

```

/*****
/*          I M A G E D I T . C          */
/*-----*/
/* Task      : Demonstration of the use of ImageLists          */
/*-----*/
/* Authors    : Michael Tischer and Bruno Jennrich          */
/* developed on : 10/15/1995          */
/* last update  : 10/18/1995          */
/*****
#include <windows.h>
#include <windowsx.h>          // Message crackers and macros!!
#include <commctrl.h>

#include "resource.h"

#include "paintpic.h"

//----- Global Variables -----
HIMAGELIST g_hImageList = 0;          // ImageList to be edited
HIMAGELIST g_hDragCursors = 0;          // ImageList with drag cursor

FARPROC g_lpOldDlgProc;          // Address of original dialog procedure
static HWND g_hListBox;          // Handle of ListBox with images

UINT WM_DRAGLISTMSGSTRING;          // registered message for drag listboxes

//----- Constants -----
#define PIC_WIDTH 32          // Width of an image
#define PIC_HEIGHT 32          // Height of an image

#define HS_CURX 12          // Custom cursor hotspots
#define HS_CURY 11

/*****
/* GetDragCursors - Make bitmap with cursor shapes available as          */
/*          ImageList.          */
/*-----*/
/* Parameter : hInstance - Instance under which the bitmap          */
/*          IDB_DRAGCURSORS is to be found          */
/* Return value: ImageList with drag cursor          */
/*****
HIMAGELIST GetDragCursors( HINSTANCE hInst )
{
    // Make bitmap available as ImageList -----
    return ImageList_LoadBitmap( hInst,
                                MAKEINTRESOURCE( IDB_DRAGCURSORS ),
                                32,          // Cursor is 32 pixels wide
                                0,          // ImageList not incremented
                                RGB( 0,128,128 ) ); // Green as mask color
}

/*****
/* ReadDIB - Reads DIB or BMP file and makes it available in the          */
/*          form of an HBITMAP          */
/*-----*/
/* Parameter : lpszFileName - Filename of DIB or BMP          */
/* Return value: Bitmap handle          */
/*****
HBITMAP ReadDIB(LPSTR lpszFileName )
{
    LPBITMAPFILEHEADER    lpBFH;          // Bitmap file header
    LPBITMAPINFOHEADER    lpBIH;          // Bitmap info header
    WORD                  nNumColors;          // Number of color items
    LPRGBQUAD             lpColors;          // Address of color table
    LPBYTE                 lpBits;          // Address of pixel data

    HANDLE                hFile;          // File handle
    HBITMAP                hBM = 0;          // Bitmap handle
    HDC                    hDC;          // Device context for DIB -> DDB

    hFile = CreateFile( lpszFileName,          // Open BMP or DIB file
                        GENERIC_READ,
                        FILE_SHARE_READ,
                        NULL,
                        OPEN_EXISTING,
                        0,

```

```

        NULL );

if( hFile != INVALID_HANDLE_VALUE )                // Is everything OK?
{
    HANDLE hFileMapping;                          // Create file mapping through file
    hFileMapping = CreateFileMapping( hFile,
                                      NULL,
                                      PAGE_READONLY | SEC_COMMIT,
                                      0,
                                      0,
                                      NULL );
    if( hFileMapping != INVALID_HANDLE_VALUE )// File mapping handle OK?
    {
        LPVOID lpBM;                             // Start address of file map

        lpBM = MapViewOfFile( hFileMapping,        // Map file into memory
                              FILE_MAP_READ,
                              0,
                              0,
                              0 );

        if( lpBM )
        {
            lpBFH = (LPBITMAPFILEHEADER)lpBM;
            if( lpBFH->bfType == 0x4d42 )          // 'BM' found?
