ( CWzdDoc* )GetDocument() ) -> m_sEdit = m_pageTwo.m_sEdit;
}
```

最后要完成的工作是从属性页中将变量值拷贝到文档，这里也有许多其他工作要做。

参考程序清单 —— 标签窗体视类可查看该类的完整代码列表。

3) 实现新标签窗体视类

在应用程序类的InitInstance()函数中替换新类，如下所示：

```
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
```

```
    IDR_WZDTYPE,
    RUNTIME_CLASS( CWzdDoc ),
    RUNTIME_CLASS( CChildFrame ),
    RUNTIME_CLASS( CWzdTabbedView ) );    <<<<<<<< ADD HERE
AddDocTemplate( pDocTemplate );
```

可以删除由App Wizard所创建的任何视类。

为了避免用户重新设置视的大小，需要修改 MainFrame或者ChildFrame类的PreCreate Window()函数：

```
BOOL CChildFrame::PreCreateWindow( CREATESTRUCT& cs )
{
    // removes min/max boxes
    cs.style &= ~( WS_MAXIMIZEBOX|WS_MINIMIZEBOX );

    // makes dialog box unsizable
    cs.style &= ~WS_THICKFRAME;

    return CMDIChildWnd::PreCreateWindow( cs );
}
```

4. 注意

为什么要从 CScrollView类而不是从 CView类派生新类呢？因为这样做可以使用CScrollView:: ResizeParentToFit()函数来缩小包围属性表和按钮的视。

5. 使用光盘时注意

执行随书附带光盘中的工程，可以注意到视实际上是包含属性页的属性表。

6. 程序清单——标签窗体视类

```
#if !defined(AFX_WZDTABBEDVIEW_H__9A0B9504_E043_11D1_9B77_00AA003D8695__INCLUDED_)
#define AFX_WZDTABBEDVIEW_H__9A0B9504_E043_11D1_9B77_00AA003D8695__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif    // _MSC_VER >= 1000
// WzdTabbedView.h : header file
//

#include "PageOne.h"
#include "PageTwo.h"

/////////////////////////////////////////////////////////////////////////////
// CWzdTabbedView view

class CWzdTabbedView : public CScrollView
{
protected:
    CWzdTabbedView();    // protected constructor used by dynamic creation
    DECLARE_DYNCREATE( CWzdTabbedView )

// Attributes
public:
```

```
// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    // {{AFX_VIRTUAL( CWzdTabbedView )
protected:
    virtual void OnDraw( CDC* pDC );            // overridden to draw this view
    virtual void OnInitialUpdate();             // first time after construct
    // }}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CWzdTabbedView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump( CDumpContext& dc ) const;
#endif

    // Generated message map functions
    // {{AFX_MSG(CWzdTabbedView)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnWzdApply();
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    // }}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CButton m_button;
    CPageOne m_pageOne;
    CPageTwo m_pageTwo;
    CPropertySheet m_sheet;
};

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

// {{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
//     before the previous line.

#endif
    //!defined(AFX_WZDTABBEDVIEW_H_9A0B9504_E043_11D1_9B77_00AA003D8695__INCLUDED_)
// WzdTabbedView.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdDoc.h"
#include "WzdTabbedView.h"

#ifdef _DEBUG
```

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

#define BUTTON_WIDTH 40
#define BUTTON_HEIGTH 25

////////////////////////////////////////////////////////////////////////////
// CWzdTabbedView

IMPLEMENT_DYNCREATE( CWzdTabbedView, CScrollView )

CWzdTabbedView::CWzdTabbedView()
{
}

CWzdTabbedView::~CWzdTabbedView()
{
}


BEGIN_MESSAGE_MAP( CWzdTabbedView, CScrollView )
    // {{AFX_MSG_MAP( CWzdTabbedView )
    ON_WM_CREATE()
    ON_COMMAND( IDC_WZD_APPLY, OnWzdApply )
    ON_WM_ERASEBKGND()
    // }}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
// CWzdTabbedView drawing

void CWzdTabbedView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();

    // home the property sheet
    CRect rect;
    m_sheet.GetClientRect( &rect );
    m_sheet.MoveWindow( rect );
    rect.bottom += BUTTON_HEIGTH + 10;

    // move apply button into place
    CRect brect( rect.right-BUTTON_WIDTH-5,
        rect.bottom-BUTTON_HEIGTH-5,
        rect.right-5,
        rect.bottom-5);
    m_button.MoveWindow( brect );

    // size child frame around property sheet
```

```
    SIZE size = {rect.Width(), rect.Height()};
    SetScrollSizes( MM_TEXT, size );
    ResizeParentToFit( FALSE );

    // make sure the scroll bars are gone
    SetScrollSizes( MM_TEXT, CSize( 20,20 ) );

    // copy document into property pages
    m_pageOne.m_bWzd1 = ( ( CWzdDoc* )GetDocument() ) -> m_bWzd1;
    m_pageTwo.m_sEdit = ( ( CWzdDoc* )GetDocument() ) -> m_sEdit;

}

void CWzdTabbedView::OnDraw( CDC* pDC )
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

/////////////////////////////////////////////////////////////////////////////
// CWzdTabbedView diagnostics

#ifdef _DEBUG
void CWzdTabbedView::AssertValid() const
{
    CScrollView::AssertValid();
}

void CWzdTabbedView::Dump(CDumpContext& dc) const
{
    CScrollView::Dump( dc );
}
#endif    //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CWzdTabbedView message handlers

int CWzdTabbedView::OnCreate( LPCREATESTRUCT lpCreateStruct )
{
    if ( CScrollView::OnCreate( lpCreateStruct ) == -1 )
    return -1;

    // create property sheet
    m_sheet.AddPage( &m_pageOne );
    m_sheet.AddPage( &m_pageTwo );
    m_sheet.Create( this,WS_CHILD|WS_VISIBLE );

    // create apply button
    CRect rect ( 0,0,10,10 );
    CFont *pFont = CFont::FromHandle( ( HFONT )::GetStockObject( ANSI_VAR_FONT ) );
    m_button.Create( “&Apply”, WS_VISIBLE | WS_CHILD, rect, this, IDC_WZD_APPLY );
```

```
    m_button.SetFont( pFont );

    return 0;
}

void CWzdTabbedView::OnWzdApply()
{
    m_sheet.PressButton( PSBTN_APPLYNOW );

    ( ( CWzdDoc* )GetDocument() ) -> m_bWzd1 = m_pageOne.m_bWzd1;
    ( ( CWzdDoc* )GetDocument() ) -> m_sEdit = m_pageTwo.m_sEdit;
}

BOOL CWzdTabbedView::OnEraseBkgnd( CDC* pDC )
{
    CPen pen( PS_SOLID,0,GetSysColor( COLOR_BTNFACE ) );
    CPen *pPen = pDC -> SelectObject( &pen );
    CBrush brush( GetSysColor( COLOR_BTNFACE ) );
    CBrush *pBrush = pDC -> SelectObject( &brush );

    CRect rect;
    GetClientRect( &rect );
    pDC -> Rectangle( rect );
    pDC -> SelectObject( pPen );
    pDC -> SelectObject( pBrush );
    return TRUE;
}
```

