Visual Basic® 2005 with .NET 3.0 Programmer's Reference

Chapter 13: Drag and Drop, and the Clipboard

ISBN-10: 0-470-13705-3
ISBN-13: 978-0-470-13705-5

13

Drag and Drop, and the Clipboard

The clipboard is an object where programs can save and restore data. A program can save data in multiple formats and retrieve it later, or another program might retrieve the data. Windows, rather than Visual Basic, provides the clipboard, so it is available to every application running on the system, and any program can save or fetch data from the clipboard.

The clipboard can store remarkably complex data types. For example, an application can store a representation of a complete object in the clipboard for use by other applications that know how to use that kind of object.

Drag-and-drop support enables the user to drag information from one control to another. The controls may be in the same application or in different applications. For example, your program could let the user drag items from one list to another, or it could let the user drag files from Windows Explorer into a file list inside your program.

A drag occurs in three main steps. First, a *drag source* control starts the drag, usually when the user presses the mouse down on the control. The control starts the drag, indicating the data that it wants to drag and the type of drag operations it wants to perform (such as Copy, Link, or Move).

When the user drags over a control, that control is a possible *drop target*. The control examines the kind of data being dragged and the type of drag operation requested (such as Copy, Link, or Move). The drop target then decides whether it will allow the drop and what type of feedback it should give to the user. For example, if the user drags a picture over a label control, the label might refuse the drop and display a no drop icon (a circle with a line through it). If the user drags the picture over a `PictureBox` that the program is using to display images, it might display a drop link icon (a box with a curved arrow in it).

Finally, when the user releases the mouse, the current drop target receives the data and does whatever is appropriate. For example, if the drop target is a `TextBox` control and the data is a string, the `TextBox` control might display the string. If the same `TextBox` control receives a file name, it might read the file and display its contents.

The following sections describe drag-and-drop events in more detail and give several examples of common drag-and-drop tasks. The section “Using the Clipboard” near the end of the chapter explains how to use the clipboard. Using it is very similar to using drag and drop, although it doesn’t require as much user feedback, so it is considerably simpler.

Drag-and-Drop Events

The drag source control starts a drag operation by calling its `DoDragDrop` method. It passes this method the data to be dragged and the type of drag operation that the control wants to perform. The drag type can be Copy, Link, or Move.

If you are dragging to other general applications, the data should be a standard data type such as a `String` or `Bitmap` so that the other application can understand it. If you are dragging data within a single application or between two applications that you have written, you can drag any type of data. This won’t necessarily work with general objects and arbitrary applications. For example, WordPad doesn’t know what an `Employee` object is, so you can’t drop an `Employee` on it.

As the user drags the data around the screen, Visual Basic sends events to the controls it moves over. Those controls can indicate whether they will accept the data and how they can accept it. For example, a control might indicate that it will allow a Copy, but not a Move. The following table describes the events that a drop target receives as data is dragged over it.

Event	Purpose
`DragEnter`	The drag is entering the control. The control can examine the type of data available and set `e.Effect` to indicate the types of drops it can handle. These can include All, Copy, Move, Link, and None. The control can also display some sort of highlighting to indicate that the data is over it. For example, it might display a dark border or shade the area where the new data would be placed.
`DragLeave`	The drag has left the control. If the control displays some sort of highlighting or other indication that the drag is over it in the `DragEnter` event, it should remove that highlight now.
`DragOver`	The drag is over the control. This event continues to fire a few times per second until the drag is no longer over the control. The control may take action to indicate how the drop will be processed much as the `DragEnter` event handler does. For example, as the user moves the mouse over a `ListBox`, the control might highlight the list item that is under the mouse to show that this item will receive the data. The program can also check for changes to the mouse or keyboard. For example, it might allow a Copy if the Ctrl key is pressed and a Move if the Ctrl key is not pressed.
`DragDrop`	The user has dropped the data on the control. The control should process the data.

A drop target with simple needs can specify the drop actions it will allow in its `DragEnter` event handler and not provide a `DragOver` event handler. It knows whether it will allow a drop based solely on the type of item being dropped. For example, a graphical application might allow the user to drop a bitmap on it, but not a string.

A more complex target that must track such items as the keyboard state, mouse position, and mouse button state can provide a `DragOver` event handler and skip the `DragEnter` event handler. For example, a circuit design application might check the drag's position over its drawing surface, and highlight the location where the dragged item would be positioned. As the user moves the object around, the `DragOver` event would continue to fire so the program could update the drop highlighting.

After the drag and drop finishes, the drag source's `DoDragDrop` method returns the last type of action that was displayed when the user dropped the data. That lets the drag source know what the drop target expects the source to do with the data. For example, if the drop target accepted a Move, the drag source should remove the data from its control. If the drop target accepted a Copy, the drag source should not remove the data from its control.

The following table describes the two events that the drag source control receives to help it control the drop.

Event	Purpose
`GiveFeedback`	The drag has entered a valid drop target. The source can take action to indicate the type of drop allowed. For example, it might allow a Copy if the target is a `Label` and allow Move or Copy if the target is a `TextBox`.
`QueryContinueDrag`	The keyboard or mouse button state has changed. The drag source can decide whether to continue the drag, cancel the drag, or drop the data immediately.

The following sections describe some examples that demonstrate common drag-and-drop scenarios.

A Simple Example

The following code shows one of the simplest examples possible that contains both a drag source and a drop target. To build this example, start a new project and add two `Label` controls named `lblDragSource` and `lblDropTarget`.

