

```

/*****
/*          T V D E M O . C          */
/*-----*/
/* Task      : TreeView Editor      */
/*-----*/
/* Authors    : Michael Tischer and Bruno Jennrich      */
/* developed on : 10/18/1995      */
/* last update  : 10/18/1995      */
/*****
#include <windows.h>
#include <commctrl.h>
#include <shellapi.h>
#include <ole2.h>

#include "resource.h"

#include "paintpic.h"

//----- Global Variables -----
HIMAGELIST g_hImagesNormal,           // normal images
            g_hImagesState;           // state images
HIMAGELIST g_hDragCursors = 0;        // ImageList with drag cursor

FARPROC g_lpOldEditProc;              // Address of old Edit Control procedure

//----- Typedefs -----
typedef union tagIMGCOMPOSITE          // for accessing a TreeItem's lParam
{
    struct tagc
    {
        BYTE bImage,                // Index of image for normal item
            bSelectedImage,         // Index of image for selected item
            bExpandedImage,         // Index of image for expanded item
            bExpandedSelectedImage; // expanded, selected item
    } c;
    LPARAM lParam;                  // lParam of TreeItem
} IMGCOMPOSITE;

typedef struct tagSAVETREE             // for saving and loading a tree item
{
    int iLen;                        // number of bytes actually used
    int iLevel;                      // level of item
    TV_ITEM tvi;                     // data of item
    char szText[ MAX_PATH ];         // text of item
} SAVETREE;

//----- Macros for UpDown controls -----
#define UpDown_DlgSetRange(h,id,min,max) SendDlgItemMessage(h,id,\
    UDM_SETRANGE,0,(LPARAM)MAKELONG(max,min))
#define UpDown_DlgGetPos(h,id) SendDlgItemMessage(h,id,UDM_GETPOS,0,0)
#define UpDown_DlgSetPos(h,id,p) SendDlgItemMessage(h,id,UDM_SETPOS,0,\
    (LPARAM)MAKELONG(p,0))

#define CRSR_HOTX 12                  // Hotspot of custom cursor
#define CRSR_HOTY 11

/*****
/* GetDragCursors - Make bitmap with cursor shapes available as      */
/*                  ImageList.                                         */
/*-----*/
/* Parameter : hInstance - Instance under which the IDB_DRAGCURSORS  */
/*                  bitmap 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 is not incremented
        RGB( 0,128,128 )); // Green as mask color
}

/*****
/* InsertTreeItem - Insertion of a new TreeView item      */

```

```

/*-----*/
/* Parameter : hTreeView      - Handle of TreeView controls      */
/*             hParent        - Handle of item under which a new  */
/*                             item is to be created              */
/*             pszText        - Text of new item                  */
/*             iImage         - Image index of a normal item      */
/*             iSelectedImage - Image index of a selected item    */
/*             iOverlayImage  - Index of overlay image (0-3)      */
/*             iStateImage    - Index of state image              */
/*             lParam         - User information                  */
/* Return value: Handle of new item                               */
/*****
HTREEITEM InsertTreeItem( HWND      hTreeView,
                        HTREEITEM hParent,
                        LPSTR      pszText,
                        int        iImage,
                        int        iSelectedImage,
                        int        iOverlayImage,
                        int        iStateImage,
                        LPARAM      lParam )
{
    TV_INSERTSTRUCT tvis;

    tvis.hParent      = hParent;                // Set parent item
    tvis.hInsertAfter = TVI_LAST;               // Insert new item at end

    tvis.item.mask = TVIF_TEXT |                // set flags for new item
                    TVIF_PARAM |
                    TVIF_IMAGE |
                    TVIF_STATE |
                    TVIF_SELECTEDIMAGE;

    tvis.item.pszText      = pszText;            // Set text
    tvis.item.cchTextMax   = ( pszText != LPSTR_TEXTCALLBACK ) ?
