Advance Topics in Windows

Implementing Drag-and-Drop Functionality

Drag-and-drop functionality is ubiquitous in Windows Forms programming. It refers to allowing the user to grab data such as text, an image, or another object with the mouse and drag it to another control. When the mouse button is released over the other control, the data that is being dragged is dropped onto the control and a variety of effects can then occur.

Dragging and dropping is similar to cutting and pasting. The mouse pointer is positioned over a control and the mouse button is pressed. Data is copied from a source control; when the mouse button is released, the drop action is completed. All code for copying the data from the source control and any actions taken on the target control must be explicitly coded.

The drag-and-drop process is primarily an event-driven process. There are events that occur on the source control and events that occur on the target control.

TABLE 1: Source Control Events Involved in Implementing Drag-and-Drop

EVENT	DESCRIPTION
MouseDown	*Occurs when the mouse button is pressed while the pointer is over the control. In general, the* DoDragDrop *method is called in the method that handles this event.*
GiveFeedBack	*Provides an opportunity for the user to set a custom mouse pointer.*
QueryContinueDrag	*Enables the drag source to determine whether a drag event should be cancelled.*

TABLE 2: Target Control Events Involved in Implementing Drag-and-Drop

EVENT	DESCRIPTION
DragEnter	Occurs when an object is dragged within a control's bounds. The handler for this event receives a *DragEventArgs* object.
Drag Over	Occurs when an object is dragged over a target control. The handler for this event receives a *DragEventArgs* object.
Drag Drop	Occurs when the mouse button is released over a target control. The handler for this event receives a *DragEventArgs* object.
DragLeave	Occurs when an object is dragged out of the control's bounds.

In addition, the *DoDragDrop* method on the source control is required to initiate the drag-and-drop process and the target control must have the *AllowDrop* property set to *True*.

The General Sequence of a Drag-and-Drop Operation

The general sequence of events that takes place in a drag-and-drop operation is as follows:

1. The drag-and-drop operation is initiated by calling the *DoDragDrop* method on the source control. This is usually done in the *MouseDown* event handler. *DoDragDrop* copies the desired data from the source control to a new instance of *DataObject* and sets flags that specify which effects are allowed with this data.
2. The *GiveFeedBack* and *QueryContinueDrag* events are raised at this point. The *GiveFeedback* event handler can set the mouse pointer to a custom shape, and the *QueryContinueDrag* event handler can be used to determine if the drag operation should be continued or aborted.
3. The mouse pointer is dragged over a target control. Any control that has the *AllowDrop* property set to *True* is a potential drop target. When the mouse pointer enters a control with the *AllowDrop* property set to *True*, the *DragEnter* event for that control is raised. The *DragEventArgs* object that the

event handler receives can be examined to determine if data appropriate for the target control is present. If so, the *Effect* property of the *DragEventArgs* object can then be set to an appropriate value.

4. The user releases the mouse button over a valid target control, raising the *DragDrop* event. The code in the *DragDrop* event handler then obtains the dragged data and takes whatever action is appropriate in the target control.

The Drag Drop Effects Enumeration

To complete a drag-and-drop operation, the drag effect specified in the *DoDragDrop* method must match the value of the *Effect* parameter of the *DragEventArgs* object associated with the drag-and-drop event, which is generally set in the *DragEnter* handler. The *Effect* property is an instance of the *DragDropEffects* enumeration. The members of the *DragDropEffects* enumeration are described in Table 3.

Table 3 : DragDropEffects Enumeration Members

MEMBER	EXPLANATION
All	Data is copied, removed from the drag source, and scrolled in the target.
Copy	The data is copied to the target.
Link	The data is linked to the target.
Move	The data is moved to the target.
None	The target does not accept the data.
Scroll	Scrolling is about to start or is currently occurring in the target.

Note that the main function of the *Effect* parameter is to change the mouse cursor when it is over the target control. The value of the *Effect* parameter has no actual effect on the action that is executed except that when the *Effect* parameter is

set to *None*, no drop can take place on that control because the *DragDrop* event will not be raised.

Initiating the Drag-and-Drop Operation

The drag-and-drop operation is initiated by calling the *DoDragDrop* method on the source control. The *DoDragDrop* method takes two parameters: an *Object*, which represents the data to be copied to the *DataObject*, and an instance of *DragDropEffects*, which specifies what drag effects will be allowed with this data. The following example demonstrates how to copy the text from a text box and set the allowed effects to *Copy* or *Move:*

```
private void textBoxLMouseDown (object sender, MouseEventArgs e)
{
    textBoxl.DoDragDrop(textBoxl.Text, DragDropEffects.Copy |DragDropEffects.Move);
}
```

Note that you can use the | operator (C#) to combine members of the *DragDropEffects* enumeration to indicate multiple effects.