Note that the `lblDropTarget` control must have its `AllowDrop` property set to `True` either at design time or at runtime or it will not receive any drag-and-drop events. When the user presses a mouse button down over the `lblDragSource` control, the `MouseDown` event handler calls that control's `DoDragDrop` method, passing it the text "Here's the drag data!" and indicating that it wants to perform a Copy. When the user drags the data over the `lblDropTarget` control, its `DragEnter` event handler executes. The event handler sets the routine's `e.Effect` value to indicate that the control will allow a Copy operation. If the user drops the data over the `lblDropTarget` control, its `DragDrop` event handler executes. This routine uses the `e.Data.GetData` method to get a text data value and displays it in a message box.

```
Public Class Form1
    ' Start a drag.
    Private Sub lblDragSource_MouseDown(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.MouseEventArgs) _
     Handles lblDragSource.MouseDown
        lblDragSource.DoDragDrop("Here's the drag data!", DragDropEffects.Copy)
    End Sub

    ' Make sure the drag is coming from lblDragSource.
    Private Sub lblDropTarget_DragEnter(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.DragEventArgs) _
     Handles lblDropTarget.DragEnter
        e.Effect = DragDropEffects.Copy
    End Sub

    ' Display the dropped data.
    Private Sub lblDropTarget_DragDrop(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.DragEventArgs) _
     Handles lblDropTarget.DragDrop
        MessageBox.Show(e.Data.GetData("Text").ToString)
    End Sub
End Class
```

As it is, this program lets you drag and drop data from the `lblDragSource` control to the `lblDropTarget` control. You can also drag data from the `lblDragSource` control into Word, WordPad, and any other application that can accept a drop of text data.

Similarly, the `lblDropTarget` control can act as a drop target for any application that provides drag sources. For example, if you open WordPad, enter some text, select it, and then click and drag it onto the `lblDropTarget` control, the application will display the text you dropped in a message box.

This example is a bit too simple to be really useful. If the drop target does nothing more, it should check the data it will receive and ensure that it is text. When you drag a file from Windows Explorer and drop it onto the `lblDropTarget` control, the `e.Data.GetData` method returns `Nothing` so the program cannot display its value. Because the program cannot display a file, it is misleading for the `lblDropTarget` control to display a Copy cursor when the user drags a file over it.

The following version of the `lblDropTarget_DragEnter` event handler uses the `e.Data .GetData?Present` method to see if the data being dragged has a textual format. If a text format is available, the control allows a Copy operation. If the data does not come in a textual form, the control doesn't allow a drop.

```
' Make sure the drag is coming from lblDragSource.
Private Sub lblDropTarget_DragEnter(ByVal sender As Object, _
 ByVal e As System.Windows.Forms.DragEventArgs) Handles lblDropTarget.DragEnter
    ' See if the drag data includes text.
    If e.Data.GetDataPresent("Text") Then
        e.Effect = DragDropEffects.Copy
    Else
        e.Effect = DragDropEffects.None
    End If
End Sub
```

Now, if you drag a file from Windows Explorer onto `lblDropTarget`, the control displays a no drop icon.

The `lblDropTarget_DragDrop` event handler doesn't need to change because Visual Basic doesn't raise the event if the control does not allow any drop operation. For example, if the user drags a file from Windows Explorer onto `lblDropTarget`, then that control's `DragEnter` event handler sets `e.Effect` to `DragDropEffects.None`, so Visual Basic doesn't raise the `DragDrop` event handler if the user drops the file there.

Moving Between ListBoxes

The following code provides a more complete example that demonstrates many of the drag-and-drop events. It allows the user to drag items between two `ListBox`es named `lstUnselected` and `lstSelected`. It also allows the user to drag an item to a specific position in its new `ListBox` and to drag an item from one position to another within a `ListBox`.

The code uses the variable `m_DragSource` to remember the control that started the drag. If this variable is `Nothing` during a drag, then the application knows that the drag was started by another application. In this example, the program ignores drags from other applications.

When the form loads, it sets the `AllowDrop` properties for its two `ListBox` controls. It then uses the `AddHandler` statement to give the two controls the same `MouseDown`, `DragOver`, `DragDrop`, and `DragLeave` event handlers.

When the user presses the mouse down on one of the `ListBox` controls, the `List_MouseDown` executes. If the user is pressing the right mouse button, the program casts the `sender` parameter into a `ListBox` to see which control raised the event. It uses the control's `IndexFromPoint` method to see which item is under the mouse and selects that item. If there is no item under the mouse, the routine exits. Next, the program saves a reference to the control in `m_DragSource` and calls the `DoDragDrop` method to start the drag for a Move operation. When `DoDragDrop` ends, the code sets `m_DragSource` to `Nothing` to indicate that the drag is over.

When the drag sits over one of the `ListBoxes`, the `List_DragOver` event handler executes. If `m_DragSource` is `Nothing`, then the drag event was caused by another application and the event handler exits without allowing the operation. If `m_DragSource` is not `Nothing`, the control allows a `Move` operation. It then uses the control's `IndexFromPoint` method to see which item is under the mouse and it selects that item.

If the user drags off of a `ListBox`, the `List_DragLeave` event handler unselects the item that is currently selected in that `ListBox`. This may help prevent some confusion about where the item will be dropped if the user releases the mouse button.

When the user drops over a `ListBox`, the `List_DragDrop` event handler executes. It uses the `e.Data.GetData` method to get the dropped text and calls subroutine `MoveItem` to move the item from the `m_DragSource` control to the drop target.

Subroutine `MoveItem` determines which item is selected in the drop target. If no item is selected, it adds the new item to the end of the target list. If an item is selected, the code inserts the new item in front of the selected item. In either case, the program selects the newly inserted item.