            {
                // Get address of bitmap info header -----
                lpBIH = (LPBITMAPINFOHEADER)( (DWORD)lpBM +
                                                sizeof( BITMAPFILEHEADER ) );

                // Correct header size? -----
                if (lpBIH->biSize != sizeof(BITMAPCOREHEADER))
                {
                    // Get number of colors -----
                    nNumColors = (WORD)lpBIH->biClrUsed;

                    // In case NumColors == 0 and no 24 bit bitmap. -
                    // Calculate number of colors from bits per pixel -
                    if( ( !nNumColors ) && ( lpBIH->biBitCount != 24 ) )
                        nNumColors = 1 << lpBIH->biBitCount;

                    // Address of color table -----
                    lpColors = (LPRGBQUAD) (( DWORD )lpBIH + lpBIH->biSize);

                    // Calculate address of image information -----
                    if (lpBFH->bfOffBits != 0L)
                    {
                        // Offset has been specified in file -----
                        lpBits = (LPBYTE)((DWORD)lpBM + lpBFH->bfOffBits);
                    }
                    else
                    {
                        // Offset must be calculated - image information located -
                        // behind the color table -
                        lpBits = (LPBYTE)( (DWORD)lpBIH +
                                           (WORD)lpBIH->biSize +
                                           nNumColors * sizeof( RGBQUAD ) );
                    }

                    // Create screen DC -----
                    hDC = GetDC( GetDesktopWindow() );

                    // Convert Device Independent Bitmap to Screen Bitmap -----
                    hBM = CreatedDIBitmap( hDC,
                                           lpBIH,
                                           CBM_INIT,
                                           (LPVOID)lpBits,
                                           (LPBITMAPINFO)lpBIH,
                                           DIB_RGB_COLORS );

                    ReleaseDC( GetDesktopWindow(), hDC );          // Release DC
                }
            }
            UnmapViewOfFile( lpBM );                    // Release memory map
        }
        CloseHandle( hFileMapping );                  // Release mapping handle
    }
}

```

```

    }
    CloseHandle( hFile ); // Close file
}
return hBM; // Handle of loaded bitmap
}

/*****
/* StretchBitmap - Create new bitmap of desired size from
/* given bitmap
/*-----
/* Parameter : hBM - Handle of original
/* newx - desired width
/* newy - desired height
/* Return value: Handle of resized bitmap
*****/
HBITMAP StretchBitmap( HBITMAP hBM, int newx, int newy )
{
    HDC hdcSRC, hdcDST; // DC for selection of bitmap
    HBITMAP hbmDST, hbmOLD1, hbmOLD2; // Temporary bitmap handles
    BITMAP bm; // Bitmap structure for obtaining bitmap info

    // Gather information (height,width) about original -----
    GetObject( hBM, sizeof( bm ), &bm );

    hdcSRC = CreateCompatibleDC( NULL ); // Create screen source DC
    hbmOLD1 = SelectObject( hdcSRC, hBM ); // Select bitmap in hdcSRC

    hdcDST = CreateCompatibleDC( NULL ); // Create destination DC
    hbmDST = CreateBitmap( newx, // Create bitmap of required
                          newy, // dimensions
                          GetDeviceCaps( hdcDST, PLANES ),
                          GetDeviceCaps( hdcDST, BITSPIXEL ),
                          NULL );
    hbmOLD2 = SelectObject( hdcDST, hbmDST ); // Select new bitmap

    // Copy original bitmap, resizing it (Stretch) -----
    StretchBlt( hdcDST, 0, 0, newx, newy,
               hdcSRC, 0, 0, bm.bmWidth, bm.bmHeight, SRCCOPY );

    SelectObject( hdcDST, hbmOLD2 ); // Reselect old bitmaps
    SelectObject( hdcSRC, hbmOLD1 );
    DeleteDC( hdcDST ); // Destroy DCs
    DeleteDC( hdcSRC );

    return hbmDST; // Return resized (stretched) bitmap
}

/*****
/* IL_GetBitmap - Isolate individual image of an ImageList in the
/* form of a bitmap.
/*-----
/* Parameter : hIL - Handle of ImageList
/* iIndex - Index of image to be made available as
/* a bitmap.
