[**http://blog.sina.com.cn/s/blog\_70aa9ba00100qn4e.html**](http://blog.sina.com.cn/s/blog_70aa9ba00100qn4e.html%20)

[**http://www.codeproject.com/Articles/9391/Dragging-tree-nodes-in-C**](http://www.codeproject.com/Articles/9391/Dragging-tree-nodes-in-C)

**那么如何为TreeNode设置图像呢？**

如下文字来自MSDN：

可以在树节点旁显示图像，方法是将一个 ImageList 分配给 ImageList 属性，然后通过引用 Image 在 ImageList 中的索引值来分配该 Image。

使用下面的属性分配图像：

将 ImageIndex 属性设置为当树节点未选定时所显示的 Image 的索引值。

将 SelectedImageIndex 属性设置为当树节点被选定时要显示的 Image 的索引值。

ImageIndex 和 SelectedImageIndex 属性值所引用的图像是所有分配给 Nodes 集合的树节点显示的默认图像。 每个树节点都可以通过设置 TreeNode.ImageIndex 和 TreeNode.SelectedImageIndex 属性来取代默认的图像。

看完了这段文字，你知道该如何操作了么？我觉得可能有好多人还是不是完全的理解，那我们就已程序员的语言来描述一下吧:

首先定义一个ImageList变量并将其赋值给TreeView的ImageList，

1 private System.Windows.Forms.ImageList imageList007;  
2 this.imageList007 = new System.Windows.Forms.ImageList(this.components);  
3 this.treeView1.ImageList = this.imageList007;

然后需要对图标进行设置,

[复制代码](javascript:void(0);)

this.imageList007.ImageStream =

((System.Windows.Forms.ImageListStreamer)(resources.GetObject("imageList007.ImageStream")));  
this.imageList007.TransparentColor = System.Drawing.Color.Transparent;  
this.imageList007.Images.SetKeyName(0, "Icon01.ico");  
this.imageList007.Images.SetKeyName(1, "Icon02.ico");  
this.imageList007.Images.SetKeyName(2, "Icon03.ico");  
this.imageList007.Images.SetKeyName(3, "Icon04.ico");  
this.imageList007.Images.SetKeyName(4, "Icon05.ico");

[复制代码](javascript:void(0);)

最后可以对TreeView 的ImageIndex 和 SelectedImageIndex 属性进行相应的设置，选择相应的图像，也可以对每个动态生成的节点进行独立的图像设置，到此你已经成功的设置了TreeNode的图像。

**那又如何实现Drag and Drop 操作呢？**

首先我们需要清楚Drag and Drop的源和目标都是什么，也就是说从哪向哪拖拽，每个部分承担的职责是什么，而只有在源和目标上都做了相应的处理，拖拽过程才可能是完整有效的。

**源组件处理：**

ItemDrag事件: 当鼠标开始对源树节点进行拖拽的时候，会引发这个事件，也就是说这个事件的发出者应该是源组件， 在这个事件处理函数中需要调用DoDragDrop方法 初始化并开始一个拖拽过程。

**目标组件处理：**

DragEnter事件: 当初始化操作结束以后，我们需要在拖拽目标处处理DragEnter事件，这个事件发生在源节点被拖动到目标组件范围内的某个点的时候，在这个事件中我们可以对这个拖拽的合法性进行验证，并且设置不同的鼠标形状来表示不同的状态，DragDropEffects 结构用于对鼠标形状进行设置。

**目标组件处理：**

DragDrop 事件: 这个事件用于在目标组件中处理拖拽事件，发生于源节点已经被拖拽到目的组件处，这时我们可以对拖拽过来的节点进行解析处理等实际工作。

**注意:为了完成拖拽操作源组件和目标组件的AllowDrop都需要设置成true.**

示例代码，如下代码完成了将一个TreeNode拖拽到一个Form上需要完成的工作：

    要实现TreeView的拖拽功能，首先我们必须将其AllowDrop属性设成true，这样就保证了此TreeView的节点是可以被拖拽的。接下来 我们为TreeView添加3个事件，它们分别是1.ItemDrag事件（当用户开始拖动节点时发生。）2.DragEnter事件（在将对象拖入控件 的边界时发生。）3.DragDrop事件（在完成拖放操作时发生。）

private Point Position = new Point(0, 0);  
  
