More Mouse

Mouse Buttons

- Some mice have more than two buttons.
- The older Microsoft Intellimouse had five buttons.
- Trackballs were also popular and behave like a normal mouse.
- Touchpads act like a mouse on laptops.
- The *MouseButtons enumeration is used to determine the button that was clicked.*

MouseButtons Enumeration

Member	*Description*
Left	The left mouse button was pressed.
Middle	The middle mouse button was pressed.
None	No mouse button was pressed.
Right	The right mouse button was pressed.
XButton1	IntelliMouse
XButton2	IntelliMouse

Mouse Button Events

- The following events are generated in the given order:

1. MouseDown
2. Click
3. MouseClick
4. MouseUp

Example Program

```
MouseClick1 - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace MouseClick1
{
    public partial class Form1 : Form
```

Example Program

```
    {
        private string s;
        public Form1()
        {
            InitializeComponent();
        }
        protected override void OnPaint(PaintEventArgs e)
        {
            e.Graphics.DrawString(s, Font, Brushes.Black,
                10, 10);
        }
        private void Form1_MouseDown(object sender,
                MouseEventArgs e)
        {
```

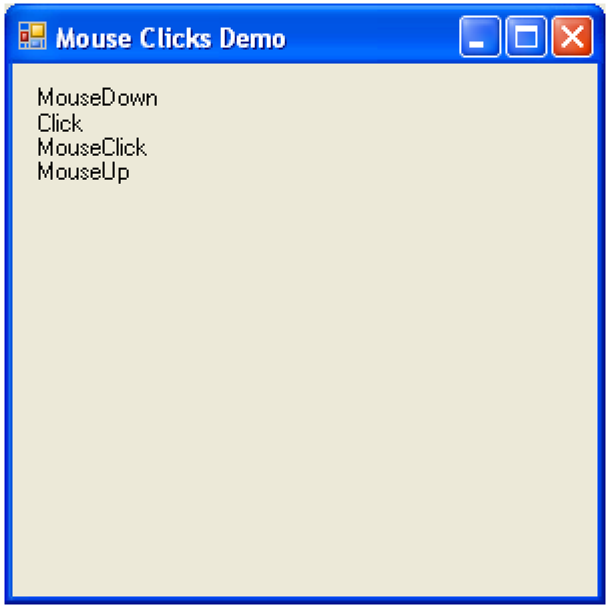
Example Program

```
            s += "MouseDown\n";
            Invalidate();
        }
        private void Form1_MouseClick(object sender,
            MouseEventArgs e)
        {
            s += "MouseClick\n";
            Invalidate();
        }
        private void Form1_MouseUp(object sender,
            MouseEventArgs e)
        {
            s += "MouseUp\n";
            Invalidate();
```

Example Program

```
        }
        private void Form1_Click(object sender, EventArgs 
e)
        {
            s += "Click\n";
            Invalidate();
        }
        private void Form1_MouseDoubleClick(object sender,
             MouseEventArgs e)
        {
            s += "MouseDoubleClick\n";
        }
    }
}
```

The Output

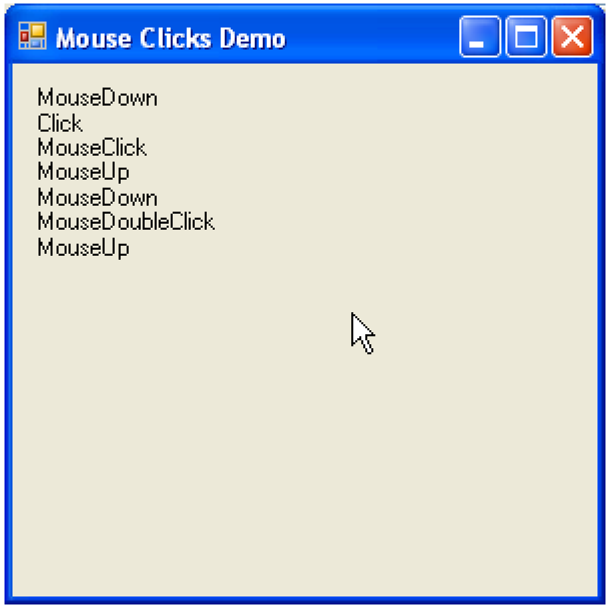


```
MouseDown
Click
MouseClick
MouseUp
```