                            strlen( pszText ) : 0;
    tvis.item.iImage       = iImage;
    tvis.item.iSelectedImage = iSelectedImage;

    // item.state contains state image and overlay image -----
    tvis.item.stateMask    = TVIS_OVERLAYMASK |
                            TVIS_STATEIMAGEMASK;
    tvis.item.state        = INDEXTOOVERLAYMASK(iOverlayImage) |
                            INDEXTOSTATEIMAGEMASK(iStateImage);
    tvis.item.lParam = lParam;                   // User data

    return TreeView_InsertItem( hTreeView, &tvis );    // Insert item
}

/*****
/* SaveTreeEngine - Workhorse for recursive saving of a tree      */
/*-----*/
/* Parameter : pStream      - OLE stream in which tree is to be   */
/*                             saved.                               */
/*             hTreeView    - Handle of TreeView Controls          */
/*             hParent      - Handle of item to be saved           */
/*             iLevel       - Level of item to be saved            */
/* Return value: none                                              */
/*****
void SaveTreeEngine( LPSTREAM  pStream,
                    HWND      hTreeView,
                    HTREEITEM hParent,
                    int        iLevel )
{
    SAVETREE st;

    while( hParent )                // Save all siblings of item
    {
        st.iLevel = iLevel;
        // Get data of item to be saved -----
        st.tvi.mask = TVIF_TEXT |
                    TVIF_IMAGE |
                    TVIF_SELECTEDIMAGE |
                    TVIF_PARAM |

```

```

        TVIF_STATE |
        TVIF_HANDLE;
st.tvi.hItem = hParent;
st.tvi.pszText = st.szText;
st.tvi.cchTextMax = sizeof( st.szText );

if( TreeView_GetItem( hTreeView, &st.tvi ) )
{
    // Calculate number of bytes required in SAVETREE -----
    st.iLen = sizeof(SAVETREE)-(MAX_PATH-(lstrlen(st.szText)+1));

    // Write data -----
    pStream->lpVtbl->Write( pStream, &st, st.iLen, NULL );

    // Save subitems -----
    SaveTreeEngine( pStream,
                    hTreeView,
                    TreeView_GetChild( hTreeView, hParent ),
                    iLevel + 1 );
}
// Get next sibling -----
hParent = TreeView_GetNextSibling( hTreeView, hParent );
}
}

/*****
/* TreeView_SaveToStream - Saves a TreeView in OLE storage */
/*-----*/
/* Parameter : hTreeView - Handle of TreeView Controls */
/*             lpszStorage - Name of OLE storage */
/*             lpszStream - Name of stream within storage */
/* Return value: TRUE - no error, FALSE - error */
/*****/
BOOL TreeView_SaveToStream( HWND hTreeView,
                           LPSTR lpszStorage,
                           LPSTR lpszStream )
{
    HRESULT hrCoInit, hr; // for HRESULTS of OLE operations
    LPSTORAGE pStorage; // Interface pointer for storage
    LPSTREAM pStream; // Interface pointer for stream
    WORD wsz[ MAX_PATH ]; // Wide Character buffer

    hrCoInit = CoInitialize( NULL ); // Initialize OLE

    // Convert ANSI string to Wide Character -----
    MultiByteToWideChar( CP_ACP, 0, lpszStorage, -1, wsz, sizeof( wsz ) );

    hr = StgCreateDocfile( wsz, STGM_DIRECT | // Create storage
                           STGM_READWRITE |
                           STGM_CREATE |
                           STGM_SHARE_EXCLUSIVE, 0, &pStorage );
    if( SUCCEEDED( hr ) ) // Storage created?
    {
        // Convert ANSI string to Wide Character -----
        MultiByteToWideChar( CP_ACP, 0, lpszStream, -1, wsz, sizeof( wsz ) );

        hr = pStorage->lpVtbl->CreateStream( pStorage, // Create stream
                                             wsz,
                                             STGM_DIRECT |
                                             STGM_READWRITE |
                                             STGM_CREATE |
                                             STGM_SHARE_EXCLUSIVE,
                                             0,
                                             0,
                                             &pStream );
        if( SUCCEEDED( hr ) ) // Stream created?
        {
            int iLen = 0;
            // Save tree -----
            SaveTreeEngine( pStream,
                            hTreeView,
                            TreeView_GetRoot( hTreeView ),
                            0 );

            // add blank item to end -----
            pStream->lpVtbl->Write( pStream, &iLen, sizeof( iLen ), NULL );

```

```

        pStream->lpVtbl->Release( pStream );           // release interface
    }
    pStorage->lpVtbl->Release( pStorage );             // release interface
}

if( SUCCEEDED( hrCoInit ) ) CoUninitialize();        // Uninstall OLE
return SUCCEEDED( hr );
}

/*****
/* TreeView_LoadFromStream - Loads tree from stream
/*-----
/* Parameter : hTreeView - Handle of TreeView control
/*             lpszStorage - Name of OLE storage
/*             lpszStream - Name of stream within the storage
/* Return value: TRUE - no error, FALSE - error
*****/
BOOL TreeView_LoadFromStream( HWND hTreeView,
                             LPSTR lpszStorage,
                             LPSTR lpszStream )
{
    HRESULT hrCoInit, hr;           // for HRESULTS of OLE operations
    LPSTORAGE pStorage;             // Interface pointer for storage
    LPSTREAM pStream;               // Interface pointer for stream
    WORD wsz[ MAX_PATH ];           // Wide Character buffer
    HTREEITEM hStack[ 99 ];         // Stack for parent items

    hrCoInit = CoInitialize( NULL ); // Initialize OLE

    // Convert ANSI string to Wide Character -----
    MultiByteToWideChar( CP_ACP, 0, lpszStorage, -1, wsz, sizeof( wsz ) );
    hr = StgOpenStorage( wsz, NULL, STGM_DIRECT |           // Open storage
                        STGM_READWRITE |
                        STGM_SHARE_EXCLUSIVE, NULL, 0, &pStorage );

    if( SUCCEEDED( hr ) )
    {
        // Convert ANSI string to Wide Character -----
        MultiByteToWideChar( CP_ACP, 0, lpszStream, -1, wsz, sizeof( wsz ) );
        hr = pStorage->lpVtbl->OpenStream( pStorage,         // Open stream
                                          wsz,
                                          0,
                                          STGM_DIRECT |
                                          STGM_READWRITE |
                                          STGM_SHARE_EXCLUSIVE,
                                          0,
                                          &pStream );

        if( SUCCEEDED( hr ) )
        {
            SAVETREE st;

            do
            {
                // Read length of a SAVETREE item -----
                pStream->lpVtbl->Read( pStream, &st.iLen, sizeof( int ), NULL );
                if( st.iLen )                               // last item?