/* Return value: Handle of bitmap
*****/
HBITMAP IL_GetBitmap( HIMAGELIST hIL, int iIndex )
{
    HDC hdcDST; // DC for GDI operations
    HBITMAP hbmDST, hbmOLD; // Temporary bitmap handles
    int cx, cy; // Image size

    // Get size of images in the passed ImageList -----
    ImageList_GetIconSize( hIL, &cx, &cy );

    hdcDST = CreateCompatibleDC( NULL ); // Create screen DC
    hbmDST = CreateBitmap( cx, // Create bitmap in image size
                          cy,
                          GetDeviceCaps( hdcDST, PLANES ),
                          GetDeviceCaps( hdcDST, BITSPIXEL ),
                          NULL );
    hbmOLD = SelectObject( hdcDST, hbmDST ); // Select bitmap

    ImageList_Draw( hIL, // Draw image in bitmap
                   iIndex,
                   hdcDST,

```

```

        0,
        0,
        ILD_NORMAL );

SelectObject( hdcDST, hbmOLD );           // Reselect old DC bitmap
DeleteDC( hdcDST );                       // Destroy DC

return hbmDST;                           // Return bitmap
}

/*****
/* SaveImageList - Save ImageList in OLE stream */
/*-----*/
/* Parameter :    hIL        - Handle of ImageList to be saved */
/*               lpszStorage - Name of OLE storage */
/*               lpszStream  - Name of OLE stream */
/* Return value: TRUE - Save OK, FALSE - Error */
*****/
BOOL SaveImageList( HIMAGELIST hIL,
                    LPSTR      lpszStorage,
                    LPSTR      lpszStream )
{
    LPSTORAGE pStorage;           // Storage interface pointer
    LPSTREAM  pStream;           // Stream interface pointer
    HRESULT  hrCoInit, hr;       // HRESULTS
    BOOL  bRetVal;               // Temporary variable for return value
    WORD  wsz[ MAX_PATH ];       // ANSI -> WideChar Puffer

    hrCoInit = CoInitialize( NULL );           // Initialize OLE

    bRetVal = FALSE;                           // Assume worst case

    // ANSI to WideChar conversion -----
    MultiByteToWideChar( CP_ACP, 0, lpszStorage, -1, wsz, sizeof( wsz ) );

    hr = StgCreateDocfile( wsz,                       // Create storage
                          STGM_DIRECT|
                          STGM_READWRITE|
                          STGM_CREATE|
                          STGM_SHARE_EXCLUSIVE,
                          0,
                          &pStorage );

    if( SUCCEEDED( hr ) )
    {
        // ANSI to WideChar conversion -----
        MultiByteToWideChar( CP_ACP, 0, lpszStream, -1, wsz, sizeof( wsz ) );

        // Create stream -----
        hr = pStorage->lpVtbl->CreateStream( pStorage,
                                             wsz,
                                             STGM_DIRECT|
                                             STGM_READWRITE|
                                             STGM_CREATE|
                                             STGM_SHARE_EXCLUSIVE,
                                             0,
                                             0,
                                             &pStream );

        if( SUCCEEDED( hr ) )
        {
            // Save ImageList -----
            bRetVal = ImageList_Write( hIL, pStream );
            pStream->lpVtbl->Release( pStream );           // Release interface
        }
        pStorage->lpVtbl->Release( pStorage );           // Release interface
    }

    if( SUCCEEDED( hrCoInit ) ) CoUninitialize();       // Uninstall OLE

    return bRetVal;                                     // Return success message
}

/*****
/* LoadImageList - Read ImageList from OLE stream */
/*-----*/
/* Parameter :    lpszStorage - Name of OLE storage */
/*               lpszStream  - Name of OLE stream */
*****/

```

```

/* Return value: Handle of ImageList (0=Error) */
/*****
HIMAGELIST LoadImageList( LPSTR lpszStorage, LPSTR lpszStream )
{
    LPSTORAGE pStorage;           // Storage interface pointer
    LPSTREAM pStream;             // Stream interface pointer
    HRESULT hrCoInit, hr;         // HRESULTS
    WORD wsz[ MAX_PATH ];         // ANSI -> WideChar Puffer
    HIMAGELIST hIL;               // The read ImageList

    hrCoInit = CoInitialize( NULL );           // Initialize OLE

    hIL = NULL;                               // Worst case assumption

    // ANSI to WideChar conversion -----
    MultiByteToWideChar( CP_ACP, 0, lpszStorage, -1, wsz, sizeof( wsz ) );

    hr = StgOpenStorage( wsz,                // Open OLE storage
        NULL,
        STGM_DIRECT|
        STGM_READWRITE|
        STGM_SHARE_EXCLUSIVE,
        NULL,
        0,
        &pStorage );

    if( SUCCEEDED( hr ) )
    {
        // ANSI to WideChar conversion -----