Double Clicks

- Double clicking the mouse generates two pairs of *MouseUp/Down* events, a *Click* event, a *MouseClick* event and a *MouseDoubleClick* event.
- The exact sequence is shown on the next slide.
- The user can control the timing required to detect a double click through the control panel's mouse dialog. A user can also switch the left and right mouse buttons. This switch is invisible to programs. In other words, if the buttons are switched a left click generates events with the *MouseButtons* enumeration indicating the right button.

Double Click Output



```
MouseDown
Click
MouseClick
MouseUp
MouseDown
MouseDoubleClick
MouseUp
```

The Click Event

- The *click* event is not strictly a mouse event as it can also be triggered by other actions such as pressing the *enter* key when the associated control has focus.
- This action is only for some controls and even though your form is a control the *click* event is not generated when you hit *enter*.

Hit Testing

- A very common task is to select an object by use of a mouse click.
- This can be easy or very difficult depending n what type of objects are drawn in the form's clients area.
- The coordinates of all mouse clicks are pixels relative to the upper left hand corner of the client area.
- If we use a transform then this coordinate may not be the same coordinate in the logical drawing space.
- We can translate the mouse click's coordinate to and from the logical coordinate very easily.
- The *transformPoints* method is a general method for translating between coordinate systems.

Tic-Tac-Toe Example

- To demonstrate all this I will present a user interface for the game of tic-tac-toe.
- The left mouse button draws an O.
- The right mouse button draws an X.
- The middle button clears a cell.
- The programs display is configurable by the use of a number of constants.
- The form's contents resizes when the form is resized.

Transforming the Mouse Coordinate

- To transform a coordinate in device units to world units the following statements are used:

```
PointF[] p = { new Point(e.X, e.Y) };
g.TransformPoints(CoordinateSpace.World,
  CoordinateSpace.Device, p);
```

- The *TransformPoints* method requires an array of points rather than a single point. The first statement sets up such an array with just one point in it.
- The result of the call to *TransformPoints* is that the resulting point in the array now corresponds to the position of the mouse click in the world coordinate system. All we need to do now is to determine which cell we are clicking in.

Determining the Cell

- These statements determine the cell we are clicking in:

```
int i = (int)(p[0].X/block);
int j = (int)(p[0].Y/block);
```

- Block is the size of each cell.
- We ignore mouse coordinates that are negative and i,j values that are out of range.

Cell Selection Enumeration

Member	*Description*
N	Empty cell, nothing displayed
O	An O is displayed
X	An X is diaplayed

Program Constants

Constant	*Determines*
clientSize	The size of the world coordinate space
lineLength	The length of the grid lines
block	The width and height of a cell (lineLength/3)
offset	The position of the upper left hand corner of the playing board
delta	The inside border of the X or O

Tic-Tac-Toe Example

```
HitTest1 - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Text;
using System.Windows.Forms;
namespace HitTest1
{
    public partial class Form1 : Form
    {
```

Tic-Tac-Toe Example

```
//dimensions
private const float clientSize = 100;
private const float lineLength = 80;
private const float block = lineLength / 3;
private const float offset = 10;
private const float delta = 5;
private enum CellSelection { N, O, X };
private CellSelection[,] grid=new
   CellSelection[3,3];
private float scale;    //current scale factor
public Form1()
{
    InitializeComponent();
```

Tic-Tac-Toe Example

```
            ResizeRedraw = true;
        }
        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            ApplyTransform(g);
            //draw board
            g.DrawLine(Pens.Black, block, 0, block,
                lineLength);
            g.DrawLine(Pens.Black,2*block, 0, 2*block,
                lineLength);
            g.DrawLine(Pens.Black, 0, block, lineLength,
                block);
```

