Article ID: 214814 - Last Review: July 11, 2005 - Revision: 1.4

How To Implementing Basic Drag and Drop In a Tree Control

This article was previously published under Q214814

SUMMARY

This article illustrates how to implement basic drag and drop functionality for a tree control using SDK style techniques.

MORE INFORMATION

In order to successfully implement drag and drop in a tree control complete the following steps:

1. Handle the TVN_BEGINDRAG notification.
2. Create a Drag Image.
3. Handle the WM_MOUSEMOVE message to move your drag image.
4. Handle the WM_LBUTTONUP (or WM_RBUTTONUP) message to deal with the end of the drag.

The TVN_BEGINDRAG notification lets your application know when the user has begun a drag and drop operation. Handling this notification affords you the opportunity to decide whether, in the context of your application, it is a legal operation or not. It's also a good time to initiate the creation of your drag image. In addition, the notification structure provides your application with information about which item is being dragged. The following section illustrates one way of handling the TVN_BEGINDRAG notification to create an image list for the drag operation and initially paint the image:

```
From your windows procedure . . . case WM_NOTIFY: { switch(((LPNMHDR)l_parm)->code) { case TVN_BEGINDRAG: { b_ret = begin_drag((LPNMTREEVIEW)l_parm) ; break ; } }
break ; . . .



BOOL begin_drag(LPNMTREEVIEW p_nmtree) { //Create an image list that holds our drag image h_drag = TreeView_CreateDragImage(h_tree, p_nmtree->itemNew.hItem) ; //begin
the drag operation ImageList_BeginDrag(h_drag, 0, 0, 0) ; //hide the cursor ShowCursor(FALSE) ; //capture the mouse SetCapture(GetParent(h_tree)) ; //set global flag
to indicate we are in the middle of a drag operation g_fdragging = TRUE ; //convert coordinates to screen coordinates ClientToScreen(h_tree, &(p_nmtree->ptDrag)) ;
//paint our drag image and lock the screen. ImageList_DragEnter(NULL, p_nmtree->ptDrag.x, p_nmtree->ptDrag.y) ; return TRUE ; }
```

Once the drag image is created, the next major section of code handles the WM_MOUSEMOVE message. The following code illustrates one way to handle implement WM_MOUSEMOVE to achieve a smooth transition:

```
From your windows procedure . . . case WM_MOUSEMOVE: { POINT pnt ; HTREEITEM h_item = NULL ; pnt.x = GET_X_LPARAM(l_parm) ; pnt.y = GET_Y_LPARAM(l_parm) ;
if(g_fdragging) { //unlock window and allow updates to occur ImageList_DragLeave(NULL) ; ClientToScreen(h_wnd, &pnt) ; //check with the tree control to see if we are
on an item TVHITTESTINFO tv_ht ; ZeroMemory(&tv_ht, sizeof(TVHITTESTINFO)) ; tv_ht.flags = TVHT_ONITEM ; tv_ht.pt.x = pnt.x ; tv_ht.pt.y = pnt.y ;
ScreenToClient(h_tree, &(tv_ht.pt)) ; h_item = (HTREEITEM)SendMessage(h_tree, TVM_HITTEST, 0, (LPARAM)&tv_ht) ; if(h_item) { //if we had a hit, then drop highlite the
item SendMessage(h_tree, TVM_SELECTITEM, TVGN_DROPHILITE, (LPARAM)h_item) ; } //paint the image in the new location ImageList_DragMove(pnt.x,pnt.y) ; //lock the
screen again ImageList_DragEnter(NULL, pnt.x, pnt.y) ; b_ret = TRUE ; } break ; }
```

The end of the drag operation occurs when a WM_LBUTTONUP or WM_RBUTTONUP message is received, depending on the type of mouse down event that initiated the drag. If you need to distinguish between the two at the time the drag is initiated, use the Windows API function GetKeyState to determine which mouse button is currently in use; that is, if you are using a different drag image for a right mouse drag or a left mouse drag. The following code illustrates one way to handle WM_LBUTTONUP to end the drag operation.

```
From your windows procedure . . . case WM_LBUTTONUP: { HTREEITEM h_item = NULL ; TVHITTESTINFO tv_ht ; TVITEM tv ; ZeroMemory(&tv, sizeof(TVITEM)) ;
ZeroMemory(&tv_ht, sizeof(TVHITTESTINFO)) ; if(g_fdragging) { ImageList_DragLeave(NULL) ; ImageList_EndDrag() ; ReleaseCapture() ; //determin if we let up on an item
GetCursorPos(&(tv_ht.pt)) ; ScreenToClient(h_tree, &(tv_ht.pt)) ; tv_ht.flags = TVHT_ONITEM ; h_item = (HTREEITEM)SendMessage(h_tree, TVM_HITTEST, 0, (LPARAM)&tv_ht)
; ShowCursor(TRUE) ; g_fdragging = FALSE ; b_ret = TRUE ; if(h_item) { /*we need to defer changing the selection until done processing this message post message
allows us to do this. */ PostMessage(h_wnd, M_CHANGEFOCUS, (WPARAM)0, (LPARAM)h_item) ; } } break ; }
```

The code for handling the M_CHANGEFOCUS message (a user defined message) is included only for clarity.

```
//message define #define M_CHANGEFOCUS WM_APP + 1 //use 0 for WPARAM and //HTREEITEM of new item for LPARAM From your windows procedure . . . case M_CHANGEFOCUS: {
TreeView_SelectDropTarget(h_tree, NULL) ; TreeView_SelectItem(h_tree, (HTREEITEM)l_parm) ; break ; }
```

How an application handles the end of a drag and drop operation is a decision made by the individual developer but if you plan to send further messages to the tree control based on the drag, it is recommended that you use a postmessage scheme. This is a highly recommended method that allows the control to finish processing the mouse message.

APPLIES TO

- Microsoft Platform Software Development Kit-January 2000 Edition, when used with:
  Microsoft Windows 98 Standard Edition

Microsoft Windows 95
Microsoft Windows NT Server 4.0 Standard Edition
Microsoft Windows NT Workstation 4.0 Developer Edition

Keywords: kbhowto kbimglist kbtreeview kbctrl KB214814

¿Necesita más ayuda?

Contactar con un profesional de soporte técnico por correo electrónico, online o por teléfono

Ayuda y soporte

Microsoft