`MoveItem` then removes the original item from the drag source. If the drag source and the drop target are the same, then the code finds the first instance of the item being moved. If that is the same item that was just added, the code uses the source list's `RemoveAt` method to remove the next occurrence of the item. If the first item is not the one that was just added, the code removes it.

```
Public Class Form1
    ' Remember where we got it.
    Private m_DragSource As ListBox = Nothing

    ' Allow drag events.
    Private Sub Form1_Load(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles MyBase.Load
        lstUnselected.AllowDrop = True
        lstSelected.AllowDrop = True

        ' Add event handlers.
        AddHandler lstUnselected.MouseDown, AddressOf List_MouseDown
        AddHandler lstUnselected.DragOver, AddressOf List_DragOver
        AddHandler lstUnselected.DragDrop, AddressOf List_DragDrop
        AddHandler lstUnselected.DragLeave, AddressOf List_DragLeave

        AddHandler lstSelected.MouseDown, AddressOf List_MouseDown
        AddHandler lstSelected.DragOver, AddressOf List_DragOver
        AddHandler lstSelected.DragDrop, AddressOf List_DragDrop
        AddHandler lstSelected.DragLeave, AddressOf List_DragLeave
    End Sub

    ' Start a drag.
    Private Sub List_MouseDown(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.MouseEventArgs)
        ' Make sure this is the right button.
        If e.Button <> MouseButtons.Right Then Exit Sub

        ' Select the item at this point.
        Dim this_list As ListBox = DirectCast(sender, ListBox)
        this_list.SelectedIndex = this_list.IndexFromPoint(e.X, e.Y)
        If this_list.SelectedIndex < 0 Then Exit Sub

        ' Remember where the drag started.
        m_DragSource = this_list

        ' Start the drag.
        this_list.DoDragDrop( _
            this_list.SelectedItem.ToString, _
            DragDropEffects.Move)

        ' We're done dragging.
        m_DragSource = Nothing
    End Sub

    ' Highlight the item under the mouse.
    Private Sub List_DragOver(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.DragEventArgs)
        If m_DragSource Is Nothing Then Exit Sub
```

```
            e.Effect = DragDropEffects.Move
            Dim this_list As ListBox = DirectCast(sender, ListBox)
            Dim pt As Point = _
                this_list.PointToClient(New Point(e.X, e.Y))
            Dim drop_index As Integer = _
                this_list.IndexFromPoint(pt.X, pt.Y)
            this_list.SelectedIndex = drop_index
        End Sub

        ' Unhighlight the target item when the drag leaves.
        Private Sub List_DragLeave(ByVal sender As Object, ByVal e As System.EventArgs)
            Dim this_list As ListBox = DirectCast(sender, ListBox)
            this_list.SelectedIndex = -1
        End Sub

        ' Accept the drop.
        Private Sub List_DragDrop(ByVal sender As Object, _
         ByVal e As System.Windows.Forms.DragEventArgs)
            Dim this_list As ListBox = DirectCast(sender, ListBox)
            MoveItem(e.Data.GetData(DataFormats.Text).ToString, _
                m_DragSource, this_list, e.X, e.Y)
        End Sub

        ' Move the value txt from drag_source to drop_target.
        Private Sub MoveItem(ByVal txt As String, ByVal drag_source As ListBox, _
         ByVal drop_target As ListBox, ByVal X As Integer, ByVal Y As Integer)
            ' See which item is selected in the drop target.
            Dim drop_index As Integer = drop_target.SelectedIndex
            If drop_index < 0 Then
                ' Add at the end.
                drop_index = drop_target.Items.Add(txt)
            Else
                ' Add before the selected item.
                drop_target.Items.Insert(drop_target.SelectedIndex, txt)
            End If

            ' Select the item.
            drop_target.SelectedIndex = drop_index

            ' Remove the value from drag_source.
            If drag_source Is drop_target Then
                ' Make sure we don't remove the item we just added.
                Dim target_index As Integer = drag_source.FindStringExact(txt)
                If target_index = drop_index Then _
                    target_index = drag_source.FindStringExact(txt, target_index)
                drag_source.Items.RemoveAt(target_index)
            Else
                ' Remove the item.
                drag_source.Items.Remove(txt)
            End If
        End Sub
    End Class
```

This example shows how to drag items from one ListBox to another or to a new position within a single ListBox. The case is somewhat simpler if you don't need to worry about dragging items within a single ListBox. In that case, the MoveItem subroutine doesn't need to worry about removing the item you just added.

Moving and Copying Between ListBoxes

It isn't too difficult to modify the previous example to allow the user to move or copy items between the two lists. If the user holds down the Ctrl key while dropping an item, the program copies the item and leaves the original item where it started. If the user doesn't hold down the Ctrl key while dropping the item, the program moves it as before.

The code is almost the same as in the previous example. The first change is in the way the program starts the drag in the List_MouseDown event handler. In the call to DoDragDrop, the new code allows both Move and Copy operations.

```
' Start the drag.
this_list.DoDragDrop( _
    this_list.SelectedItem.ToString, _
    DragDropEffects.Move Or DragDropEffects.Copy)
```

Instead of always allowing the Move operation, the new DragOver event handler uses the following code to allow a Move or Copy, depending on whether the Ctrl key is pressed:

```
' Display the Move or Copy cursor.
Const KEY_CTRL As Integer = 8
If (e.KeyState And KEY_CTRL) <> 0 Then
    e.Effect = DragDropEffects.Copy
Else
    e.Effect = DragDropEffects.Move
End If
```

Remember that the DragOver event handler fires periodically as long as the drag sits over the control. This not only lets the program highlight the item beneath the mouse but also lets the code change the drag effect if the user presses and releases the Ctrl key while the drag is still in progress.