6.2 实例16：创建具有通用控件的视

1. 目标

创建具有任何通用控件的视。本实例将创建具有图 6-2所示简单组合框的视。

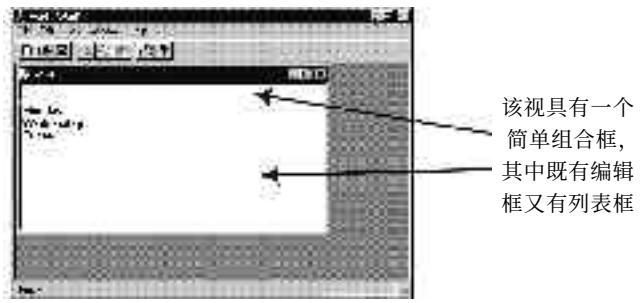

图6-2 简单组合框的视

2. 策略

本例将在标准的 CView类中嵌入通用控件的MFC类(本例使用CComboBox)，并添加两个消息处理函数以迫使该控件窗口占据视。WM_CREATE消息处理函数将创建该控件窗口。WM_SIZE消息处理函数则扩展该控件窗口以填充视，然后重载 OnInitialUpdate()和OnUpdate()函数为控件赋予文档值。

3. 步骤

1) 创建组合框视

使用AppWizard按照通常方式创建应用程序，并以缺省的 CView类创建自己的视类。

在视类中嵌入组合框：

```
CComboBox  m_combobox;
```

用ClassWizard添加一个WM_CREATE 消息处理函数到自己的视类中，并按照下列风格创建简单的组合框：

```
int CWzdView::OnCreate( LPCREATESTRUCT lpCreateStruct )
{
   if ( CView::OnCreate( lpCreateStruct ) = -1 )
      return -1;

   CRect rect( 0,0,0,0 );
   m_combobox.Create( WS_CHILD| WS_VISIBLE| CBS_SIMPLE|
      CBS_NOINTEGRALHEIGHT | WS_VSCROLL,
      rect, this, IDC_WZD_COMBOBOX );

   return 0;
}
```

要转变为视的任何控件窗口都应该具有 WS_CHILD和WS_VISIBLE风格设置。其他风格则随着控件的不同而不同。

由于简单组合框的列表框总是可见的，因此使用 CBS_SIMPLE风格创建简单的组合框。否则，视将总是一个具有下拉按钮的编辑框。

用CLassWizard为视类添加WM_SIZE消息处理函数。用CWnd::MoveWindow()函数来扩展控件的大小以填充视：

```
void CWzdView::OnSize( UINT nType, int cx, int cy )
{
   CView::OnSize( nType, cx, cy );

   m_combobox.MoveWindow( 0,0,cx,cy );
}
```

2) 更新组合框视

用ClassWizard重载CView的InInitialUpdate()和OnUpdate()成员函数。将用文档中的数据填充控件：

```
void CWzdView::OnInitialUpdate()
{
   CView::OnInitialUpdate();

   // get initial data from document
   m_combobox.AddString( "Monday" );
   m_combobox.AddString( "Wednesday" );
   m_combobox.AddString( "Friday" );

}

void CWzdView::OnUpdate( CView* pSender, LPARAM lHint, CObject* pHint )
{
   // in response to UpdateAllViews from document
}
```

为控件通知消息手工添加处理函数(例如，通知视列表框的选择项发生变化或者用户在编辑框中进行了击键操作等)。确定应该加入内容的简单方法是创建对话框然后为它增加一个简单的组合框，用ClassWizard创建对话框类并添加视类需要的处理函数到该对话框类。最后只需从中剪切或者粘贴所需要的消息宏。

参考本实例结尾的程序清单—组合框视以了解该类的全部源代码。

4. 注意

本实例的一个改进是用两个或者两个以上的控件窗口创建一个视。举例说，可以在顶部创建一个静态控件窗口以显示某些静态文本，在底部则创建一个编辑框来显示某些数据。通过在视类中嵌入这两种控件类可以做到这一点，它们都由 WM_CREATE消息处理函数创建，然后在WM_SIZE消息处理函数中决定每个控件窗口得到多少视区域。

5. 使用光盘时注意

执行随书附带的光盘中的工程时，可以注意到视由一个简单的组合框填充。

6. 程序清单 —— 组合框视类

```
// WzdView.h : interface of the CWzdView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined( AFX_WZDVIEW_H__CA9038F0_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_)
#define AFX_WZDVIEW_H__CA9038F0_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif    // _MSC_VER >= 1000

class CWzdView : public CView
{
protected: // create from serialization only
    CWzdView();
    DECLARE_DYNCREATE( CWzdView )

// Attributes
public:
    CWzdDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    // {{AFX_VIRTUAL( CWzdView )
public:
    virtual void OnDraw( CDC* pDC );    // overridden to draw this view
    virtual BOOL PreCreateWindow( CREATESTRUCT& cs );
    virtual void OnInitialUpdate();
protected:
    virtual BOOL OnPreparePrinting( CPrintInfo* pInfo );
    virtual void OnBeginPrinting( CDC* pDC, CPrintInfo* pInfo );
```

```
    virtual void OnEndPrinting( CDC* pDC, CPrintInfo* pInfo );
    virtual void OnUpdate( CView* pSender, LPARAM lHint, CObject* pHint );
    // }}AFX_VIRTUAL

// Implementation
public:
    virtual ~CWzdView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump( CDumpContext& dc ) const;
#endif

protected:

// Generated message map functions
protected:
    // {{AFX_MSG( CWzdView )
    afx_msg int OnCreate( LPCREATESTRUCT lpCreateStruct );
    afx_msg void OnSize( UINT nType, int cx, int cy );
    // }}AFX_MSG
    afx_msg void OnEditchangeCombo();
    afx_msg void OnEditupdateCombo();
    afx_msg void OnSelchangeCombo();
    DECLARE_MESSAGE_MAP()
private:
    CComboBox m_combobox;
};

#ifndef _DEBUG    // debug version in WzdView.cpp
inline CWzdDoc* CWzdView::GetDocument()
   { return ( CWzdDoc* )m_pDocument; }
#endif

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

// {{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
//     before the previous line.

#endif
    // !defined( AFX_WZDVIEW_H__CA9038F0_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_ )
// WzdView.cpp : implementation of the CWzdView class
//

#include "stdafx.h"
#include "Wzd.h"

#include "WzdDoc.h"
#include "WzdView.h"

#ifdef _DEBUG
```

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

/////////////////////////////////////////////////////////////////////////////
// CWzdView

IMPLEMENT_DYNCREATE( CWzdView, CView )

BEGIN_MESSAGE_MAP( CWzdView, CView )
    // {{AFX_MSG_MAP( CWzdView )
    ON_WM_CREATE()
    ON_WM_SIZE()
    // }}AFX_MSG_MAP
    ON_CBN_EDITCHANGE( IDC_WZD_COMBOBOX, OnEditchangeCombo )
    ON_CBN_EDITUPDATE( IDC_WZD_COMBOBOX, OnEditupdateCombo )
    ON_CBN_SELCHANGE( IDC_WZD_COMBOBOX, OnSelchangeCombo )
    // Standard printing commands
    ON_COMMAND( ID_FILE_PRINT, CView::OnFilePrint )
    ON_COMMAND( ID_FILE_PRINT_DIRECT, CView::OnFilePrint )
    ON_COMMAND( ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview )
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWzdView construction/destruction