                { // no, so read rest of item -----
                    TV_INSERTSTRUCT tvis;
                    pStream->lpVtbl->Read( pStream,
                                          &st.iLevel,
                                          st.iLen - sizeof( int ),
                                          NULL );
                    st.tvi.pszText = st.szText;

                    // Get handle of parent item -----
                    tvis.hParent = st.iLevel ? hStack[ st.iLevel - 1 ] : TVI_ROOT;

                    tvis.hInsertAfter = TVI_LAST;           // Add item to existing one

                    // Hide nonsensical states -----
                    st.tvi.stateMask &= ~(TVIS_SELECTED|
                                           TVIS_DROPHILITED|
                                           TVIS_CUT);

                    // The following mask flags are not saved with everything ---

```

```

        // else, so they must be set afterwards.          ---
        st.tvi.stateMask |= TVIS_STATEIMAGEMASK |
                                TVIS_OVERLAYMASK;
        // Custom extension!!!!!!                        ----
        // In this sample program the images are obtained  ----
        // via LVN_GETDISPINFO                             ----
        st.tvi.iImage = I_IMAGECALLBACK;
        st.tvi.iSelectedImage = I_IMAGECALLBACK;

        tvis.item = st.tvi;
        // Insert item and save handle, in case handle is a    ---
        // possible parent item.                                ---
        hStack[ st.iLevel ] = TreeView_InsertItem( hTreeView, &tvis );
    }
} while( st.iLen );
pStream->lpVtbl->Release( pStream );          // release interface
}
pStorage->lpVtbl->Release( pStorage );        // release interface
}

if( SUCCEEDED( hrCoInit ) ) CoUninitialize();    // Uninstall OLE
return SUCCEEDED( hr );
}

/*****
/* EditSubclass - Subclass function for EditControl of a TreeView */
/*-----*/
/* Parameter : default parameters */
/* Return value: default return value */
/*-----*/
/* Info: The EditControl requires all the keyboard input, since ESC */
/*        and RETURN will otherwise be interpreted as cancelling */
/*        the dialog box or pressing the default button. */
/*****
LRESULT WINAPI EditSubclass( HWND hWnd,
                            UINT wMsg,
                            WPARAM wp,
                            LPARAM lp )
{
    // I want all keystrokes -----
    if( wMsg == WM_GETDLGCODE ) return DLGC_WANTALLKEYS;

    // Call original EditBox function -----
    return CallWindowProc( g_lpOldEditProc, hWnd, wMsg, wp, lp );
}

/*****
/* SetDropHighlight - Identify TreeItem as DropTarget */
/*-----*/
/* Parameter : hTreeView - Handle of TreeView */
/*             hNewDropTarget - Handle of new DropTarget item */
/*             hOldDropTarget - Handle of old DropTarget item */
/* Return value: none */
/*****
void SetDropHighlight( HWND hTreeView,
                      HTREEITEM hNewDropTarget,
                      HTREEITEM hOldDropTarget )
{
    // Do you really want another item as a DropTarget? -----
    if( hNewDropTarget != hOldDropTarget )
    {
        TV_ITEM tvi;
        if( hNewDropTarget )
        {
            // Set DROPHILITED status of new DropTarget item -----
            tvi.hItem = hNewDropTarget;
            tvi.mask = TVIF_STATE; // only item.state and stateMask valid
            tvi.state = TVIS_DROPHILITED; // Set DropHilight
            tvi.stateMask = TVIS_DROPHILITED;
            TreeView_SetItem( hTreeView, &tvi );
        }
        if( hOldDropTarget )
        {
            // Delete DROPHILITED status of old DropTarget item -----
            tvi.hItem = hOldDropTarget;

```

```

        tvi.mask = TVIF_STATE;          // only item.state and stateMask valid
        tvi.state = 0;                  // Delete DropHighlight
        tvi.stateMask = TVIS_DROPHILITED;
        TreeView_SetItem( hTreeView, &tvi );
    }
    UpdateWindow( hTreeView );          // Display changes immediately
}

/*****
/* IsChildOf - Resolves "Paternity issues"
*/
-----
/* Parameter : hTreeView - Handle of TreeView
/*             hParent   - Parent item
/*             hItem     - Handle of item to be checked to determine
/*                       whether it is "below" the parent
/*                       in the hierarchy.