The new DragDrop event handler must determine whether the Ctrl key was pressed when the data was dropped. The e.Effect parameter indicates which drag-and-drop effect was displayed when the drop occurred. The new DragDrop compares e.Effect with DragDropEffects.Move and passes the MoveItem subroutine True if the effect was Move.

```
' Accept the drop.
Private Sub List_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs)
    Dim this_list As ListBox = DirectCast(sender, ListBox)
    MoveItem(e.Data.GetData(DataFormats.Text).ToString, _
        m_DragSource, this_list, e.X, e.Y, _
        e.Effect = DragDropEffects.Move)
End Sub
```

Finally, the `MoveItem` subroutine takes a new Boolean parameter `move_item`, which indicates whether it should move the item or copy it. It is similar to the previous version, except that it does not remove the item from its original list if `move_item` is `False`.

```
' Move the value txt from drag_source to drop_target.
Private Sub MoveItem(ByVal txt As String, ByVal drag_source As ListBox, _
 ByVal drop_target As ListBox, ByVal X As Integer, ByVal Y As Integer, _
 ByVal move_item As Boolean)
    ' See which item is selected in the drop target.
    Dim drop_index As Integer = drop_target.SelectedIndex
    If drop_index < 0 Then
        ' Add at the end.
        drop_index = drop_target.Items.Add(txt)
    Else
        ' Add before the selected item.
        drop_target.Items.Insert(drop_target.SelectedIndex, txt)
    End If

    ' Select the item.
    drop_target.SelectedIndex = drop_index

    ' See if we are moving or copying.
    If move_item Then
        ' Remove the value from drag_source.
        If drag_source Is drop_target Then
            ' Make sure we don't remove the item we just added.
            Dim target_index As Integer = drag_source.FindStringExact(txt)
            If target_index = drop_index Then _
                target_index = drag_source.FindStringExact(txt, target_index)
            drag_source.Items.RemoveAt(target_index)
        Else
            ' Remove the item.
            drag_source.Items.Remove(txt)
        End If
    End If
End Sub
```

Learning Data Types Available

When the user drags data over a drop target, the target's `DragEnter` event handler decides which kinds of drop to allow. The event handler can use the `e.GetDataPresent` method to see whether the data is available in a desired data format.

`GetDataPresent` takes as a parameter a string giving the desired data type. An optional second parameter indicates whether the program will accept another format if the system can derive it from the original format. For example, the system can convert Text data into System.String data so you can decide whether to allow the system to make this conversion.

The `DataFormats` class provides a shared series of standardized string values specifying various data types. For example, `DataFormats.Text` returns the string `Text` representing the text data type.

If you use a DataFormats value, you don't need to worry about misspelling one of these formats. Some of the most commonly used DataFormats include Bitmap, Html, StringFormat, and Text. See the online help for other formats. The web page http://msdn.microsoft.com/library/en-us/cpref/html/frlrfsystemwindowsformsdataformatsmemberstopic.asp lists the DataFormats class's supported formats.

GetDataPresent can also take as a parameter a data type. For example, the following code fragment uses GetDataPresent to allow a Copy operation if the drag data contains an Employee object.

```
If e.Data.GetDataPresent(GetType(Employee)) Then
    ' Allow Copy.
    e.Effect = DragDropEffects.Copy
Else
    ' Allow no other drops.
    e.Effect = DragDropEffects.None
End If
```

In addition to GetDataPresent, you can use the e.Data.GetFormats method to get an array of strings giving the names of the available formats. The following code shows how a program can list the formats available. It clears its lstWithoutConversion ListBox and then loops through the values returned by e.Data.GetFormats, adding them to the ListBox. It passes GetFormats the parameter False to indicate that it should return only data formats that are directly available, not those that can be derived from others. The program then repeats these steps, this time passing GetFormats the parameter True to include derived formats.

```
Private Sub lblDropTarget_DragEnter(ByVal sender As Object, _
 ByVal e As System.Windows.Forms.DragEventArgs) Handles lblDropTarget.DragEnter
    lstWithoutConversion.Items.Clear()
    For Each fmt As String In e.Data.GetFormats(False)
        lstWithoutConversion.Items.Add(fmt)
    Next fmt

    lstWithConversion.Items.Clear()
    For Each fmt As String In e.Data.GetFormats(True)
        lstWithConversion.Items.Add(fmt)
    Next fmt
End Sub
```

Dragging Within an Application

Sometimes, you may want a drop target to accept only data dragged from within the same application. The following code shows one way to handle this. Before it calls DoDragDrop, the program sets its m_Dragging variable to True. The lblDropTarget control's DragEnter event checks m_Dragging. If the user drags data from a program other than this one, m_Dragging will be False and the program sets e.Effect to DragDropEffects.None, prohibiting a drop. If m_Dragging is True, that means this program started the drag, so the program allows a Copy operation. After the drag and drop finishes, the lblDragSource control's MouseDown event handler sets m_Dragging to False, so the drop target will refuse future drags from other applications.

```
Public Class Form1
    ' True while we are dragging.
    Private m_Dragging As Boolean
    ' Start a drag.
    Private Sub lblDragSource_MouseDown(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.MouseEventArgs) _
     Handles lblDragSource.MouseDown
        m_Dragging = True
        lblDragSource.DoDragDrop("Some text", DragDropEffects.Copy)
        m_Dragging = False
    End Sub

    ' Only allow Copy if we are dragging.
    Private Sub lblDropTarget_DragEnter(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.DragEventArgs) Handles lblDropTarget.DragEnter
        If m_Dragging Then
            e.Effect = DragDropEffects.Copy
        Else
            e.Effect = DragDropEffects.None
        End If
    End Sub

    ' Display the dropped text.
    Private Sub lblDropTarget_DragDrop(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.DragEventArgs) Handles lblDropTarget.DragDrop
        MessageBox.Show(e.Data.GetData(DataFormats.Text).ToString)
    End Sub
End Class
```

There is no easy way to allow your program to drag data to its own controls, but not allow it to drag data to another program. The philosophy is that a drag source provides data for any application that can handle it.