        MultiByteToWideChar( CP_ACP, 0, lpszStream, -1, wsz, sizeof( wsz ) );

        hr = pStorage->lpVtbl->OpenStream( pStorage,        // Open OLE stream
            wsz,
            NULL,
            STGM_DIRECT|
            STGM_READWRITE|
            STGM_SHARE_EXCLUSIVE,
            0,
            &pStream );

        if( SUCCEEDED( hr ) )
        {
            hIL = ImageList_Read( pStream );               // Read ImageList
            pStream->lpVtbl->Release( pStream );            // Release interface
        }
        pStorage->lpVtbl->Release( pStorage );              // Release interface
    }

    if( SUCCEEDED( hrCoInit ) ) CoUninitialize();         // Uninstall OLE

    return hIL;                                           // Return the read ImageList
}

/*****
/* DlgSubclass - Subclass procedure for dialog */
/*-----
/* Parameter :    default parameters */
/* Return value: default return value */
/*-----
/* Note:         Subclassing the dialog is necessary because the use */
/*               of ListBox drag API requires genuine return values. */
/*               You cannot achieve this through the default DlgProc */
/*               procedure of a dialog, since only TRUE or FALSE can */
/*               be returned there. This return value only informs */
/*               Windows as to whether or not a dialog message from */
/*               Windows needs to be processed. */
/*****
LRESULT WINAPI DlgSubclass( HWND hWnd, UINT wParam, WPARAM wp, LPARAM lp )
{
    static int iDragItem;           // Index ListBox item to be dragged
    static int iDropTarget;         // Index of DropTarget item
    static POINT ptHotspot;

    if( wParam == WM_DRAGLISTMSGSTRING )           // ListBox drag message?
    {
        LPDRAGLISTINFO pDLI = ( LPDRAGLISTINFO )lp;

```

```

switch( pDLI->uNotification )           // Which type of drag message?
{
    case DL_BEGINDRAG:                   // Begin drag operation
    {
        POINT pt;
        int iImage;
        RECT r;

        // Get item under Cursor -----
        iDragItem = LBItemFromPt(g_hListBox, pDLI->ptCursor, FALSE );
        // Get index of item image -----
        iImage = ListBox_GetItemData( g_hListBox, iDragItem );

        // Get rectangle of clicked item
        ListBox_GetItemRect( g_hListBox, iDragItem, &r );

        // Client coordinates of rectangle to screen coordinates
        pt.x = r.left;
        pt.y = r.top;
        ClientToScreen( g_hListBox, &pt );

        // Build rectangle of actual image from screen coordinates.
        // The drag operation will be launched only when the image
        // itself has been clicked.
        r.left = pt.x;
        r.top = pt.y;
        r.right = pt.x + PIC_WIDTH;
        r.bottom = pt.y + PIC_HEIGHT;

        if( !PtInRect( &r, pDLI->ptCursor ) ) return FALSE;    // Cancel

        // Draw transparent DragImage -----
        ImageList_SetBkColor(g_hImageList, CLR_NONE );

        // Calculate Click Position within the image (Hotspot)
        ptHotspot.x = pDLI->ptCursor.x - pt.x;
        ptHotspot.y = pDLI->ptCursor.y - pt.y;

        if( !ImageList_BeginDrag( g_hImageList,           // Start dragging
                                   iImage,
                                   ptHotspot.x,
                                   ptHotspot.y ) ) return FALSE;

        // Manage custom cursor -----
        ImageList_SetDragCursorImage( g_hDragCursors,
                                       0,
                                       HS_CURX - ptHotspot.x,
                                       HS_CURY - ptHotspot.y );

        ImageList_DragEnter( NULL,           // Drag across entire screen
                             pDLI->ptCursor.x,
                             pDLI->ptCursor.y );

        // Disable mouse, since CursorImage will be displayed -----
        // within the DragImage -----
        ShowCursor( FALSE );

        // Trick: Since BL_DRAGGING messages aren't sent until -----
        // the mouse has been moved away from the DragItem, a -----
        // MOUSEMOVE fake is sent here, causing the Listbox to -----
        // think that the mouse has been moved from the current item. -----
        PostMessage( g_hListBox,WM_MOUSEMOVE,0,MAKELONG(-32768,-32768));
        return TRUE;                                // Launch drag operation
    }
    case DL_DRAGGING:                       // Drag messages
    {
        RECT rDropRemove;

        // Let DragImage follow the cursor -----
        ImageList_SetBkColor( g_hImageList, CLR_NONE );
        ImageList_DragMove( pDLI->ptCursor.x, pDLI->ptCursor.y );

        // Get rectangle of Recycle Bin.