/* Return value: TRUE  - Item is subordinate to parent or identical
/*                   to parent.
/*                   FALSE - Item is higher than parent or in a different
/*                   branch.
*****/
BOOL IsChildOf( HWND hTreeView, HTREEITEM hParent, HTREEITEM hItem )
{
    if( hParent == hItem ) return TRUE;          // direct descendant ?
    hItem = TreeView_GetParent( hTreeView, hItem );
    // Consider any parent items as "Father" -----
    return hItem ? IsChildOf( hTreeView, hParent, hItem ) : FALSE;
}

/*****
/* CopyTree - Copy subtree
*/
-----
/* Parameter : hTreeView - Handle of TreeView
/*             hDst      - Item underneath which the subtree spec-
/*                       ified in hSrc is to be copied.
/*             hSrc      - Handle of first item of subtree to
/*                       be copied
/* Return value: none
*****/
void CopyTree( HWND hTreeView, HTREEITEM hDst, HTREEITEM hSrc )
{
    TV_ITEM tvi;
    char szText[ MAX_PATH ];

    // Prepare TV_ITEM structure for obtaining items -----
    tvi.mask = TVIF_TEXT |
               TVIF_IMAGE |
               TVIF_SELECTEDIMAGE |
               TVIF_PARAM |
               TVIF_STATE |
               TVIF_HANDLE;
    tvi.hItem = hSrc;          // Set handle of source item -----
    tvi.pszText = szText;
    tvi.cchTextMax = sizeof( szText );

    if( TreeView_GetItem( hTreeView, &tvi ) )          // Get item data
    {
        TV_INSERTSTRUCT tvis;
        HTREEITEM hChild, hNextParent;

        // Add item to destination -----
        tvis.hParent = hDst;
        tvis.hInsertAfter = TVI_LAST;
        tvis.item = tvi;
        hNextParent = TreeView_InsertItem( hTreeView, &tvis );

        if( hNextParent )
        {
            // Copy subtree of source item to destination you just created --

            // Get first subitem of the source item -----
            hChild = TreeView_GetChild( hTreeView, hSrc );
            while( hChild )
            { // Copy subitems -----
                CopyTree( hTreeView, hNextParent, hChild );
            }
        }
    }
}

```

```

        hChild = TreeView_GetNextSibling( hTreeView, hChild );
    }
}
}

/*****
/* GetTreePath - Get entire pathname from the root of the tree
/*               to the specified TreeItem. The individual names
/*               are separated by "\".
/*-----*/
/* Parameter :   hTree      - Handle of TreeView
/*               hTI        - Handle of TreeItem up to which the
/*               path is to be obtained.  <!AWK!>
/*               pszBuffer  - Address of buffer that receives
/*               the path.
/*               cbBuffer   - Size of buffer
/*               iRevLevel  - Reverse level of item specified in
/*                           hTI.
/* Return value : none
/*-----*/
/* Info : This function starts from a TreeItem and works its way up
/*         recursively to the root. Once there, it copies the names
/*         of the individual items to the passed buffer, adding a
/*         backslash (\) to all items up to the last one
/*         (iRevLevel=0).
*****/
void GetTreePath( HWND      hTree,
                  HTREEITEM hTI,
                  LPSTR    pszBuffer,
                  int       cbBuffer,
                  int       iRevLevel )
{
    static int iPos = 0;                // current position in buffer
    TV_ITEM tvi;                       // current TreeItem

    // Get parent item of the item specified in hTI -----
    HTREEITEM hParent = (HTREEITEM) SendMessage( hTree,
                                                  TVM_GETNEXTITEM,
                                                  TVGN_PARENT,
                                                  (LONG)hTI );

    if( hParent )                      // work your way up to the root
        GetTreePath( hTree,
                      hParent,
                      pszBuffer,
                      cbBuffer,
                      iRevLevel + 1 );
    else iPos = 0;

    // Copy the text of the current TreeItem to the buffer -----
    tvi.mask      = TVIF_TEXT;
    tvi.pszText   = &pszBuffer[ iPos ];
    tvi.cchTextMax = cbBuffer - iPos;
    tvi.hItem     = hTI;
    SendMessage( hTree, TVM_GETITEM, 0, (LONG)&tvi);

    iPos = strlen( pszBuffer );         // Get length of buffer

    if( ( iPos + 1 < cbBuffer ) && ( iRevLevel ) ) // room for "\" too ?