CWzdView::CWzdView()
{
    // TODO: add construction code here

}

CWzdView::~CWzdView()
{
}

BOOL CWzdView::PreCreateWindow( CREATESTRUCT& cs )
{
    // TODO: Modify the Window class or styles here by modifying
    //     the CREATESTRUCT cs

    return CView::PreCreateWindow( cs );
}

/////////////////////////////////////////////////////////////////////////////
// CWzdView drawing

void CWzdView::OnDraw( CDC* pDC )
{
    CWzdDoc* pDoc = GetDocument();
```

```
    ASSERT_VALID( pDoc );

    // TODO: add draw code for native data here
}

////////////////////////////////////////////////////////////////////////////
// CWzdView printing

BOOL CWzdView::OnPreparePrinting( CPrintInfo* pInfo )
{
    // default preparation
    return DoPreparePrinting( pInfo );
}

void CWzdView::OnBeginPrinting( CDC* /*pDC*/, CPrintInfo* /*pInfo*/ )
{
    // TODO: add extra initialization before printing
}

void CWzdView::OnEndPrinting( CDC* /*pDC*/, CPrintInfo* /*pInfo*/ )
{
    // TODO: add cleanup after printing
}

////////////////////////////////////////////////////////////////////////////
// CWzdView diagnostics

#ifdef _DEBUG
void CWzdView::AssertValid() const
{
    CView::AssertValid();
}

void CWzdView::Dump( CDumpContext& dc ) const
{
    CView::Dump( dc );
}

CWzdDoc* CWzdView::GetDocument()    // non-debug version is inline
{
    ASSERT( m_pDocument -> IsKindOf( RUNTIME_CLASS( CWzdDoc ) ) );
    return ( CWzdDoc* )m_pDocument;
}
#endif    //_DEBUG

////////////////////////////////////////////////////////////////////////////
// CWzdView message handlers

int CWzdView::OnCreate( LPCREATESTRUCT lpCreateStruct )
{
    if ( CView::OnCreate( lpCreateStruct ) == -1 )
```

```
        return -1;

    CRect rect( 0,0,0,0 );
    m_combobox.Create( WS_CHILD|WS_VISIBLE|CBS_SIMPLE|CBS_NOINTEGRALHEIGHT|WS_ VSCROLL,
        rect, this, IDC_WZD_COMBOBOX );

    return 0;
}

void CWzdView::OnSize( UINT nType, int cx, int cy )
{
    CView::OnSize( nType, cx, cy );

    m_combobox.MoveWindow( 0,0,cx,cy );
}

void CWzdView::OnInitialUpdate()
{
    CView::OnInitialUpdate();

    // get initial data from document
    m_combobox.AddString( “Monday” );
    m_combobox.AddString( “Wednesday” );
    m_combobox.AddString( “Friday” );

}

void CWzdView::OnUpdate( CView* pSender, LPARAM lHint, CObject* pHint )
{
    // in response to UpdateAllViews from document
}

void CWzdView::OnEditchangeCombo()
{
    // TODO: Add your control notification handler code here
}

void CWzdView::OnEditupdateCombo()
{
    // TODO: Add your control notification handler code here
}

void CWzdView::OnSelchangeCombo()
{
    // TODO: Add your control notification handler code here
}
```

6.3 实例17：打印报表

1. 目标

如图6-3所示打印一个文档中的报表信息。

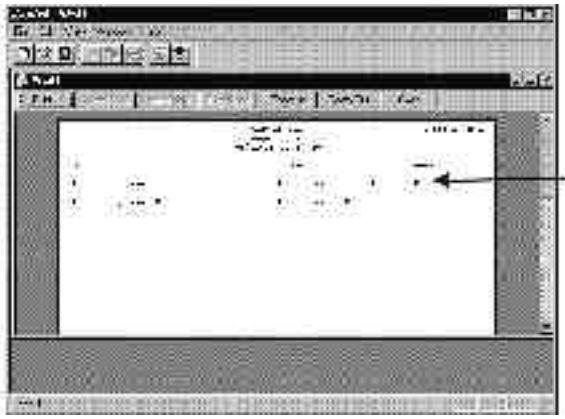

图6-3 打印报表预览

2. 策略

MFC应用程序中打印报表习惯上在视类中实现，可以认为这是查看文档的另一方式。File/Print和File/Print Preview菜单命令由视类自动地处理。为了充分利用这个自动功能则必须重载两个视类函数，它们是OnBeginPrint()，负责打印前初始化打印变量，以及OnPrint()，该函数在每次新的页面需要被打印时即被调用。在此将遍历文档的数据集合并将其绘制到打印设备环境。本实例中还将提示用户要打印的报表类型。

3. 步骤

1) 提示用户要打印的报表类型

编写一个小型的帮助函数可以提示用户要打印的报表类型，将该函数添加到视类中。下面的实例简单地创建一个对话框询问用户想要一个简短的还是一个较长的报表。实际编写时可以询问更多的细节：

```
BOOL CWzdView::GetReportOptions()
{
   CWzdDialog dlg;
   dlg.m_nReportType = m_nReportType;
   if( dlg.DoModal() == IDOK )
   {
      m_nReportType = dlg.m_nReportType;
      return TRUE;
   }
   return FALSE;
}
```

最好在用户单击了Print或者Print Preview菜单按钮后用对话框提示用户。但是，如果要直接截获这些命令(ID_FILE_PRINT和ID_FILE_PRINGT_PREVIEW)并在对话框内放置其消息处理函数，那么在用户预览打印页的时候可能会出问题。当用户使用预览页面上的打印按钮直接打印该页面时，将会发出另一个ID_FILE_PRINT命令而再次打开该提示对话框。为了避免出现这样的问题，在发送出IDC_FILE_PRINT和IDC_FILE_PRINT_PREVIEW消息前必须修改主菜单并发送两个新的命令消息(IDC_FILE_PRINT和IDC_FILE_PRINT_PREVIEW)以提

示用户报表类型。

用菜单编辑器 (Menu Editor)分别改变File/Print和File/Print Preview的命令标识符为IDC_FILE_PRINT和IDC_FILE_PRINT_PREVIEW。

用ClassWizard为视类增加这两个消息的命令消息处理函数。从中可以调用GetReportOptions()函数，它将打开对报表类型的提示对话框，然后生成使得视类进入打印模式的命令标识符:

```
void CWzdView::OnFilePrint()
{
   if ( GetReportOptions() )
   {
      AfxGetMainWnd() -> SendMessage( WM_COMMAND, ID_FILE_PRINT );
   }
}

void CWzdView::OnFilePrintPreview()
{
   if ( GetReportOptions() )
   {
      AfxGetMainWnd() ->
         SendMessage( WM_COMMAND, ID_FILE_PRINT_PREVIEW );
   }
}
```

GetReportOptions()函数的代码可以在程序清单—视类中找到。

2) 初始化视类以供打印

填充视类的OnBeginPrint()成员函数。由于AppWizard已经添加了这个函数，因此不再需要Class Wizard。在打印开始之前将调用一次 OnBeginPrint()。该函数将计算某些供打印需要的数值，包括平均打印字符大小、打印行允许字符数目、每页可能行数以及报表所需要的整个打印页数等:

```
void CWzdView::OnBeginPrinting( CDC* pDC, CPrintInfo* pInfo )
{
   // get printed character height and width
   TEXTMETRIC tm;
   pDC -> GetTextMetrics( &tm );
   m_nPrintCharHeight = tm.tmHeight;
   m_nPrintCharWidth = tm.tmAveCharWidth;

   // get number of characters per line
   int nPageWidth = pDC -> GetDeviceCaps( HORZRES );
   m_nPageWidth = nPageWidth/m_nPrintCharWidth;

   // get number of lines per page (with and w/o title on each page)
   int nPageHeight = pDC -> GetDeviceCaps( VERTRES );
   int nPrintLinesPerPage = nPageHeight / m_nPrintCharHeight;
   m_nPrintableLinesPerPage =
      nPrintLinesPerPage-LINES_IN_REPORT_TITLE;

   // determine number of total pages in this document
```

```
    int nLines = GetDocument() -> GetWzdInfoList() -> GetCount();
    int nPages = ( nLines + m_nPrintableLinesPerPage - 1 )/
        m_nPrintableLinesPerPage;
    if ( nPages <= 0 ) nPages = 1;
    pInfo -> SetMaxPage( nPages );
    pInfo -> m_nCurPage = 1;
}
```

3) 打印页面

用ClassWizard重载视类中的OnPrint()成员函数。一旦每次准备打印新的页面， OnPrint ()函数将由视类调用直到打印结束。因此 OnPrint()函数必须要做的一件事就是确定文档中页面从哪儿开始。为了打印本实例中的报表， OnPrint()函数调用了3个辅助函数： PrintTitle()打印报表标题， PrintColumnHeader()打印报表的列头，而PrintLine()则打印每一行:

```
void CWzdView::OnPrint( CDC* pDC, CPrintInfo* pInfo )
{
    // print title
    int y = 0;
    PrintTitle( pDC,&y,pInfo );

    // print column headers
    PrintColumnHeaders( pDC,&y );

    // determine part of document to print on this page
    int nWzdInfoListInx =
        ( pInfo -> m_nCurPage-1 )*m_nPrintableLinesPerPage;
    int nWzdInfoListEnd =
        GetDocument() -> GetWzdInfoList() -> GetCount();
    if ( nWzdInfoListEnd > nWzdInfoListInx + m_nPrintableLinesPerPage )
    {
        nWzdInfoListEnd = nWzdInfoListInx + m_nPrintableLinesPerPage;
    }

    // print report lines
    for ( ; nWzdInfoListInx < nWzdInfoListEnd; nWzdInfoListInx++ )
    {
        POSITION pos = GetDocument() ->
            GetWzdInfoList() -> FindIndex(nWzdInfoListInx);
        CWzdInfo *pInfo =
            GetDocument() -> GetWzdInfoList() -> GetAt( pos );
        PrintLine( pDC,&y,pInfo );
        y += m_nPrintCharHeight;
    }

    CView::OnPrint( pDC, pInfo );
}
```

将这3个辅助函数： PrintTitle()、 PrintColumnHeader()和PrintLine()添加到视类中。PrintTitle()将使用CDC类的TextOut成员函数打印标题。 PrintColumnHeader()和PrintLine()则用TabbedTextOut()函数自动地创建各列。作为这些辅助函数的例子，请参考下面的程序清单

—— 视类。

4. 注意

如果视包含了图像而不是列表或者其他文本，那就不需要为视类增加任何东西来打印该视。这是因为CView类将调用其OnDraw()成员函数，而该函数具有其打印设备环境而不是屏幕设备环境，这样系统自动地将视绘制到打印机。

为了创建其他视类类型的报表，请参考实例 18。

5. 使用光盘时注意

执行光盘中的该工程时，单击 File，然后单击是Print或者Print Preview菜单命令，将出现一个对话框询问用户是否想打印短的还是长的报表。选择一项并按下 OK按钮，可以注意到一份报表被打印到了纸上或者屏幕上。

6. 程序清单 —— 视类

```
// WzdView.h : interface of the CWzdView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined( AFX_WZDVIEW_H__CA9038F0_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_ )
#define AFX_WZDVIEW_H__CA9038F0_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif    // _MSC_VER >= 1000

class CWzdView : public CView
{
protected:    // create from serialization only
    CWzdView();
    DECLARE_DYNCREATE( CWzdView )

// Attributes
public:
    CWzdDoc* GetDocument();

    enum {
        SHORTREPORT,
        LONGREPORT
    };

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    // {{AFX_VIRTUAL( CWzdView )
public:
    virtual void OnDraw( CDC* pDC );    // overridden to draw this view
    virtual BOOL PreCreateWindow( CREATESTRUCT& cs );
protected:
    virtual BOOL OnPreparePrinting( CPrintInfo* pInfo );
```

```
    virtual void OnBeginPrinting( CDC* pDC, CPrintInfo* pInfo );
    virtual void OnEndPrinting( CDC* pDC, CPrintInfo* pInfo );
    virtual void OnPrint( CDC* pDC, CPrintInfo* pInfo );
    // }}AFX_VIRTUAL

// Implementation
public:
    virtual ~CWzdView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump( CDumpContext& dc ) const;
#endif

protected:

// Generated message map functions
protected:
    // {{AFX_MSG(CWzdView)
    afx_msg void OnFilePrint();
    afx_msg void OnFilePrintPreview();
    // }}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    int m_nPrintCharHeight;
    int m_nPrintCharWidth;
    int m_nPageWidth;
    int m_nPrintableLinesPerPage;
    int m_nReportType;
    static int m_reportTabs[];

    BOOL GetReportOptions();
    void PrintTitle( CDC *pDC, int *y, CPrintInfo* pInfo );
    void PrintColumnHeaders( CDC *pDC, int *y );
    void PrintLine( CDC *pDC, int *y, CWzdInfo *pInfo );

};

#ifndef _DEBUG    // debug version in WzdView.cpp
inline CWzdDoc* CWzdView::GetDocument()
  { return ( CWzdDoc* )m_pDocument; }
#endif

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

// {{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
//     before the previous line.

#endif
    // !defined( AFX_WZDVIEW_H__CA9038F0_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_ )
// WzdView.cpp : implementation of the CWzdView class
```

```
//

#include "stdafx.h"
#include "Wzd.h"

#include "WzdDoc.h"
#include "WzdView.h"
#include "WzdDialog.h"

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


int CWzdView::m_reportTabs[] = {1200,2000};
#define NUM_TABS sizeof(m_reportTabs)/sizeof(int)

/////////////////////////////////////////////////////////////////////////////
// CWzdView

IMPLEMENT_DYNCREATE( CWzdView, CView )

BEGIN_MESSAGE_MAP( CWzdView, CView )
    // {{AFX_MSG_MAP( CWzdView )
    ON_COMMAND( IDC_FILE_PRINT, OnFilePrint )
    ON_COMMAND( IDC_FILE_PRINT_PREVIEW, OnFilePrintPreview )
    // }}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND( ID_FILE_PRINT, CView::OnFilePrint )
    ON_COMMAND( ID_FILE_PRINT_DIRECT, CView::OnFilePrint )
    ON_COMMAND( ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview )
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWzdView construction/destruction

CWzdView::CWzdView()
{
    m_nReportType = 0;