If you don't want other applications to read data dragged from your application, you can package the data in an object and drag the object as described in the section "Dragging Objects" later in this chapter. This will make it very difficult for most applications to understand the data, even if they try to accept it.

Accepting Dropped Files

Many applications let you drop files onto them. When you drag files over a drop target, the data object contains data of several types, including `FileDrop`. This data is an array of strings containing the names of the files being dragged.

The following code shows how a program might process files dragged onto it. The `lblDropTarget` control's `DragEnter` event handler uses the `GetDataPresent` method to see if the drag contains `FileDrop` data, and allows the Copy operation if it does. The control's `DragDrop` event handler uses `GetData` to get the data in `FileDrop` format. It converts the data from a generic object into an array of strings, and then loops through the entries, adding each to the `lstFiles` `ListBox`.

```
Public Class Form1
    ' Allow Copy if there is FileDrop data.
    Private Sub lblDropTarget_DragEnter(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.DragEventArgs) Handles lblDropTarget.DragEnter
        If e.Data.GetDataPresent(DataFormats.FileDrop) Then
            e.Effect = DragDropEffects.Copy
        Else
```

```
            e.Effect = DragDropEffects.None
        End If
    End Sub

    ' Display the dropped file names.
    Private Sub lblDropTarget_DragDrop(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.DragEventArgs) Handles lblDropTarget.DragDrop
        lstFiles.Items.Clear()
        Dim file_names As String() = _
            DirectCast(e.Data.GetData(DataFormats.FileDrop), String())
        For Each file_name As String In file_names
            lstFiles.Items.Add(file_name)
        Next file_name
    End Sub
End Class
```

A more realistic application would do something more useful than simply listing the files. For example, it might delete them, move them into the wastebasket, copy them to a backup directory, display thumbnails of image files, and so forth.

Dragging Objects

Dragging text is simple enough. Simply pass the text into the `DoDragDrop` method and you're finished.

You can drag an arbitrary object in a similar manner, as long as the drag source and drop target are within the same application. If you want to drag objects between applications, however, you must use serializable objects. A *serializable* object is one that provides methods for translating the object into and out of a stream-like format. Usually, this format is text, and lately XML is the preferred method for storing text streams.

For example, consider the following `Employee` class:

```
Public Class Employee
    Public FirstName As String
    Public LastName As String

    Public Sub New()
    End Sub

    Public Sub New(ByVal first_name As String, ByVal last_name As String)
        FirstName = first_name
        LastName = last_name
    End Sub
End Class
```

You could serialize an `Employee` object having `FirstName` = "Rod" and `LastName` = "Stephens" with the following XML text:

```
<Employee>
  <FirstName>Rod</FirstName>
  <LastName>Stephens</LastName>
</Employee>
```

You can use drag and drop to move a serializable object between applications. The drag source converts the object into its serialization and sends the resulting text to the drop target. The drop target uses the serialization to recreate the object.

You might think it would be hard to make an object serializable. Fortunately, Visual Basic .NET provides many features for automatically discovering the structure of objects, so it can do most of the work for you. In most cases, all you need to do is add the `Serializable` attribute to the class, as shown in the following code:

```
<Serializable()> _
Public Class Employee
    Public FirstName As String
    Public LastName As String

    Public Sub New()
    End Sub

    Public Sub New(ByVal first_name As String, ByVal last_name As String)
        FirstName = first_name
        LastName = last_name
    End Sub
End Class
```

The drag source can pass objects of this type to the `DoDragDrop` method.

The following code shows how an application can act as a drag source and a drop target for objects of the `Employee` class. It starts by defining the `Employee` class. It also defines the constant `DATA_EMPLOYEE`. This value, `DragEmployee.frmDragEmployee+Employee`, is the name of the data format type assigned to the `Employee` class. This name combines the project name, the module name where the class is defined, and the class name.

When the user presses the mouse down over the `lblDragSource` control, its `MouseDown` event handler creates an `Employee` object, initializing it with the values contained in the `txtFirstName` and `txtLastName` text boxes. It then calls the `lblDragSource` control's `DoDragDrop` method, passing it the `Employee` object and allowing the Move and Copy operations. If `DoDragDrop` returns the value `Move`, the user performed a Move rather than a Copy, so the program removes the values from its text boxes.

When the user drags over the `lblDropTarget` control, its `DragOver` event handler executes. The routine first uses the `GetDataPresent` method to verify that the dragged data contains an `Employee` object. It then checks the Ctrl key's state. If the user is holding down the Ctrl key, then the event handler allows the Copy operation. If the user is not holding down the Ctrl key, the subroutine allows the Move operation.