    {
        pszBuffer[ iPos++ ] = '\\';
        pszBuffer[ iPos ] = '\\0';
    }
}

/*****
/*DlgProc - Dialog function
/*-----*/
/* Parameter :   default parameters
/* Return value: default return value
*****/
BOOL WINAPI DlgProc( HWND hWnd, UINT wMsg, WPARAM wp, LPARAM lp )
{
    // static variables for simplified access -----
    static HWND      hTreeView;        // Handle of TreeView

```

```

static BOOL bDragging; // Is an item currently being dragged?
static HTREEITEM hDragItem; // item to be dragged
static HTREEITEM hDropTarget; // "hit" item
static HIMAGELIST hDragImage; // image for drag operation

switch( wParam )
{
case WM_INITDIALOG: // prepare static variables, etc.
{
int iImageCount;

// Get TreeView handle -----
hTreeView = GetDlgItem( hWnd, IDC_TREEVIEW );

// Set ImageList -----
TreeView_SetImageList( hTreeView, g_hImagesNormal, TVSIL_NORMAL );
TreeView_SetImageList( hTreeView, g_hImagesState, TVSIL_STATE );

// Set limits for UpDown controls -----
iImageCount = ImageList_GetImageCount(g_hImagesNormal);

UpDown_DlgSetRange(hWnd, IDC_IMAGE, 0, iImageCount);
UpDown_DlgSetRange(hWnd, IDC_SELECTEDIMAGE, 0, iImageCount);
UpDown_DlgSetRange(hWnd, IDC_EXPANDEDIMAGE, 0, iImageCount);
UpDown_DlgSetRange(hWnd, IDC_EXPANDEDSELECTEDIMAGE, 0, iImageCount);

UpDown_DlgSetRange(hWnd, IDC_OVERLAYINDEX, 0, 3);

iImageCount = ImageList_GetImageCount(g_hImagesState);
UpDown_DlgSetRange(hWnd, IDC_STATEIMAGE, 0, iImageCount);

bDragging = FALSE; // no Drag/Drop operation

// load previously saved tree -----
TreeView_LoadFromStream( hTreeView,
                        "TreeView.dat",
                        "TREEVIEW" );
}
break;
case WM_COMMAND:
switch( LOWORD( wParam ) )
{
case IDC_HASBUTTONS: // Change styles of TreeView control
case IDC_HASLINES:
case IDC_LINESATROOT:
case IDC_EDITLABELS:
case IDC_DISABLEDRAHDROP:
case IDC_SHOWSELALWAYS:
{
STYLESTRUCT st;

// Get old style -----
st.styleOld = GetWindowLong( hTreeView, GWL_STYLE );
st.styleNew = st.styleOld;
if( IsDlgButtonChecked( hWnd, LOWORD( wParam ) ) )
{
switch( LOWORD( wParam ) ) // Set flags -----
{
case IDC_HASBUTTONS:
st.styleNew |= TVS_HASBUTTONS;
break;
case IDC_HASLINES:
st.styleNew |= TVS_HASLINES;
break;
case IDC_LINESATROOT:
st.styleNew |= TVS_LINESATROOT;
break;
case IDC_EDITLABELS:
st.styleNew |= TVS_EDITLABELS;
break;
case IDC_DISABLEDRAHDROP:
st.styleNew |= TVS_DISABLEDRAHDROP;
break;
case IDC_SHOWSELALWAYS:
st.styleNew |= TVS_SHOWSELALWAYS;
}
}
}
}
}
}

```



```

        break;
    }
}
else
{
    switch( LOWORD( wp ) ) // Clear flags -----
    {
        case IDC_HASBUTTONS:
            st.styleNew &= ~TVS_HASBUTTONS;
            break;
        case IDC_HASLINES:
            st.styleNew &= ~TVS_HASLINES;
            break;
        case IDC_LINESATROOT:
            st.styleNew &= ~TVS_LINESATROOT;
            break;
        case IDC_EDITLABELS:
            st.styleNew &= ~TVS_EDITLABELS;
            break;
        case IDC_DISABLEDRAHDROP:
            st.styleNew &= ~TVS_DISABLEDRAHDROP;
            break;
        case IDC_SHOWSELALWAYS:
            st.styleNew &= ~TVS_SHOWSELALWAYS;
            break;
    }
}
// Set new style -----
SetWindowLong( hTreeView, GWL_STYLE, st.styleNew );

// and inform control of change -----
SendMessage( hTreeView,
              WM_STYLECHANGED,
              ( WPARAM ) GWL_STYLE,
              ( LPARAM ) &st );
}
break;
case IDC_INSERT: // Insert new item -----
{
    char szText[ MAX_PATH ];
    IMGCOMPOSITE ic;
    HTREEITEM hParent;

    // Get text -----
    GetWindowText( GetDlgItem( hWnd, IDC_TEXT ),
                  szText,
                  sizeof( szText ) );

    // Initialize lParam via IMGCOMPOSITE -----

    // Display current values of UpDown controls and set -----
    // as indexes of corresponding images -----