private void treeView1\_ItemDrag(object sender, ItemDragEventArgs e)  
{  
   DoDragDrop(e.Item, DragDropEffects.Move);  
}  
  
private void treeView1\_DragEnter(object sender, DragEventArgs e)  
{  
   if (e.Data.GetDataPresent(typeof(TreeNode)))  
   e.Effect = DragDropEffects.Move;  
   else  
   e.Effect = DragDropEffects.None;  
}  
  
private void treeView1\_DragDrop(object sender, DragEventArgs e)  
{  
   TreeNode myNode = null;  
   if (e.Data.GetDataPresent(typeof(TreeNode)))  
   {  
      myNode = (TreeNode)(e.Data.GetData(typeof(TreeNode)));  
   }  
   else  
   {  
      MessageBox.Show("error");  
   }  
   Position.X = e.X;  
   Position.Y = e.Y;  
   Position = treeView1.PointToClient(Position);  
   TreeNode DropNode = this.treeView1.GetNodeAt(Position);  
   // 1.目标节点不是空。2.目标节点不是被拖拽接点的子节点。3.目标节点不是被拖拽节点本身  
   if (DropNode != null && DropNode.Parent != myNode && DropNode != myNode)  
   {  
      TreeNode DragNode = myNode;  
      // 将被拖拽节点从原来位置删除。  
      myNode.Remove();  
      // 在目标节点下增加被拖拽节点  
      DropNode.Nodes.Add(DragNode);  
   }  
   // 如果目标节点不存在，即拖拽的位置不存在节点，那么就将被拖拽节点放在根节点之下  
   if (DropNode == null)  
   {  
      TreeNode DragNode = myNode;  
      myNode.Remove();  
      treeView1.Nodes.Add(DragNode);  
   }  
}

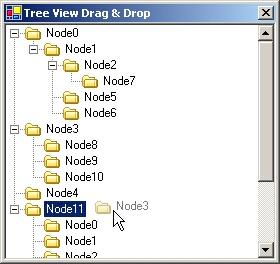
**Dragging tree nodes in C#**

[Michea77](http://www.codeproject.com/script/Membership/View.aspx?mid=75449), 23 Jan 2005

|  |  |
| --- | --- |
| http://s.codeproject.com/script/Ratings/Images/stars-fill-lg.png  http://s.codeproject.com/script/Ratings/Images/stars-empty-lg.png | 4.80 (59 votes) |
|  | |  |  |  | | --- | --- | --- | | Rate this: | [vote 1vote 2vote 3vote 4vote 5](http://www.codeproject.com/Articles/9391/Dragging-tree-nodes-in-C) |  | |

This article shows how to implement an Explorer like treeview drag and drop in C#.

* [Download source - 9.15 Kb](http://www.codeproject.com/KB/tree/TreeViewDragDrop/TreeViewDragDrop.zip)



**Introduction**

During the implementation of a C# application, I was faced with the problem of adding drag and drop functionality to a tree view. So far no problems, but in order to make the whole a bit fancier, I decided to add image dragging too, as it is done by the Windows Explorer when dragging files or directories. Here I got into troubles, since image dragging isn't supported by the .NET controls. Not finding satisfactory code in the Internet (maybe I'm not a good surfer...), I decided to try it by myself. Having fun on it, I improved the code by adding scrolling to the component so that a dragged element can be dropped everywhere on the control.

This article will only briefly describe the basics of drag and drop, as it is quite straightforward to implement and there already are a lot of good articles about it. The aim of the article is to describe how image dragging and automatic scrolling while dragging can be implemented in C#.

**TreeView Drag and Drop**

In order to allow Drag and Drop on a TreeView, the AllowDrop flag must be set and handlers for the the following events (or some of them) must be implemented:

* ItemDrag - This event is fired as soon as a drag operation is started. This event is specific for listviews and treeviews. The dragged element is passed as an argument of the event. The handler of this event should contain the DoDragDrop() call that begins the drag and drop operation.
* DragOver - This event is fired when the user drags over a drag and drop control with the mouse.
* DragEnter - This event is fired when the user moves the mouse onto the control while dragging an element.
* DragLeave - This event is fired when the user leaves the control with the mouse while dragging an element.
* DragDrop - This event is fired when the user releases the mouse over the drop target.
* GiveFeedback - This event gives feedback about the current drag effect and cursor.