}

CWzdView::~CWzdView()
{
}

BOOL CWzdView::PreCreateWindow( CREATESTRUCT& cs )
{
    // TODO: Modify the Window class or styles here by modifying
```

```
    //     the CREATESTRUCT cs

    return CView::PreCreateWindow( cs );
}

/////////////////////////////////////////////////////////////////////////////
// CWzdView drawing

void CWzdView::OnDraw( CDC* pDC )
{
    CWzdDoc* pDoc = GetDocument();
    ASSERT_VALID( pDoc );

    // TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
// CWzdView printing

#define LINES_IN_REPORT_TITLE 7

BOOL CWzdView::OnPreparePrinting( CPrintInfo* pInfo )
{
    // default preparation
    return DoPreparePrinting( pInfo );
}

void CWzdView::OnBeginPrinting( CDC* pDC, CPrintInfo* pInfo )
{
    // get printed character height and width
    TEXTMETRIC tm;
    pDC -> GetTextMetrics( &tm );
    m_nPrintCharHeight = tm.tmHeight;
    m_nPrintCharWidth = tm.tmAveCharWidth;

    // get number of characters per line
    int nPageWidth = pDC -> GetDeviceCaps( HORZRES );
    m_nPageWidth = nPageWidth/m_nPrintCharWidth;

    // get number of lines per page (with and w/o title on each page)
    int nPageHeight = pDC -> GetDeviceCaps( VERTRES );
    int nPrintLinesPerPage = nPageHeight / m_nPrintCharHeight;
    m_nPrintableLinesPerPage = nPrintLinesPerPage-LINES_IN_REPORT_TITLE;

    // determine number of total pages in this document
    int nLines = GetDocument() -> GetWzdInfoList() -> GetCount();
    int nPages = ( nLines + m_nPrintableLinesPerPage - 1 )/
        m_nPrintableLinesPerPage;
    if ( nPages <= 0 ) nPages = 1;
    pInfo -> SetMaxPage( nPages );
    pInfo -> m_nCurPage = 1;
```

```
}

void CWzdView::OnEndPrinting( CDC* /*pDC*/, CPrintInfo* /*pInfo*/ )
{
   // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CWzdView diagnostics

#ifdef _DEBUG
void CWzdView::AssertValid() const
{
   CView::AssertValid();
}

void CWzdView::Dump( CDumpContext& dc ) const
{
   CView::Dump( dc );
}

CWzdDoc* CWzdView::GetDocument()    // non-debug version is inline
{
   ASSERT( m_pDocument -> IsKindOf( RUNTIME_CLASS( CWzdDoc ) ) );
   return ( CWzdDoc* )m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CWzdView message handlers

void CWzdView::OnFilePrint()
{
   if ( GetReportOptions() )
   {
      AfxGetMainWnd() -> SendMessage( WM_COMMAND, ID_FILE_PRINT );
   }
}

void CWzdView::OnFilePrintPreview()
{
   if ( GetReportOptions() )
   {
      AfxGetMainWnd() -> SendMessage( WM_COMMAND, ID_FILE_PRINT_PREVIEW );
   }
}

BOOL CWzdView::GetReportOptions()
{
   CWzdDialog dlg;
   dlg.m_nReportType = m_nReportType;
```

```
    if( dlg.DoModal() == IDOK )
    {
        m_nReportType = dlg.m_nReportType;
        return TRUE;
    }
    return FALSE;
}

void CWzdView::OnPrint( CDC* pDC, CPrintInfo* pInfo )
{
    // print title
    int y = 0;
    PrintTitle( pDC,&y,pInfo );

    // print column headers
    PrintColumnHeaders( pDC,&y );

    // determine part of document to print on this page
    int nWzdInfoListInx = ( pInfo -> m_nCurPage-1 )*m_nPrintableLinesPerPage;
    int nWzdInfoListEnd = GetDocument() -> GetWzdInfoList() -> GetCount();
    if ( nWzdInfoListEnd > nWzdInfoListInx + m_nPrintableLinesPerPage )
    {
        nWzdInfoListEnd = nWzdInfoListInx + m_nPrintableLinesPerPage;
    }

    // print report lines
    for ( ; nWzdInfoListInx < nWzdInfoListEnd; nWzdInfoListInx++ )
    {
        POSITION pos =
            GetDocument() -> GetWzdInfoList() -> FindIndex( nWzdInfoListInx );
        CWzdInfo *pInfo = GetDocument() -> GetWzdInfoList() -> GetAt( pos );
        PrintLine( pDC,&y,pInfo );
        y += m_nPrintCharHeight;
    }

    CView::OnPrint( pDC, pInfo );
}


void CWzdView::PrintTitle( CDC *pDC, int *y, CPrintInfo* pInfo )
{
    // title
    int x = pDC -> GetDeviceCaps( HORZRES );
    pDC -> SetTextAlign( TA_CENTER );
    pDC -> TextOut( x/2, *y, “WZD REPORT” );

    // page #
    CString str;
    str.Format( “Page %d of %d”,pInfo -> m_nCurPage,pInfo -> GetMaxPage() );
    pDC -> SetTextAlign( TA_RIGHT );
    pDC -> TextOut( x, *y, str );
```

```
    // report type
    *y += m_nPrintCharHeight;
    pDC -> SetTextAlign( TA_CENTER );
    switch ( m_nReportType )
    {
        case CWzdView::SHORTREPORT:
            str = "Short Report";
            break;

        case CWzdView::LONGREPORT:
            str = "Long Report";
            break;

    }
    pDC -> TextOut( x/2, *y, str );

    // date
    *y += m_nPrintCharHeight;
    COleDateTime dt( COleDateTime::GetCurrentTime() );
    pDC -> TextOut( x/2, *y, dt.Format( “%c” ) );
    *y += m_nPrintCharHeight;
    *y += m_nPrintCharHeight;    // leave space between title and column headers
}


void CWzdView::PrintColumnHeaders( CDC *pDC, int *y )
{
    CString str;
    switch ( m_nReportType )
    {
        case CWzdView::SHORTREPORT:
            str = “Group\tVersion”;
            break;

        case CWzdView::LONGREPORT:
            str = “Group\tComment\tVersion”;
            break;
    }
    pDC -> SetTextAlign( TA_LEFT );
    pDC -> TabbedTextOut( 0,*y,str,NUM_TABS,m_reportTabs,0 );
    *y += m_nPrintCharHeight;
    *y += m_nPrintCharHeight;    // leave space between column headers and report
}

void CWzdView::PrintLine( CDC *pDC, int *y, CWzdInfo *pInfo )
{
    CString str;
    switch ( m_nReportType )
    {
        case CWzdView::SHORTREPORT:
```