        GetWindowRect( GetDlgItem( hWnd, IDC_DROPREMOVEWND ),
                       &rDropRemove );
    }
}

```

```

// If the Recycle Bin wasn't hit, the NO cursor is -
// displayed. Otherwise the default cursor is used, -
// indicating that a drop onto the current mouse position -
// is allowed.
if( PtInRect( &rDropRemove, pDLI->ptCursor ) )
    ImageList_SetDragCursorImage( g_hDragCursors,
                                0,
                                ptHotspot.x - HS_CURX ,
                                ptHotspot.y - HS_CURY );

else
    ImageList_SetDragCursorImage( g_hDragCursors,
                                1,
                                ptHotspot.x - HS_CURX ,
                                ptHotspot.y - HS_CURY );
}
return 0;
case DL_DROPPED:                                // Drop!
{
    RECT rDropRemove;

    ImageList_DragLeave( NULL );                    // End drag
    ImageList_EndDrag();
    ShowCursor( TRUE );                            // Display cursor again

    // Was mouse button released over Recycle Bin? -----
    GetWindowRect( GetDlgItem( hWnd, IDC_DROPREMOVEWND ),
                  &rDropRemove );
    if( PtInRect( &rDropRemove, pDLI->ptCursor ) )
    {
        int iImage, iItem;

        // Get current item and the image displayed in it -----
        iItem = ListBox_GetCurSel( g_hListBox );
        iImage = ListBox_GetItemData( g_hListBox, iItem );

        // Remove image from ImageList -----
        if( ImageList_Remove( g_hImageList, iImage ) )
        {
            int iCnt;
            // Delete item in Listbox -----
            iCnt = ListBox_DeleteString( g_hListBox, iItem );

            // Adapt image indexes of the following items -----
            for( ;iItem < iCnt; iItem++ )
            {
                iImage = ListBox_GetItemData( g_hListBox, iItem );
                iImage --;
                ListBox_SetItemData( g_hListBox, iItem, iImage );
            }
        }
    }
}
break;
case DL_CANCELDRAG:    // Cancel drag (ESC or right mouse button)
    ImageList_DragLeave( NULL );
    ImageList_EndDrag();
    ShowCursor( TRUE );                            // Display cursor again
break;
}
}
// call original dialog procedure -----
return CallWindowProc( g_lpOldDlgProc, hWnd, wParam, lParam );
}

/*****
/* DlgProc - Dialog function */
/*-----*/
/* Parameter : default parameters */
/* Return value: default return value */
/*****/
BOOL WINAPI DlgProc( HWND hWnd, UINT wParam, WPARAM wp, LPARAM lp )
{
    switch( wParam )
    {
        case WM_INITDIALOG:
        {

```

```

// Write handle of ListBox to global variable -----
g_hListBox = GetDlgItem( hWnd, IDC_PICTURES );

// Load existing ImageList if necessary -----
g_hImageList = LoadImageList( "IMAGE.DAT", "IMAGELIST" );

if( g_hImageList )
{ int i; // Create item in ListBox for each image -----
  for( i = 0; i < ImageList_GetImageCount(g_hImageList); i++ )
    SendMessage( g_hListBox, LB_ADDSTRING, 0, i );
}
else
  // Create empty ImageList -----
  g_hImageList = ImageList_Create( PIC_WIDTH,
                                   PIC_HEIGHT,
                                   ILC_COLORDB |
                                   ILC_MASK,
                                   10, // Room for 10 images
                                   10 ); // 10 images increase

// Make Listbox into Drag-Aware Listbox -----
MakeDragList( g_hListBox );

// Files can be dragged to dialog box -----
DragAcceptFiles( hWnd, TRUE );

// Subclass dialog box in order to process drag messages -----
// properly -----
g_lpOldDlgProc = (FARPROC)SetWindowLong( hWnd,
                                           GWL_WNDPROC,
                                           (LONG)DlgSubclass );
}
break;
case WM_DROPFILES: // Somebody dropped a couple of files
{
  int iNumFiles, i; // Number of files
  int iFirstItem = -1; // first new item
  HDROP hDrop = (HDROP)wp; // Drop handle

  // Get number of dropped files -----