    ic.c.bImage = (BYTE)UpDown_DlgGetPos(hWnd, IDC_IMAGE);
    ic.c.bSelectedImage = (BYTE)UpDown_DlgGetPos(hWnd,
                                                  IDC_SELECTEDIMAGE);
    ic.c.bExpandedImage = (BYTE)UpDown_DlgGetPos(hWnd,
                                                  IDC_EXPANDEDIMAGE);
    ic.c.bExpandedSelectedImage = (BYTE)UpDown_DlgGetPos(hWnd,
                                                          IDC_EXPANDEDSELECTEDIMAGE);

    // new item is subordinate to current item -----
    hParent = TreeView_GetSelection( hTreeView );
    if( !hParent ) hParent = TVI_ROOT;

    InsertTreeItem( hTreeView, // Insert item
                   hParent,
                   szText,
                   I_IMAGECALLBACK, // s. TVN_GETDISPINFO
                   I_IMAGECALLBACK,
                   UpDown_DlgGetPos(hWnd, IDC_OVERLAYINDEX),
                   UpDown_DlgGetPos(hWnd, IDC_STATEIMAGE),
                   ic.lParam );
}
break;
case IDC_DELETE: // delete current item
{

```

```

HTREEITEM hItem;

// Get current item -----
hItem = TreeView_GetSelection( hTreeView );
if( hItem ) // and delete if present
    TreeView_DeleteItem( hTreeView, hItem );
}
break;
case IDCANCEL: // Close button of dialog box
    // Save tree -----
    TreeView_SaveToStream( hTreeView, "TreeView.dat", "TREEVIEW" );

    // End dialog -----
    EndDialog( hWnd, FALSE );
break;
}
break;
case WM_HSCROLL: // Edit UpDown controls
{
    // Unfortunately, finding out the ID of an UpDown control ----
    // is a bit complicated: ----
    switch( GetDlgCtrlID( (HWND) lp ) )
    {
        case IDC_IMAGE:
        case IDC_EXPANDEDIMAGE:
        case IDC_SELECTEDIMAGE:
        case IDC_EXPANDEDSELECTEDIMAGE:
        case IDC_OVERLAYINDEX:
        case IDC_STATEIMAGE:
        {
            HTREEITEM hItem;
            TV_ITEM tvi;
            IMGCOMPOSITE ic;

            // Get current item -----
            hItem = TreeView_GetSelection( hTreeView );

            if( hItem )
            {
                // Put together new value for lParam via IMGCOMPOSITE -----
                ic.c.bImage = (BYTE)UpDown_DlgGetPos(hWnd, IDC_IMAGE);
                ic.c.bSelectedImage = (BYTE)UpDown_DlgGetPos(hWnd,
                    IDC_SELECTEDIMAGE);
                ic.c.bExpandedImage = (BYTE)UpDown_DlgGetPos(hWnd,
                    IDC_EXPANDEDIMAGE);
                ic.c.bExpandedSelectedImage = (BYTE)UpDown_DlgGetPos(hWnd,
                    IDC_EXPANDEDSELECTEDIMAGE);

                // Set overlay and state image -----
                tvi.state = INDEXTOOVERLAYMASK( UpDown_DlgGetPos(hWnd,
                    IDC_OVERLAYINDEX) ) |
                    INDEXTOSTATEIMAGEMASK( UpDown_DlgGetPos(hWnd,
                    IDC_STATEIMAGE) );
                tvi.stateMask = TVIS_OVERLAYMASK | TVIS_STATEIMAGEMASK;

                // Change data in item -----
                tvi.mask = TVIF_PARAM | TVIF_STATE | TVIF_HANDLE;
                tvi.lParam = ic.lParam;
                tvi.hItem = hItem;

                TreeView_SetItem( hTreeView, &tvi );

                // Update TreeView -----
                SendMessage( hTreeView, WM_SETREDRAW, TRUE, 0 );
            }
        }
    }
break;
}
break;
case WM_NOTIFY:
    switch( LOWORD( wp ) )
    {
        case IDC_TREEVIEW:
        {
            LPNM_TREEVIEW pnmTV = (LPNM_TREEVIEW)lp;

```

```

switch( pnmtv->hdr.code)
{
    case NM_DBLCLK:                                // Double-click
    {
        TV_HITTESTINFO tvhtst;
        HTREEITEM hItem;

        GetCursorPos( &tvhtst.pt );                // Get mouse coordinates
        ScreenToClient( hTreeView, &tvhtst.pt );    // to TV-coords

        // Get item under the cursor -----
        hItem = TreeView_HitTest( hTreeView, &tvhtst );
        if( hItem )                                // was any item hit?