```
            str.Format( "%s\t%d", pInfo -> m_sGroupName, pInfo -> m_nVersion );
            break;

        case CWzdView::LONGREPORT:
            str.Format( "%s\t%s\t%d",pInfo -> m_sGroupName,
                pInfo -> m_sComment, pInfo -> m_nVersion );
            break;
    }

    pDC -> SetTextAlign( TA_LEFT );
    pDC -> TabbedTextOut( 0,*y,str,NUM_TABS,m_reportTabs,0 );
    *y += m_nPrintCharHeight;
}
```

6.4 实例18：打印视

1. 目标

动态地抓取应用程序的屏幕图像并将其打印。

2. 策略

上一个实例打印的是文档的报表，而不是当前出现在屏幕上的内容。在本例中，将实现屏幕抓取并打印其内容。MFC为打印视提供了有限的支持，但只要用MFC在CView::OnDraw()中所提供的设备环境，就可以绘制自己的视。在打印视的时候，MFC只需调用具备打印机设备环境的OnDraw()函数即可。但是，如果不绘制自己的视，例如，用一个或者一个以上的控件窗口填充自己的视就不会打印任何东西。每个控件将使用自己的设备环境将自己打印到屏幕，因此打印视的唯一途径就是抓取屏幕(拷贝其内容到一个位图对象)并将其打印到打印机。由于该功能整个与位图相关，因此将该功能封装到了位图类中。

3. 步骤

1) 创建新的位图类

用ClassWizard创建一个派生自CBitmap的新类。

为该类添加在其他实例中将提到的两个函数。Capture()函数用于拷贝屏幕某区域的内容到一个位图对象，如实例33所示的那样。CreateDIB()函数则用于将一个设备位图转变成设备独立位图(DIB)，在实例34中将回顾该函数。

最后为新类增加的函数是Print()函数，在调用该函数之前，首先假设视已经被Capture()函数抓取到了位图中，为其提供了打印机设备环境之后Print()函数就会将所包含的该位图打印到打印机:

```
void CWzdBitmap::Print( CDC *pDC )
{
    // get DIB version of bitmap
    int bmData;
    HANDLE hDIB = CreateDIB( &bmData );
```

现在可以将该DIB位图拷贝到打印机设备环境中。由于打印机一般有比屏幕高得多的分辨率，因此需要扩展位图以填充空白。接下来就确定需要填充的打印机区域有多大：

```
    // stretch bitmap to fill printed page with 1/4 inch borders
    int cxBorder = pDC -> GetDeviceCaps(LOGPIXELSX)/4;
```

```
    int cyBorder = pDC -> GetDeviceCaps(LOGPIXELSY)/4;
    int cxPage = pDC -> GetDeviceCaps(HORZRES) - (cxBorder*2);
    int cyPage = ( int )( ( ( double )cxPage/
        ( double )m_nWidth ) * ( double )m_nHeight );
```

使用StretchDIBits()函数在打印机设备环境中将位图伸展。 StretchDIBits()函数需要指向DIB位图的两个指针：一个指向头，另一个指向数据：

```
LPBITMAPINFOHEADER lpDIBHdr = ( LPBITMAPINFOHEADER )::GlobalLock( hDIB );
LPSTR lpDIBBits = ( LPSTR )lpDIBHdr+bmData;
```

最后完成扩展和清除工作：

```
    // stretch the bitmap for the best fit on the printed page
    pDC -> SetStretchBltMode( COLORONCOLOR );
    ::StretchDIBits( pDC -> m_hDC,
        cxBorder,cyBorder,cxPage,cyPage,          // destination dimensions
        0,0,m_nWidth,m_nHeight,
        // source bitmap dimensions (use all of bitmap)
        lpDIBBits,                                // bitmap picture data
        ( LPBITMAPINFO )lpDIBHdr,                 // bitmap header info
        DIB_RGB_COLORS,                           // specify color table has
                                                  //     RGB values
        SRCCOPY                                   // simple source to destination copy
    );

    // cleanup
    ::GlobalUnlock( hDIB );
    ::GlobalFree( hDIB );
    return;
}
```

参考本实例结尾的程序清单—位图类可以查看该类的完整列表。

可以用该类随意地捕获屏幕并将其打印到打印机。但更好的办法是将该类加入视类，这样用户一单击File/Print或者File/Print Preview命令，视就被打印到打印机，上接下来就实现该方式。

2) 实现新的位图类

在视类中嵌入该新类：

```
CWzdBitmap m_bitmap;
```

注释掉缺省的 ID_FILE_PRINT和ID_FILE_PRINT_PREVIEW命令处理函数并使用ClassWizard来增加自己的命令处理函数。

```
BEGIN_MESSAGE_MAP( CWzdView, CView )
    // {{AFX_MSG_MAP( CWzdView )
    ON_COMMAND( ID_FILE_PRINT, OnFilePrint )
    ON_COMMAND( ID_FILE_PRINT_PREVIEW, OnFilePrintPreview )
    // }}AFX_MSG_MAP
    // Standard printing commands
    // ON_COMMAND( ID_FILE_PRINT, CView::OnFilePrint )
    ON_COMMAND( ID_FILE_PRINT_DIRECT, CView::OnFilePrint )
    // ON_COMMAND( ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview )
END_MESSAGE_MAP()
```

填充OnFilePrint()和OnFilePrintPreview()函数以使用新类的Capture()来抓取视：

```
void CWzdView::OnFilePrint()
{
   // capture our view
   CRect rect;
   GetWindowRect( &rect );
   m_bitmap.Capture( rect );

   CView::OnFilePrint();
}

void CWzdView::OnFilePrintPreview()
{
   // capture our view
   CRect rect;
   GetWindowRect( &rect );
   m_bitmap.Capture( rect );

   CView::OnFilePrintPreview();
}
```

用ClassWizard 重载OnPrint()函数并在其中填充位图类的 Print()函数：

```
void CWzdView::OnPrint( CDC* pDC, CPrintInfo* pInfo )
{
   // print captured bitmap to pDC
   m_bitmap.Print( pDC );

   // CView::OnPrint( pDC, pInfo );
}
```

4. 注意

为什么用::StretchDIBits()而不是用CDC::StretchBlt()函数来扩展位图呢？为什么在打印之前需要将位图转变成DIB呢？为什么不能仅仅将屏幕调色板选入打印机设备环境，让设备环境出于真正的协同工作性而将其选出呢？答案是：从设备调色板转变为其他设备调色板的功能比起从已有的DIB格式开始转变，其开销将明显大很多。

5. 使用光盘时注意

执行随书附带的光盘中的工程时，单击 File，然后再单击Print或者Print Preview菜单命令。视(碰巧是空的)将打印到打印机或者打印预览对话框。

6. 程序清单 —— 位图类

```
#ifndef WZDBITMAP_H
#define WZDBITMAP_H

class CWzdBitmap : public CBitmap
{
public:
   DECLARE_DYNAMIC( CWzdBitmap )

// Constructors
```

```
    CWzdBitmap();

    void Capture( CRect &rect );
    CPalette *GetPalette(){return m_pPalette;};
    HANDLE CreateDIB( int *pbmData = NULL );
    void Print( CDC *pDC );

// Implementation
public:
    virtual ~CWzdBitmap();

// Attributes
    int m_nWidth;
    int m_nHeight;
// Operations

private:
    CPalette *m_pPalette;
};
#endif
// WzdBitmap.cpp : implementation of the CWzdBitmap class
//

#include "stdafx.h"
#include "WzdBtmap.h"

////////////////////////////////////////////////////////////////////////////
// CWzdBitmap

IMPLEMENT_DYNAMIC( CWzdBitmap, CBitmap )


CWzdBitmap::CWzdBitmap()
{
    m_pPalette = NULL;
}

CWzdBitmap::~CWzdBitmap()
{
    if ( m_pPalette )
    {
        delete m_pPalette;
    }
}

void CWzdBitmap::Capture(CRect &rect)
{
    // cleanup from last capture
    if (m_pPalette)
    {
        delete m_pPalette;
```

```
    DeleteObject();
}

// save width and height
m_nWidth = rect.Width();
m_nHeight = rect.Height();

////////////////////////////////////////
// copy screen image into a bitmap object
////////////////////////////////////////

// create a device context that accesses the whole screen
CDC dcScreen;
dcScreen.CreateDC( “DISPLAY”, NULL, NULL, NULL );

// create an empty bitmap in memory
CDC dcMem;
dcMem.CreateCompatibleDC( &dcScreen );
CreateCompatibleBitmap( &dcScreen, m_nWidth, m_nHeight );
dcMem.SelectObject( this );

// copy screen into empty bitmap
dcMem.BitBlt( 0,0,m_nWidth,m_nHeight,&dcScreen,rect.left,rect.top,SRCCOPY );

// this bitmap is worthless without the current system palette, so...