  iNumFiles = DragQueryFile(hDrop, (UINT)-1, NULL, 0 );

  for( i = 0; i < iNumFiles; i++ ) // browse all drop files
  {
    char szFileName[ MAX_PATH ]; // Buffer for file names
    HBITMAP hBM; // Handle of loaded bitmap

    // Get filename -----
    DragQueryFile(hDrop, i, szFileName, MAX_PATH );

    // Assumption: file is a BMP or DIB file -----
    hBM = ReadDIB( szFileName );

    if( hBM == NULL ) // it was not a BMP or DIB file
    {
      MessageBox( hWnd,
                  "Error loading the file. Not a DIB or BMP file!",
                  szFileName,
                  0 );
    }
    else
    {
      HBITMAP hBMS; // Handle of resized bitmap
      int iImage, iItem;

      // Adapt loaded bitmap to size of images in ImageList -
      hBMS = StretchBitmap( hBM, PIC_WIDTH, PIC_HEIGHT );

      DeleteObject( hBM ); // Delete original

      // Add bitmap to ImageList -----
      iImage = ImageList_AddMasked(g_hImageList,
                                   hBMS,
                                   RGB(255,255,255));
      DeleteObject( hBMS ); // Bitmap no longer required

      // Create item for image in ListBox

```

```

        iItem = SendMessage( g_hListBox, LB_ADDSTRING, 0, iImage );

        // Note first new item -----
        if( iFirstItem == -1 ) iFirstItem = iItem;
    }
}
// Select first new entry -----
ListBox_SetTopIndex( g_hListBox, iFirstItem );
ListBox_SetCurSel( g_hListBox, iFirstItem );
}
break;
case WM_MEASUREITEM:                                // Get size of a ListBox item
{
    LPMEASUREITEMSTRUCT lpMIS = (LPMEASUREITEMSTRUCT) lp;
    if( lpMIS->CtlType == ODT_LISTBOX )                // Do we have a ListBox?
        if( lpMIS->CtlID == IDC_PICTURES )            // Which one?
        {
            lpMIS->itemWidth  = PIC_WIDTH;            // Height and width match the
            lpMIS->itemHeight = PIC_HEIGHT;           // the height and width of an
            return TRUE;                               // image
        }
}
break;
case WM_DRAWITEM:                                    // Display ListBox item
{
    LPDRAWITEMSTRUCT lpDIS = (LPDRAWITEMSTRUCT) lp;
    if( lpDIS->CtlType == ODT_LISTBOX )                // Do we have a ListBox?
        if( lpDIS->CtlID == IDC_PICTURES )            // Which one?
        {
            ImageList_SetBkColor( g_hImageList, RGB(255,255,255));

            // Paint image from ImageList at requested location -----
            ImageList_Draw( g_hImageList,
                lpDIS->itemData,
                lpDIS->hDC,
                lpDIS->rcItem.left,
                lpDIS->rcItem.top,
                ( lpDIS->itemState & ODS_FOCUS ) ?
                ILD_FOCUS : ILD_NORMAL );

            return TRUE;
        }
}
break;
case WM_COMMAND:                                    // Somebody's playing around with the buttons
    switch( LOWORD( wp ) )
    {
        case IDC_ADDSYSIMGLIST:                        // Add images to system ImageList
        {
            HIMAGELIST hSysImages;
            SHFILEINFO sfi;
            int iNumImgs, i;
            int iFirstItem = -1;

            // Get handle of system ImageList -----
            hSysImages = (HIMAGELIST)SHGetFileInfo( "",
                0,
                &sfi,
                sizeof( sfi ),
                SHGFI_SYSICONINDEX );

            // Get number of images -----
            iNumImgs = ImageList_GetImageCount( hSysImages );

            // Add each image separately -----
            for( i = 0; i < iNumImgs; i++ )
            {
                HBITMAP hBM, hBMS;

                // First make a bitmap out of an image -----
                hBM = IL_GetBitmap( hSysImages, i );

                // Transfer image to required size -----
                hBMS = StretchBitmap( hBM, PIC_WIDTH, PIC_HEIGHT );

                DeleteObject( hBM );                    // Delete original
            }
        }
    }
}

```

```

if( hBMS == NULL )                                // Error?