        {
            TV_ITEM tvi;
            char szText[ MAX_PATH ];

            tvi.mask = TVIF_TEXT;                    // Get item text
            tvi.hItem = hItem;
            tvi.pszText = szText;
            tvi.cchTextMax = sizeof( szText );
            TreeView_GetItem( hTreeView, &tvi );

            MessageBox( hWnd, szText, "Double-click", 0 );
        }
        else MessageBox( hWnd, "Missed!", "Double-click", 0 );
    }
    break;
    case TVN_BEGINDRAG:                            // Launch drag operation
    {
        POINT pt;
        // Get drag image of item to be dragged -----
        hDragImage = TreeView_CreateDragImage( hTreeView,
                                                pnmtv->itemNew.hItem );

        // Set drag image hotspot and launch drag operation -----
        if ( !ImageList_BeginDrag(hDragImage, 0, 0, 0 ) )
            return FALSE;

        // Cross-fade custom cursor -----
        ImageList_SetDragCursorImage( g_hDragCursors,
                                       0,
                                       -CRSR_HOTX,
                                       -CRSR_HOTY );

        ShowCursor( FALSE );                        // Disable system cursor

        pt.x = pnmtv->ptDrag.x;
        pt.y = pnmtv->ptDrag.y;

        ClientToScreen( hTreeView, &pt );

        // Launch drag operation and prepare buffer -----
        ImageList_DragEnter(NULL, pt.x, pt.y );

        SetCapture( hWnd );                          // Capture mouse

        bDragging = TRUE;                            // Drag operation waiting
        hDropTarget = 0;                             // still no DropTarget
        hDragItem = pnmtv->itemNew.hItem;             // Note DragItem
    }
    break;
    case TVN_BEGINLABELEDIT:                        // Input of a new item text
    {
        HWND hEdit;

        // Get handle of Edit control... -----
        hEdit = TreeView_GetEditControl( hTreeView );

        // ...and prepare to receive all keystrokes -----
        g_lpOldEditProc = (FARPROC)SetWindowLong( hEdit,
                                                    GWL_WNDPROC,
                                                    (LONG)EditSubclass );
    }
    return FALSE;                                    // Allow editing
    case TVN_ENDLABELEDIT:                          // End of text input

```

```

{
    TV_DISPINFO *ptvdi = (TV_DISPINFO *) lp;

    // Input completed by RETURN? -----
    if( ptvdi->item.pszText )
    {
        // yes, set new item text -----
        ptvdi->item.mask = TVIF_TEXT;
        TreeView_SetItem( hTreeView, &ptvdi->item );
    }
}
break;
case TVN_GETDISPINFO:                // Send info for display
{
    TV_DISPINFO *ptvdi = ( TV_DISPINFO *)lp;
    IMGCOMPOSITE ic;

    // lParam of item must be unravelled to supply index -
    // of correct image from the Image List -
    ic.lParam = ptvdi->item.lParam;

    // Does the TreeView control require info about image? ---
    if( ptvdi->item.mask & TVIF_IMAGE )
        // Expanded or normal image? -----
        ptvdi->item.iImage = ( ptvdi->item.state &
                                TVIS_EXPANDED ) ?
                                ic.c.bExpandedImage :
                                ic.c.bImage;

    // Does the TreeView control require information about -
    // selected image? -
    if( ptvdi->item.mask & TVIF_SELECTEDIMAGE )
        // Expanded or normal image? -----
        ptvdi->item.iSelectedImage = ( ptvdi->item.state &
                                        TVIS_EXPANDED ) ?