Handling the DragEnter Event

The *DragEnter* event should be handled for every target control. This event occurs when a drag-and-drop operation is in progress and the mouse pointer enters the control. This event passes a *DragEventArgs* object to the method that handles it, and you can use the *DragEventArgs* object to query the *DataObject* associated with the drag-and-drop operation. If the data is appropriate for the target control, you can set the *Effect* property to an appropriate value for the control. The following example demonstrates how to examine the data format of the *DataObject* and set the *Effect* property:

```
private void textBox2_DragEnter (object sender, DragEventArgs e)
 {
         if (e.Data.CetDataPresent(DataFormats.Text))
         {
              e.Effect = DragDropEffects.Copy;
         }
}
```

Handling the DragDrop Event

When the mouse button is released over a target control during a drag-and-drop operation, the *DragDrop* event is raised. In the method that handles the *DragDrop* event, you can use the *GetData* method of the *DataObject* to retrieve the copied data from the *DataObject* and take whatever action is appropriate for the control. The following example demonstrates how to drop a *String* into a *TextBox:*

```
private void textBox2_DragDrop(object sender, DragEventArgs e)
{
        textBox2.Text = (string)e.Data.GetData(DataFormats.Text);
}
```

Implementing Drag-and-Drop Between Applications

The system intrinsically supports drag-and-drop operations between .NET Framework applications. You don't need to take any additional steps to enable drag-and-drop operations that take place between applications. The only conditions that must be satisfied to enable a drag-and-drop operation between applications are:

- The target control must allow one of the drag effects specified in the *DoDragDrop* method call.
- The target control must accept data in the format that was set in the *DoDragDrop* method call.

Implementing Drag-and-Drop in a Tree View Control

A common scenario for the *TreeView* control is to allow the user to rearrange the structure of the tree at run time. This can be implemented with drag-and-drop. Drag-and-drop in a *TreeView* control is slightly different from that in regular controls. When a drag operation is initiated on a *TreeView* node, the *TreeView* control raises the *ItemDrag* event, which passes an instance of *ItemDragEventArgs* to the method that handles the event. The *ItemDragEventArgs* object contains a reference to the *TreeNode* that is being dragged, and this reference can be copied to the *DataObject* in the *DoDragDrop* method. The following procedure describes how to implement drag-and-drop functionality in a *TreeView* control.

1. Set the *AllowDrop* property of the *TreeView* to *True*. This enables the *DragEnter* and *DragDrop* events to be raised from the *TreeView* control.
2. In the *ItemDrag* event handler of the *TreeView*, call the *DoDragDrop* method of the *TreeView*, specifying the *Item* property of the *ItemDragEventArgs* object as the *Data* parameter, as shown in the following example:

```
private void TreeView1_ItemDrag(object sender,  ItemDragEventArgs e)
{
      treeView1.DoDragDrop(e.Item, DragDropEffects.Move);
}
```

3. In the DragEnter event of the TreeView event handler, set the Effect property of the DragDropEventArgs to an appropriate value, as shown in the following example:

```
private void treeView1_DragEnter(object sender, DragEventArgs e)
{
       e.Effect = DragDropEffects.Move;
}
```

4. In the *DragDrop* event handler, examine the data contained in the *DataObject* of *Drag DropEventArgs* to determine if a *TreeNode* is present. If a *TreeNode* is present, execute code to move the *TreeNode* to the appropriate spot. The following code example demonstrates how to move a dropped node into the child node structure of the node under the mouse pointer:

```
private void treeView1DragDrop(object sender, DragEventArgs e)
{
   TreeNode aNode;
   // Checks to see if a TreeNode is present

   if (e.Data.CetDataPresent("System.Windows.Forms.TreeNode", false))

   {
               Point apoint;
               TreeNode TargetNode;

               // Gets the point under the mouse pointer
               apoint =treeView1.PointToClient(new Point(e.X, e.Y));
               //apoint = ((TreeView)sender).PointToClient(new Point(e.X, e.Y));

               // Gets the node at the specified point

               TargetNode = ((TreeView)sender).GetNodeAt(apoint);
               aNode = (TreeNode)e.Data.GetData("System.Windows.Forms.TreeNode");

               // Adds the dragged node as a child to the target node

               TargetNode.Nodes.Add((TreeNode)aNode.Clone());
               TargetNode.Expand();

               // Removes original node
               aNode.Remove();
     }
}
```