////////////////////////////////////////////
// save system palette in this bitmap’s palette
////////////////////////////////////////////

// create an empty logical palette that’s big enough to hold all the colors
int nColors = ( 1 << ( dcScreen.GetDeviceCaps( BITSPIXEL ) *
    dcScreen.GetDeviceCaps( PLANES ) ) );
LOGPALETTE *pLogPal = ( LOGPALETTE * )new BYTE[
    sizeof( LOGPALETTE ) + ( nColors * sizeof( PALETTEENTRY ) )];

// initialize this empty palette’s header
pLogPal -> palVersion    = 0x300;
pLogPal -> palNumEntries = nColors;

// load this empty palette with the system palette’s colors
::GetSystemPaletteEntries( dcScreen.m_hDC, 0, nColors,
    ( LPPALETTEENTRY )( pLogPal -> palPalEntry ) );

// create the palette with this logical palette
m_pPalette = new CPalette;
m_pPalette -> CreatePalette( pLogPal );

// clean up
delete []pLogPal;
dcMem.DeleteDC();
```

```
    dcScreen.DeleteDC();
}


HANDLE CWzdBitmap::CreateDIB( int *pbmData )
{
    ////////////////////////////////////////
    // create DIB header from our BITMAP header
    ////////////////////////////////////////

    BITMAPINFOHEADER bi;
    memset( &bi, 0, sizeof( bi ) );
    bi.biSize = sizeof( BITMAPINFOHEADER );
    bi.biPlanes = 1;
    bi.biCompression = BI_RGB;

    // get and store dimensions of bitmap
    BITMAP bm;
    GetObject( sizeof( bm ),( LPSTR )&bm );
    bi.biWidth = bm.bmWidth;
    bi.biHeight = bm.bmHeight;

    // get number of bits required per pixel
    int bits = bm.bmPlanes * bm.bmBitsPixel;
    if (bits <= 1)
        bi.biBitCount = 1;
    else if ( bits <= 4 )
        bi.biBitCount = 4;
    else if ( bits <= 8 )
        bi.biBitCount = 8;
    else
        bi.biBitCount = 24;

    // calculate color table size
    int biColorSize = 0;
    if ( bi.biBitCount! = 24 ) biColorSize = ( 1 << bi.biBitCount );
    biColorSize* = sizeof( RGBQUAD );

    // calculate picture data size
    bi.biSizeImage = ( DWORD )bm.bmWidth * bi.biBitCount;       // bits per row
    bi.biSizeImage = ( ( ( bi.biSizeImage ) + 31 ) / 32 ) * 4;   // DWORD aligned
    bi.biSizeImage* = bm.bmHeight;            // bytes required for whole bitmap

    // return size to caler in case they want to save to file
    if ( pbmData )
        *pbmData = bi.biSize + biColorSize;

    ////////////////////////////////////////
    // get DIB color table and picture data
    ////////////////////////////////////////

    // allocate a hunk of memory to hold header, color table and picture data
    HANDLE hDIB = ::GlobalAlloc( GHND, bi.biSize + biColorSize + bi.biSizeImage );
```

```
    // get a memory pointer to this hunk by locking it
    LPBITMAPINFOHEADER lpbi = ( LPBITMAPINFOHEADER )::GlobalLock( hDIB );

    // copy our header structure into hunk
    *lpbi = bi;

    // get a device context and select our bitmap's palette into it
    CDC dc;
    dc.Attach( ::GetDC( NULL ) );
    CPalette *pPal = dc.SelectPalette( m_pPalette,FALSE );
    dc.RealizePalette();

    // load our memory hunk with the color table and picture data
    ::GetDIBits( dc.m_hDC, ( HBITMAP )m_hObject, 0, ( UINT )bi.biHeight,
       ( LPSTR )lpbi + (WORD)lpbi->biSize + biColorSize, (LPBITMAPINFO)lpbi,
       DIB_RGB_COLORS);

    // clean up
    ::GlobalUnlock( hDIB );
    dc.SelectPalette( pPal,FALSE );
    dc.RealizePalette();

    // return handle to the DIB
    return hDIB;
}

void CWzdBitmap::Print( CDC *pDC )
{
    // get DIB version of bitmap
    int bmData;
    HANDLE hDIB = CreateDIB( &bmData );

    // get memory pointers to the DIB's header and data bits
    LPBITMAPINFOHEADER lpDIBHdr = ( LPBITMAPINFOHEADER )::GlobalLock( hDIB );
    LPSTR lpDIBBits = ( LPSTR )lpDIBHdr+bmData;

    // stretch bitmap to fill printed page with 1/4 inch borders
    int cxBorder = pDC -> GetDeviceCaps(LOGPIXELSX)/4;
    int cyBorder = pDC -> GetDeviceCaps(LOGPIXELSY)/4;
    int cxPage = pDC -> GetDeviceCaps(HORZRES) - (cxBorder*2);
    int cyPage = ( int )( ( ( double )cxPage/
       ( double )m_nWidth ) * ( double )m_nHeight );

    // stretch the bitmap for the best fit on the printed page
    pDC -> SetStretchBltMode( COLORONCOLOR );
    int i = ::StretchDIBits( pDC -> m_hDC,
       cxBorder,cyBorder,cxPage,cyPage,          // destination dimensions
       0,0,m_nWidth,m_nHeight,                   // source bitmap dimensions
                                                 //     (use all of bitmap)
       lpDIBBits,                                // bitmap picture data
       (LPBITMAPINFO)lpDIBHdr,                   // bitmap header info
```

```
        DIB_RGB_COLORS,             // specify color table
                                    //     has RGB values
        SRCCOPY                     // simple source to destination copy
    );