                                        ic.c.bExpandedSelectedImage :
                                        ic.c.bSelectedImage;
}
break;
case TVN_SELCHANGED:                // new item selected
{
    IMGCOMPOSITE ic;
    int iOverlay, iState;
    char szPath[ MAX_PATH ];

    // Adapt UpDown controls to circumstances of current
    // item.
    ic.lParam = pnmtv->itemNew.lParam;

    UpDown_DlgSetPos(hWnd, IDC_IMAGE, ic.c.bImage);
    UpDown_DlgSetPos(hWnd, IDC_SELECTEDIMAGE,
                    ic.c.bSelectedImage);
    UpDown_DlgSetPos(hWnd, IDC_EXPANDEDIMAGE,
                    ic.c.bExpandedImage);
    UpDown_DlgSetPos(hWnd, IDC_EXPANDEDSELECTEDIMAGE,
                    ic.c.bExpandedSelectedImage);

    // Adapt overlay and state image to UpDown control -----
    iOverlay = ( pnmtv->itemNew.state & 0x0300 )>>8;
    iState = ( pnmtv->itemNew.state & 0xF000 )>>12;

    UpDown_DlgSetPos(hWnd, IDC_OVERLAYINDEX, iOverlay);
    UpDown_DlgSetPos(hWnd, IDC_STATEIMAGE, iState);

    // Get current selection and transfer path to static ----
    // text box ----
    GetTreePath( hTreeView,
                pnmtv->itemNew.hItem,
                szPath,
                MAX_PATH,
                0 );
    SetWindowText( GetDlgItem( hWnd, IDC_TREEPATH ), szPath );
}
break;
}
}

```

```

        break;
    }
break;
case WM_MOUSEMOVE:
{
    if( bDragging )                // is an item being dragged now?
    {
        HTREEITEM hItem;
        TV_HITTESTINFO tvhtst;

        // Get mouse position -----
        tvhtst.pt.x = LOWORD( lp );
        tvhtst.pt.y = HIWORD( lp );

        // Get item under the cursor -----
        hItem = TreeView_HitTest( hTreeView, &tvhtst );

        if( hItem != hDropTarget )
        {
            // Hide drag image -----
            ImageList_DragShowNoLock(FALSE);

            // Item under cursor is new DropTarget -----
            SetDropHighlight( hTreeView,
                             hItem,
                             hDropTarget );
            hDropTarget = hItem;                // Note DropTarget

            // Display drag image again -----
            ImageList_DragShowNoLock(TRUE);
        }

        ClientToScreen( hTreeView, &tvhtst.pt);

        // Move drag image to mouse position -----
        ImageList_DragMove( tvhtst.pt.x,
                           tvhtst.pt.y );

        // Cursor shape indicates whether drag to DropTarget is allowed
        ImageList_SetDragCursorImage( g_hDragCursors,
                                       IsChildOf( hTreeView,
                                                    hDragItem,
                                                    hDropTarget )?1:0,
                                       -CRSR_HOTX,
                                       -CRSR_HOTY );
    }
}
break;
case WM_LBUTTONDOWN:
    if( bDragging )                // End drag operation ?
    {
        ImageList_EndDrag();        // Release ImageList internal buffer
        ImageList_DragLeave(NULL);    // Remove DragImage from window

        ImageList_Destroy( hDragImage );        // Destroy DragImage

        SetDropHighlight( hTreeView,        // Display DropTarget normally
                           NULL,
                           hDropTarget );

        if( hDropTarget )
        {
            // Can subtree be dragged to target ?
            if( !IsChildOf( hTreeView, hDragItem, hDropTarget ) )
            {
                // Copy subtree -----
                CopyTree( hTreeView, hDropTarget, hDragItem );
                // Delete "original" -----
                TreeView_DeleteItem( hTreeView, hDragItem );
            }
        }
        bDragging = FALSE;                // no more drag operations
        ReleaseCapture();                // Release capture
        ShowCursor( TRUE );
    }
}
break;
case WM_PAINT:

```

```

        PaintPicture( (HINSTANCE)GetWindowLong( hWnd, GWL_HINSTANCE ),
                      hWnd,
                      IDB_PCINTERN,
                      GetDlgItem( hWnd, IDC_PCINTERN ) );
    }
    break;
}
return FALSE;
}

/*****
/* WinMain - Start function */
/*-----*/
/* Parameter: default parameters */
/* Return value: default return value */
/*****
int WINAPI WinMain( HINSTANCE hInst,
                  HINSTANCE hPrev,
                  LPSTR lpCmdLine,
                  int nCmdShow )
{
    SHFILEINFO sfi;
    InitCommonControls(); // Initialize CommonControls

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

    // Use system ImageList as ImageList -----
    g_hImagesNormal= (HIMAGELIST)SHGetFileInfo( "",
                                                0,
                                                &sfi,
                                                sizeof( sfi ),
                                                SHGFI_SYSICONINDEX |
                                                SHGFI_SMALLICON );
    g_hImagesState = g_hImagesNormal; // State images from Sys-Img list too

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

    // Release ImageLists only if you created them yourself. System ---
    // ImageLists can NEVER be released. ---
    //ImageList_Destroy( g_hImagesNormal ); // Release ImageLists
    //ImageList_Destroy( g_hImagesState );

    ImageList_Destroy( g_hDragCursors ); // Destroy drag cursor
    return 0;
}

```