**Image dragging**

The implementation of image dragging requires functionalities that first of all create a ghost image of the dragging element and then move this image as mouse cursor moves over the TreeView control. Part of the needed functionalities are available in the ImageList implementation of Win32 (WinAPI). In order to call these functions, I wrote the class DragHelper that accesses them via P/Invoke.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/9391/Dragging-tree-nodes-in-C)

public class DragHelper

{

[DllImport("comctl32.dll")]

public static extern bool InitCommonControls();

[DllImport("comctl32.dll", CharSet=CharSet.Auto)]

public static extern bool ImageList\_BeginDrag(

IntPtr himlTrack, // Handler of the image list containing the image to drag

int iTrack, // Index of the image to drag

int dxHotspot, // x-delta between mouse position and drag image

int dyHotspot // y-delta between mouse position and drag image

);

[DllImport("comctl32.dll", CharSet=CharSet.Auto)]

public static extern bool ImageList\_DragMove(

int x, // X-coordinate (relative to the form,

// not the treeview) at which to display the drag image.

int y, // Y-coordinate (relative to the form,

// not the treeview) at which to display the drag image.

);

[DllImport("comctl32.dll", CharSet=CharSet.Auto)]

public static extern void ImageList\_EndDrag();

[DllImport("comctl32.dll", CharSet=CharSet.Auto)]

public static extern bool ImageList\_DragEnter(

IntPtr hwndLock, // Handle to the control that owns the drag image.

int x, // X-coordinate (relative to the treeview)

// at which to display the drag image.

int y // Y-coordinate (relative to the treeview)

// at which to display the drag image.

);

[DllImport("comctl32.dll", CharSet=CharSet.Auto)]

public static extern bool ImageList\_DragLeave(

IntPtr hwndLock // Handle to the control that owns the drag image.

);

[DllImport("comctl32.dll", CharSet=CharSet.Auto)]

public static extern bool ImageList\_DragShowNolock(

bool fShow // False to hide, true to show the image

);

static DragHelper()

{

InitCommonControls();

}

}

The first thing to do when we start dragging an element is to create the ghost image of the tree node. Help is provided by the function ImageList\_BeginDrag which creates for us a ghost image. This function needs as parameter the handler of an ImageList with the image to be made transparent in it. To create the image of the tree node to drag, a new bitmap is created and the icon and the label are drawn in it. At the end of the dragging operation, the ghost image is destroyed by a call to the ImageList\_EndDrag function. We do all this in the ItemDrag event handler.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/9391/Dragging-tree-nodes-in-C)

private void treeView\_ItemDrag(object sender,

System.Windows.Forms.ItemDragEventArgs e)

{

// Get drag node and select it

this.dragNode = (TreeNode)e.Item;

this.treeView1.SelectedNode = this.dragNode;

// Reset image list used for drag image

this.imageListDrag.Images.Clear();

this.imageListDrag.ImageSize =

new Size(this.dragNode.Bounds.Size.Width

+ this.treeView1.Indent, this.dragNode.Bounds.Height);

// Create new bitmap

// This bitmap will contain the tree node image to be dragged

Bitmap bmp = new Bitmap(this.dragNode.Bounds.Width

+ this.treeView1.Indent, this.dragNode.Bounds.Height);

// Get graphics from bitmap

Graphics gfx = Graphics.FromImage(bmp);

// Draw node icon into the bitmap

gfx.DrawImage(this.imageListTreeView.Images[0], 0, 0);

// Draw node label into bitmap

gfx.DrawString(this.dragNode.Text,

this.treeView1.Font,

new SolidBrush(this.treeView1.ForeColor),

(float)this.treeView1.Indent, 1.0f);

// Add bitmap to imagelist

this.imageListDrag.Images.Add(bmp);

// Get mouse position in client coordinates

Point p = this.treeView1.PointToClient(Control.MousePosition);

// Compute delta between mouse position and node bounds

int dx = p.X + this.treeView1.Indent - this.dragNode.Bounds.Left;

int dy = p.Y - this.dragNode.Bounds.Top;

// Begin dragging image

if (DragHelper.ImageList\_BeginDrag(this.imageListDrag.Handle, 0, dx, dy))