If the user drops the data on the `lblDropTarget` control, its `DragDrop` event handler executes. It uses the `GetData` method to retrieve the `Employee` object. `GetData` returns a generic `Object`, so the program uses `DirectCast` to convert the result into an `Employee` object. The event handler finishes by displaying the object's `FirstName` and `LastName` properties in its text boxes.

```
Imports System.IO
Imports System.Xml.Serialization
```

```
Public Class frmDragEmployee
    Public Const DATA_EMPLOYEE As String = "DragEmployee.frmDragEmployee+Employee"
    <Serializable()> _
    Public Class Employee
        Public FirstName As String
        Public LastName As String
        Public Sub New()
        End Sub
        Public Sub New(ByVal first_name As String, ByVal last_name As String)
            FirstName = first_name
            LastName = last_name
        End Sub
    End Class

    ' Start dragging the Employee.
    Private Sub lblDragSource_MouseDown(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.MouseEventArgs) _
     Handles lblDragSource.MouseDown
        Dim emp As New Employee(txtFirstName.Text, txtLastName.Text)

        If lblDragSource.DoDragDrop(emp, _
            DragDropEffects.Copy Or DragDropEffects.Move) = DragDropEffects.Move _
        Then
            ' A Move succeeded. Clear the TextBoxes.
            txtFirstName.Text = ""
            txtLastName.Text = ""
        End If
    End Sub

    ' If an Employee object is available, allow a Move
    ' or Copy depending on whether the Ctrl key is pressed.
    Private Sub lblDropTarget_DragOver(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.DragEventArgs) _
     Handles lblDropTarget.DragOver
        If e.Data.GetDataPresent(DATA_EMPLOYEE) Then
            ' Display the Move or Copy cursor.
            Const KEY_CTRL As Integer = 8
            If (e.KeyState And KEY_CTRL) <> 0 Then
                e.Effect = DragDropEffects.Copy
            Else
                e.Effect = DragDropEffects.Move
            End If
        End If
    End Sub

    ' Display the dropped Employee object.
    Private Sub lblDropTarget_DragDrop(ByVal sender As Object, _
     ByVal e As System.Windows.Forms.DragEventArgs) _
     Handles lblDropTarget.DragDrop
        Dim emp As Employee = DirectCast(e.Data.GetData(DATA_EMPLOYEE), Employee)
        lblFirstName.Text = emp.FirstName
        lblLastName.Text = emp.LastName
    End Sub
End Class
```

If you compile this program, you can run two copies of the executable program stored in the bin directory and drag from the drag source in one to the drop target in the other.

If you remove the Serializable attribute from the Employees class, the program still works if you drag from the drag source to the drop target within the same instance of the application. If you run two instances and drag from one to the other, however, the drop target gets the value Nothing from the GetData method, so the drag and drop fails.

Changing Format Names

The previous example dragged data with the rather unwieldy data format name DragEmployee .frmDragEmployee+Employee. This name identifies the class reasonably well, so it is unlikely that another application will try to load this data if it has some other definition for the Employee class.

On the other hand, the name is rather awkward. It is also problematic if you want to drag objects between two different applications, because each will use its project and module name to define the data format type. If you want to drag Employee objects between the TimeSheet program and the EmployeePayroll program, the names of the data formats generated by the two programs won't match.

The DataObject class provides more control over how the data is represented. Instead of dragging an Employee object directly, you create a DataObject, store the Employee object inside it with the data format name of your choosing, and then drag the DataObject.

The following code fragment shows this technique. It creates an Employee object as before and then creates a DataObject. It calls the DataObject object's SetData method, passing it the Employee object and the data format name.

```
Dim emp As New Employee(txtFirstName.Text, txtLastName.Text)
Dim data_object As New DataObject()
data_object.SetData("Employee", emp)

If lblDragSource.DoDragDrop(data_object, _
    DragDropEffects.Copy Or DragDropEffects.Move) = DragDropEffects.Move _
Then
    ' A Move succeeded. Clear the TextBoxes.
    txtFirstName.Text = ""
    txtLastName.Text = ""
End If
```

In general, you should try to avoid very generic names such as Employee for data types. Using such a simple name increases the chances that another application will use the same name for a different class. Another program will not be able to convert your Employee data into a different type of Employee class.

To ensure consistency across applications, you must define a naming convention that can identify objects across projects. To ensure that different applications use exactly the same object definitions, you might also want to define the objects in a separate DLL used by all of the applications. That simplifies the naming problem, because you can use the DLL's name as part of the object's name.

For example, suppose that you build an assortment of billing database objects such as Employee, Customer, Order, OrderItem, and so on. If the objects are defined in the module BillingObjects.dll, you could give the objects names such as BillingObjects.Employee, BillingObjects.Customer, and so on.

Dragging Multiple Data Formats

The DataObject not only allows you to pick the data form name used by a drag; it also allows you to associate more than one piece of data with a drag. To do this, the program simply calls the object's SetData method more than once, passing it data in different formats.

The following code shows how a program can drag the text in a RichTextBox control in three data formats: RTF, plain text, and HTML. The lblDragSource control's MouseDown event handler makes a DataObject and calls its SetData method, passing it the rchSource control's contents in the Rtf and Text formats. It then builds an HTML string and passes that to the SetData method as well.

```
' Start a drag.
Private Sub lblDragSource_MouseDown(ByVal sender As Object, _
 ByVal e As System.Windows.Forms.MouseEventArgs) Handles lblDragSource.MouseDown
    ' Make a DataObject.
    Dim data_object As New DataObject

    ' Add the data in various formats.
    data_object.SetData(DataFormats.Rtf, rchSource.Rtf)
    data_object.SetData(DataFormats.Text, rchSource.Text)

    ' Build the HTML version.
    Dim html_text As String
    html_text = "<HTML>" & vbCrLf
    html_text &= "  <HEAD>The Quick Brown Fox</HEAD>" & vbCrLf
    html_text &= "  <BODY>" & vbCrLf
    html_text &= rchSource.Text & vbCrLf
    html_text &= "  </BODY>" & vbCrLf & "</HTML>"
    data_object.SetData(DataFormats.Html, html_text)

    ' Start the drag.
    lblDragSource.DoDragDrop(data_object, DragDropEffects.Copy)
End Sub
```