    // cleanup
    ::GlobalUnlock( hDIB );
    ::GlobalFree( hDIB );
    return;
}
```

6.5 实例19：绘制MDI客户视

1. 目标

如图6-4所示，在MDI应用程序的MDI客户视(MDI Client View)中绘制装饰图案。

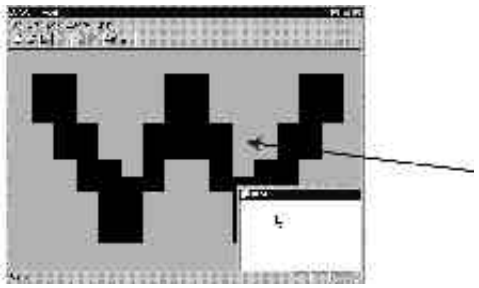

图6-4 填充MDI客户区

2. 策略

MDI应用程序由几个子框架窗口填充，这样的每个子框架窗口包含一个视类的窗口。这些子框架窗口覆盖的区域并不只是主框架窗口的客户区。实际上该区域是由 MDI Client窗口类创建的子窗口所覆盖。因此，为了绘制背景，不能简单地为 CMainframe类增加WM_PAINT消息处理函数并绘制背景。实际必须对 MDI Client子类化并截获WM_PAINT窗口消息。

3. 步骤

1) 创建新的MDI Client窗口类

用CLassWizard创建一个从generic CWnd派生的新类。

在该新类中嵌入一个CBitmap变量：

```
CBitmap  m_bitmap;
```

在新类的构造函数内为该成员变量装载要显示到 MDI Client区域的位图：

```
CMDIClientWnd::CMDIClientWnd()
{
    // get bitmap and info
    m_bitmap.LoadBitmap( IDB_WZD_MDIBITMAP );
    m_bitmap.GetObject( sizeof( BITMAP ), &m_bm );
}
```

用ClassWizard为该类添加 WM_PAINT消息处理函数。在该处理函数内为客户区绘制位图：

```
void CMDIClientWnd::OnPaint()
{
    CPaintDC dc( this );    // device context for painting
```

```
    CDC dcMem;
    dcMem.CreateCompatibleDC( &dc );
    dcMem.SelectObject( m_bitmap );

    // stretch bitmap
    CRect rect;
    GetClientRect(&rect);
    dc.StretchBlt( rect.left, rect.top, rect.right, rect.bottom,
      &dcMem, 0, 0, m_bm.bmWidth-1, m_bm.bmHeight-1, SRCCOPY );
```

本实例结尾的程序清单——MDI CLient窗口类提供以上MDI Client窗口类的完整列表。

现在已经拥有了一个新的MDI窗口类，需要将实际的MDI Client窗口子类化以便新类能够截获WM_PAINT消息。

2) 实现新的MDI Client窗口类

在CMainFrame中嵌入新的MDI Client窗口类：

```
public:
    CMDIClientWnd m_wndMDIClient;
```

然后在CMainFrame的OnCreate成员函数中用新的窗口类子类化当前的 MDI Client窗口：

```
int CMainFrame::OnCreate( LPCREATESTRUCT lpCreateStruct )
{
    :  :  :

    // subclass our CMDIClientWnd class to MDIClient window
    if( !m_wndMDIClient.SubclassWindow( m_hWndMDIClient ) )
    {
      TRACE( " Failed to subclass MDI client window\n" );
      return -1;
    }
```

主窗口被重置大小时，只有MDI Client窗口新区域无效——创建了一块扩展的位图。为了消除这种现象，应该使整个客户区在重置主窗口时都无效。

用ClassWizard添加一个WM_SIZE 消息处理函数并如下加入代码:

```
void CMainFrame::OnSize( UINT nType, int cx, int cy )
{
    CMDIFrameWnd::OnSize( nType, cx, cy );

    m_wndMDIClient.Invalidate();
}
```

4. 注意

本例在MDI Client窗口中打印一个大大的W。更漂亮点的位图可能是包含重复小图片的那种，例如按照行和列逐个显示图标等。为实现在适当位置堆砌图标当然要为 WM_PAINT处理函数添加更多的代码。

既然已经控制了MDI应用程序的客户区，在空间限制以内就可以添加控件统计列表等等。

5. 使用光盘时注意

执行随书附带光盘中的该工程，将注意到在 MDI Client区域中有一个大大的W。重置主窗

口将使得该W字随之扩大或者缩小。

6. 程序清单 —— MDI客户窗口类

```
#if !defined MDICLIENTWND_H
#define MDICLIENTWND_H

/////////////////////////////////////////////////////////////////////////////
// CMDIClientWnd window

class CMDIClientWnd : public CWnd
{
// Construction
public:
    CMDIClientWnd();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    // {{AFX_VIRTUAL( CMDIClientWnd )
    // }}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMDIClientWnd();

    // Generated message map functions
protected:
    // {{AFX_MSG( CMDIClientWnd )
    afx_msg void OnPaint();
    // }}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    BITMAP m_bm;
    CBitmap m_bitmap;
};

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

#endif
// MDIClientWnd.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "MDIClientWnd.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMDIClientWnd

CMDIClientWnd::CMDIClientWnd()
{
    // get bitmap and info
    m_bitmap.LoadBitmap( IDB_WZD_MDIBITMAP );
    m_bitmap.GetObject( sizeof( BITMAP ), &m_bm );
}

CMDIClientWnd::~CMDIClientWnd()
{
}

BEGIN_MESSAGE_MAP( CMDIClientWnd, CWnd )
    // {{AFX_MSG_MAP( CMDIClientWnd )
    ON_WM_PAINT()
    // }}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMDIClientWnd message handlers

void CMDIClientWnd::OnPaint()
{
    CPaintDC dc( this );    // device context for painting

    CDC dcMem;
    dcMem.CreateCompatibleDC( &dc );
    dcMem.SelectObject( m_bitmap );

    // stretch bitmap
    CRect rect;
    GetClientRect( &rect );
    dc.StretchBlt( rect.left, rect.top, rect.right, rect.bottom,
        &dcMem, 0, 0, m_bm.bmWidth-1, m_bm.bmHeight-1, SRCCOPY );
}
```

6.6 实例20：拖放文件到视

1. 目标

打开拖放到视中的文件。

2. 策略

众所周知，可以从Windows 资源管理器或者其他工具软件中拖动一个文件放到 Developer Studio中，Studio就可以魔术般地打开该文件。为了在自己的应用程序中实现同样的功能，则需要首先用CWnd::DragAcceptFiles()函数通知系统应用程序窗口中可以支持文件拖放。然后

为CMainFrame类增加一个WM_DROPFILES消息处理函数，以获取所拖放的文件的名字并打开相应的文件。在本实例中，将修改一个 MDI应用程序来为每个拖放的文件打开新文档和视。

3. 步骤

修改主框架类

从CMainFrame的OnCreate()消息处理函数内调用下列函数，将使应用程序能够接受拖放的文件：

```
// allow files to be dropped on main window or any child window
DragAcceptFiles();
```

用ClassWizard为CMainFrame类添加WM_DROPFILES消息处理函数，使得可以使用CWinApp::OpenDocumentFile()函数为该文件创建新的文档/视：

```
void CMainFrame::OnDropFiles( HDROP hDropInfo )
{
   // get filename stored in hDropInfo and use app to open it
   TCHAR szFileName[_MAX_PATH];
   ::DragQueryFile( hDropInfo, 0, szFileName, _MAX_PATH );
   ::DragFinish( hDropInfo );
   AfxGetApp() -> OpenDocumentFile( szFileName );

   CMDIFrameWnd::OnDropFiles( hDropInfo );
}
```

4. 注意

该消息处理函数在 SDI应用程序中也可以使用，但在打开新文件之前将破坏当前的文档 /视。

5. 使用光盘时注意

执行随书附带的光盘中的该工程时，打开 Windows 资源浏览器，从Windows Explorer拖动一个文件到该应用程序中，注意一个新的文档 /视被打开了。