{
    MessageBox( hWnd,
                "Error creating a system image",
                "Error",
                0 );
    i = iNumImgs;                                // End loop
}
else
{
    int iImage, iItem;

    // Add image to custom ImageList -----
    iImage = ImageList_AddMasked( g_hImageList,
                                   hBMS,
                                   RGB(255,255,255));
    DeleteObject( hBMS );                          // Delete image

    // Create item for new image in ListBox -----
    iItem = SendMessage( g_hListBox, LB_ADDSTRING, 0, iImage);

    // and note first new item -----
    if( iFirstItem == -1 ) iFirstItem = iItem;
}
}
// select first new entry -----
ListBox_SetTopIndex( g_hListBox, iFirstItem );
ListBox_SetCurSel( g_hListBox, iFirstItem );
}
break;
case IDC_PICTURES:                                // Evaluate messages of ListBox
    switch( HIWORD( wp ) )
    {
        case LBN_SELCHANGE:                      // Selection of a new item
        {
            int iItem, iImage;
            RECT rListBox;                        // Window-Rect of ListBox
            RECT rMe;                             // Window-Rect of dialog
            HDC hDC;

            // Which item was selected? -----
            iItem = ListBox_GetCurSel( ( HWND )lp );
            // and which image is "associated" with the item? -----
            iImage = ListBox_GetItemData( (HWND)lp, iItem );

            // Display image in different forms underneath the
            // Listbox
            -

            GetWindowRect( g_hListBox, &rListBox ); // Rectangle of LB
            GetWindowRect( hWnd, &rMe );           // Rectangle of dialog

            // Move ListBox rectangle so that the coordinates of the
            // ListBox are in Client coordinates of the dialog box
            OffsetRect( &rListBox, -rMe.left, -rMe.top );

            hDC = GetDC( hWnd );                   // Get dialog DC
            ImageList_SetBkColor(g_hImageList, RGB( 255, 255, 255 ) );

            // normal image -----
            ImageList_Draw( g_hImageList,
                            iImage,
                            hDC,
                            rListBox.left,
                            rListBox.bottom + 10,
                            ILD_NORMAL );

            // BLEND25 -----
            ImageList_Draw( g_hImageList,
                            iImage,
                            hDC,
                            rListBox.left + PIC_WIDTH,
                            rListBox.bottom + 10,
                            ILD_BLEND25 );

            // BLEND50 -----

```

```

        ImageList_Draw( g_hImageList,
                        iImage,
                        hDC,
                        rListBox.left + PIC_WIDTH * 2,
                        rListBox.bottom + 10,
                        ILD_BLEND50 );

        // MASK -----
        ImageList_Draw( g_hImageList,
                        iImage,
                        hDC,
                        rListBox.left + PIC_WIDTH * 3,
                        rListBox.bottom + 10,
                        ILD_MASK );

        ReleaseDC( hWnd, hDC ); // Release DC
    }
    break;
}
break;
case IDCANCEL: // Close button of dialog box
    EndDialog( hWnd, 0 ); // End dialog
    break;
}
break;
case WM_PAINT:
{
    PaintPicture( (HINSTANCE)GetWindowLong( hWnd, GWL_HINSTANCE ),
                  hWnd,
                  IDB_PCINTERN,
                  GetDlgItem( hWnd, IDC_PCIINTERN ) );
}
break;
case WM_DESTROY:
    if( g_hImageList )
    {
        // Save ImageList prior to exiting -----
        SaveImageList( g_hImageList, "IMAGE.DAT", "IMAGELIST" );
        ImageList_Destroy( g_hImageList ); // and then destroy!!
    }
    break;
}
return FALSE;
}

/*****
/* WinMain - Start function */
/*-----*/
/* Parameter : default parameters */
/* Return value: default return value */
/*****/
int WINAPI WinMain( HINSTANCE hInst,
                   HINSTANCE hPrev,
                   LPSTR lpszCmdLine,
                   int nCmdShow )
{
    // Get number of registered message for drag ListBoxes -----
    WM_DRAGLISTMSGSTRING = RegisterWindowMessage( DRAGLISTMSGSTRING );

    InitCommonControls(); // Initialize CommonControls

    g_hDragCursors = GetDragCursors( hInst ); // Load drag cursor

    DialogBox( hInst, // Call dialog box
              MAKEINTRESOURCE( IDD_DIALOG ),
              NULL,
              DlgProc );

    ImageList_Destroy( g_hDragCursors ); // Destroy drag cursor

    return 0;
}

```