The following code shows the lblDropTarget control's DragEnter event handler. If the data includes the RTF, Text, or HTML data formats, the control allows a Copy operation.

```
' Allow drop of Rtf, Text, and HTML.
Private Sub lblDropTarget_DragEnter(ByVal sender As Object, _
 ByVal e As System.Windows.Forms.DragEventArgs) Handles lblDropTarget.DragEnter
    If e.Data.GetDataPresent(DataFormats.Rtf) Or _
       e.Data.GetDataPresent(DataFormats.Text) Or _
       e.Data.GetDataPresent(DataFormats.Html) _
    Then
        e.Effect = DragDropEffects.Copy
    End If
End Sub
```

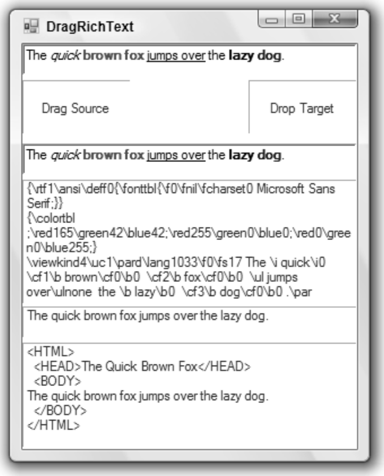
The following code shows how a program can read these formats. If the dropped data includes the Rtf format, the code displays it in the RichTextControl rchTarget. It also displays the RTF data in the lblRtf Label. This lets you see the Rich Text codes. If the data includes the Text format, the program displays it in the `lblTarget` label. Finally, if the data includes HTML, the program displays it in the `lblHtml` label.

```
' Display whatever data we can.
Private Sub lblDropTarget_DragDrop(ByVal sender As Object, _
 ByVal e As System.Windows.Forms.DragEventArgs) Handles lblDropTarget.DragDrop
    If e.Data.GetDataPresent(DataFormats.Rtf) Then
        rchTarget.Rtf = e.Data.GetData(DataFormats.Rtf).ToString
        lblRtf.Text = e.Data.GetData(DataFormats.Rtf).ToString
    Else
        rchTarget.Text = ""
        lblRtf.Text = ""
    End If

    If e.Data.GetDataPresent(DataFormats.Text) Then
        lblTarget.Text = e.Data.GetData(DataFormats.Text).ToString
    Else
        lblTarget.Text = ""
    End If

    If e.Data.GetDataPresent(DataFormats.Html) Then
        lblHtml.Text = e.Data.GetData(DataFormats.Html).ToString
    Else
        lblHtml.Text = ""
    End If
End Sub
```

Figure 13-1 shows the results. The `RichTextBox` on the top shows the original data in `rchSource`. Below the drag source and drop target labels, other controls show the dropped results. The first control is a `RichTextBox` that shows the `Rtf` data. The second control is a label displaying the Rich Text codes. The third control is a label showing the `Text` data, and the final control is a label showing the `Html` data.



The *quick* **brown fox** <u>jumps over</u> the **lazy dog**.

The *quick* **brown fox** <u>jumps over</u> the **lazy dog**.

```
{\rtf1\ansi\deff0{\fonttbl{\f0\fnil\fcharset0 Microsoft Sans
Serif;}}
{\colortbl
;\red165\green42\blue42;\red255\green0\blue0;\red0\gree
n0\blue255;}
\viewkind4\uc1\pard\lang1033\f0\fs17 The \i quick\i0
\cf1\b brown\cf0\b0  \cf2\b fox\cf0\b0  \ul jumps
over\ulnone  the \b lazy\b0  \cf3\b dog\cf0\b0 .\par
```

The quick brown fox jumps over the lazy dog.

```
<HTML>
  <HEAD>The Quick Brown Fox</HEAD>
  <BODY>
The quick brown fox jumps over the lazy dog.
  </BODY>
</HTML>
```

Figure 13-1: This program drags and drops data in Text, RTF, and HTML formats.

If you drag data from another application onto the drop target, this program displays only the data that is available. For example, if you drag data from WordPad, this program will display only `Rtf` and `Text` data, because those are the only compatible formats provided by WordPad.

Using the Clipboard

Using the clipboard is very similar to using drag and drop. To save a single piece of data, call the `Clipboard` object's `SetDataObject` method, passing it the data that you want to save. For example, the following code copies the text in the `txtLastName` control to the clipboard:

```
Clipboard.SetDataObject(txtLastName.Text)
```

Copying data to the clipboard in multiple formats is very similar to dragging and dropping multiple data formats. First, create a `DataObject` and use its `SetData` method to store the data exactly as before. Then call the `Clipboard` object's `SetDataObject` method, passing it the `DataObject`.

The following code adds `Rtf`, `Text`, and `Html` data to the clipboard:

```
' Copy data to the clipboard.
Private Sub btnCopy_Click(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnCopy.Click
    ' Make a DataObject.
    Dim data_object As New DataObject

    ' Add the data in various formats.
    data_object.SetData(DataFormats.Rtf, rchSource.Rtf)
    data_object.SetData(DataFormats.Text, rchSource.Text)

    ' Build the HTML version.
    Dim html_text As String
    html_text = "<HTML>" & vbCrLf
    html_text &= "  <HEAD>The Quick Brown Fox</HEAD>" & vbCrLf
    html_text &= "  <BODY>" & vbCrLf
    html_text &= rchSource.Text & vbCrLf
    html_text &= "  </BODY>" & vbCrLf & "</HTML>"
    data_object.SetData(DataFormats.Html, html_text)

    ' Copy data to the clipboard.
    Clipboard.SetDataObject(data_object)
End Sub
```

To retrieve data from the clipboard, use the `GetDataObject` method to get an `IDataObject` representing the data. Use that object's `GetDataPresent` method to see if a data type is present, and use its `GetData` method to get data with a particular format.