{

// Begin dragging

this.treeView1.DoDragDrop(bmp, DragDropEffects.Move);

// End dragging image

DragHelper.ImageList\_EndDrag();

}

}

When the mouse is now moved while a tree node is dragged, the ghost image should follow the mouse cursor. This is done by the function ImageList\_DragMove. We implement it in the DragOver event handler.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/9391/Dragging-tree-nodes-in-C)

private void treeView1\_DragOver(object sender,

System.Windows.Forms.DragEventArgs e)

{

// Compute drag position and move image

Point formP = this.PointToClient(new Point(e.X, e.Y));

DragHelper.ImageList\_DragMove(formP.X - this.treeView1.Left,

formP.Y - this.treeView1.Top);

...

}

If we leave the TreeView, the ghost image should disappear, and as soon as we re-enter the control, the image should appear again. This is done by the functions ImageList\_DragLeave and ImageList\_DragEnter. The function ImageList\_DragEnter also locks the window for updates to allow a clean dragging of the image. ImageList\_DragLeave respectively releases the update lock. We implement these two functions in the corresponding event handlers (treeView1\_DragEnter and treeView1\_DragLeave).

While an element is dragged, Windows automatically changes the mouse cursor according to the drag effect (copy, move, none, ...). In our example, we use the DragDropEffects.Move drag effect. The mouse cursor we want to have while dragging the ghost image is the normal pointer cursor. The cursor can be set in the GiveFeedback event handler.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/9391/Dragging-tree-nodes-in-C)

private void treeView1\_GiveFeedback(object sender,

System.Windows.Forms.GiveFeedbackEventArgs e)

{

if(e.Effect == DragDropEffects.Move)

{

// Show pointer cursor while dragging

e.UseDefaultCursors = false;

this.treeView1.Cursor = Cursors.Default;

}

else e.UseDefaultCursors = true;

}

As soon as the user drops the element and thus terminates the dragging operation, the control updates must be unlocked calling the function ImageList\_DragLeave. The DoDragDrop() call (see treeView1\_ItemDrag) terminates and the ghost image is released with ImageList\_EndDrag.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/9391/Dragging-tree-nodes-in-C)

private void treeView1\_DragDrop(object sender,

System.Windows.Forms.DragEventArgs e)

{

// Unlock updates

DragHelper.ImageList\_DragLeave(this.treeView1.Handle);

...

}

}

**Scrolling while dragging**

Without scrolling, we can not reach each node within a tree while we are dragging an element, unless the tree is entirely visible on the screen. We begin scrolling the control if the mouse cursor reaches the top or the bottom of the TreeView control. The scrolling is achieved through the EnsureVisible() method of the TreeNode class. To scroll up, we get the previous visible node through the property PrevVisibleNode of the TreeNode class, and set it visible. Analogously, we scroll down with the property NextVisibleNode and a call to EnsureVisible(). As soon as we begin dragging a node, a timer is started. At each tick of the timer, the current position of the cursor is checked, and if it is near the upper or lower border of the control, the tree is scrolled. To avoid that the dragging image interferes with the scrolling producing ugly graphical effects, we briefly hide the drag image and unlock the paint updates with the function ImageList\_DragShowNolock. That's it!

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/9391/Dragging-tree-nodes-in-C)

private void timer\_Tick(object sender, EventArgs e)

{

// get node at mouse position

Point pt = PointToClient(Control.MousePosition);

TreeNode node = this.treeView1.GetNodeAt(pt);

if(node == null) return;

// if mouse is near to the top, scroll up

if(pt.Y < 30)

{

// set actual node to the upper one

if (node.PrevVisibleNode!= null)

{

node = node.PrevVisibleNode;

// hide drag image

DragHelper.ImageList\_DragShowNolock(false);

// scroll and refresh

node.EnsureVisible();

this.treeView1.Refresh();

// show drag image

DragHelper.ImageList\_DragShowNolock(true);

}

}

// if mouse is near to the bottom, scroll down

else if(pt.Y > this.treeView1.Size.Height - 30)

{

if (node.NextVisibleNode!= null)

{

node = node.NextVisibleNode;

DragHelper.ImageList\_DragShowNolock(false);

node.EnsureVisible();

this.treeView1.Refresh();

DragHelper.ImageList\_DragShowNolock(true);

}

}

}