The following code displays `Rtf`, `Text`, and `Html` data from the clipboard:

```
' Paste data from the clipboard.
Private Sub btnPaste_Click(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnPaste.Click
    Dim data_object As IDataObject = Clipboard.GetDataObject()

    If data_object.GetDataPresent(DataFormats.Rtf) Then
        rchTarget.Rtf = data_object.GetData(DataFormats.Rtf).ToString
        lblRtf.Text = data_object.GetData(DataFormats.Rtf).ToString
    Else
        rchTarget.Text = ""
        lblRtf.Text = ""
    End If

    If data_object.GetDataPresent(DataFormats.Text) Then
        lblTarget.Text = data_object.GetData(DataFormats.Text).ToString
    Else
        lblTarget.Text = ""
    End If

    If data_object.GetDataPresent(DataFormats.Html) Then
        lblHtml.Text = data_object.GetData(DataFormats.Html).ToString
    Else
        lblHtml.Text = ""
    End If
End Sub
```

The `IDataObject` returned by the `GetDataObject` method also provides a `GetFormats` method that returns an array of the data formats available. This array is very similar to the one returned by the `GetFormats` method provided by the `DragEnter` event described earlier in this chapter.

You can copy and paste objects using the clipboard much as you drag and drop objects. Simply make the object's class serializable and add an instance of the class to the `DataObject`.

The following code shows how a program can copy and paste an `Employee` object. The `btnCopy_Click` event handler makes an `Employee` object and a `DataObject`. It passes the `Employee` object to the `DataObject` object's `SetData` method, giving it the data format name Employee. The program then passes the `DataObject` to the `Clipboard` object's `SetDataObject` method. The `btnPaste_Click` event handler retrieves the clipboard's data object and uses its `GetDataPresent` method to see if the clipboard is holding data with the `Employee` format. If the data is present, the program uses the data object's `GetData` method to fetch the data, casts it into an `Employee` object, and displays the object's property values.

```
' Copy the Employee to the clipboard.
Private Sub btnCopy_Click(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnCopy.Click
    Dim emp As New Employee(txtFirstName.Text, txtLastName.Text)
    Dim data_object As New DataObject
    data_object.SetData("Employee", emp)
    Clipboard.SetDataObject(data_object)
End Sub

' Paste data from the clipboard.
Private Sub btnPaste_Click(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnPaste.Click
```

```
        Dim data_object As IDataObject = Clipboard.GetDataObject()
        If data_object.GetDataPresent("Employee") Then
            Dim emp As Employee = _
                DirectCast(data_object.GetData("Employee"), Employee)
            txtPasteFirstName.Text = emp.FirstName
            txtPasteLastName.Text = emp.LastName
        End If
    End Sub
```

The following table lists most of the methods provided by the `Clipboard` object, including several that make working with common data types easier.

Method	Purpose
`Clear`	Removes all data from the clipboard.
`ContainsAudio`	Returns `True` if the clipboard contains audio data.
`ContainsData`	Returns `True` if the clipboard contains data in a particular format.
`ContainsFileDropList`	Returns `True` if the clipboard contains a file drop list.
`ContainsImage`	Returns `True` if the clipboard contains an image.
`ContainsText`	Returns `True` if the clipboard contains text.
`GetAudioStream`	Returns the audio stream contained in the clipboard.
`GetData`	Returns data in a specific format.
`GetDataObject`	Returns the clipboard's `DataObject`.
`GetFileDropList`	Returns the file drop list contained in the clipboard.
`GetImage`	Returns the image contained in the clipboard.
`GetText`	Returns the text contained in the clipboard.
`SetAudio`	Saves audio bytes or an audio stream in the clipboard.
`SetData`	Saves data in a particular format in the clipboard.
`SetDataObject`	Saves the data defined by a `DataObject` in the clipboard.
`SetFileDropList`	Saves a file drop list in the clipboard. The data should be a `StringCollection` containing the file names.
`SetImage`	Saves an image in the clipboard.
`SetText`	Saves text in the clipboard.

The following code retrieves file drop list data from the clipboard:

```
Private Sub btnPaste_Click(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnPaste.Click
    lstFiles.Items.Clear()
    If Clipboard.ContainsFileDropList() Then
        Dim file_names As StringCollection = Clipboard.GetFileDropList()
        For Each file_name As String In file_names
            lstFiles.Items.Add(file_name)
        Next file_name
    End If
End Sub
```

Summary

Drag and drop events and the clipboard both move data from a source to a destination. The source and destination can be in the same program or in two different applications.

The clipboard lets a program save and retrieve data in a central, shared location. Data copied to the clipboard may remain in the clipboard for a long time so that the user can paste it into another application later.

Drag and drop support lets the user directly copy or move date immediately. Once the operation is complete, the data is not left lying around as it is in the clipboard. A user who wants to copy the data again later must perform a new drag-and-drop operation.

Providing drag-and-drop support with appropriate feedback is more work than using the clipboard, but it provides the user with more direct control, and it doesn't replace whatever data currently sits in the clipboard.

Together, these two tools let you provide the user with more control over the application's data. They let the user move data between different parts of an application and between different applications. Although drag-and-drop support and the clipboard are usually not the main purpose of an application, they can add to the user's experience an extra dimension of hands-on control.

The chapters in the book so far have focused on specific Visual Basic programming details. They explained the Visual Basic development environment, language syntax, standard controls and forms, custom controls, drag and drop, and the clipboard.

Chapter 14 examines applications at a slightly higher level in the context of the operating system. It explains the new User Account Control (UAC) security system provided by the Vista operating system and tells how UAC can prevent your application from running properly. Unless you understand how UAC works and how to interact with it, the operating system may not allow your program to perform the tasks that it should.