Tic-Tac-Toe Example

```
          g.DrawLine(Pens.Black, 0, 2*block, lineLength,
              2*block);
          for (int i = 0; i < 3; ++i)
              for (int j = 0; j < 3; ++j)
                  if (grid[i, j] == CellSelection.O)
                     DrawO(i, j, g);
                  else if (grid[i, j] ==
                     CellSelection.X) DrawX(i, j, g);
      }
      private void ApplyTransform(Graphics g)
      {
          scale = Math.Min(ClientRectangle.Width /
              clientSize,
```

Tic-Tac-Toe Example

```
        ClientRectangle.Height / clientSize);
        if (scale == 0f) return;
        g.ScaleTransform(scale, scale);
        g.TranslateTransform(offset, offset);
    }
    private void DrawX(int i, int j, Graphics g)
    {
        g.DrawLine(Pens.Black, i*block+delta,
              j*block+delta,
            (i*block)+block-delta, (j*block)+block-
                  delta);
        g.DrawLine(Pens.Black, (i*block)+block-delta,
            j*block+delta, (i*block)+delta,
                (j*block)+block-delta);
```

Tic-Tac-Toe Example

```
}
        private void DrawO(int i, int j, Graphics g)
        {
            g.DrawEllipse(Pens.Black, i*block+delta,
                    j*block+delta,
                block-2*delta, block-2*delta);
        }
        private void Form1_MouseDown(object sender,
                     MouseEventArgs e)
        {
            Graphics g = CreateGraphics();
            ApplyTransform(g);
            PointF[] p = { new Point(e.X, e.Y) };
```

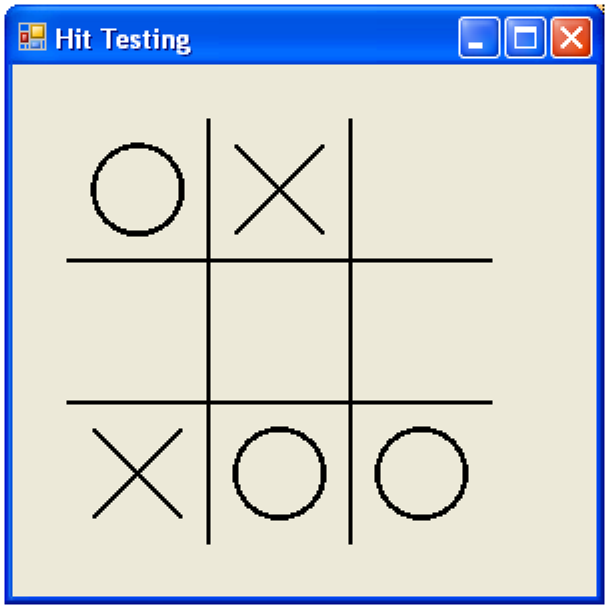
Tic-Tac-Toe Example

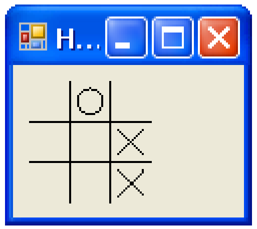
```
g.TransformPoints(CoordinateSpace.World,
    CoordinateSpace.Device, p);
if (p[0].X < 0 || p[0].Y < 0) return;
int i = (int)(p[0].X/block);
int j = (int)(p[0].Y/block);
if (i > 2 || j > 2) return;
if (e.Button == MouseButtons.Middle)
    grid[i,j] = CellSelection.N;
//only allow setting empty cells
if (grid[i, j] == CellSelection.N)
{
    if (e.Button == MouseButtons.Left)
        grid[i, j] = CellSelection.O;
```

Tic-Tac-Toe Example

```
                  if (e.Button == MouseButtons.Right)
                       grid[i, j] = CellSelection.X;
              }
              Invalidate();
          }
      }
}
```

The Output





The MouseMove Event

- When you move the mouse the Windows operating system generates an event.
- The spacing between the coordinates reported by the *MouseMove* event depends on how fast you move the mouse and the speed of your system.
- You can't turn off mouse movement events, but you can just ignore them.
- The following example shows how this works by displaying the mouse coordinate continuously as the mouse is moved.

MouseMove Example

```
 MouseMove1 - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace MouseMove1
{
    public partial class Form1 : Form
    {
```

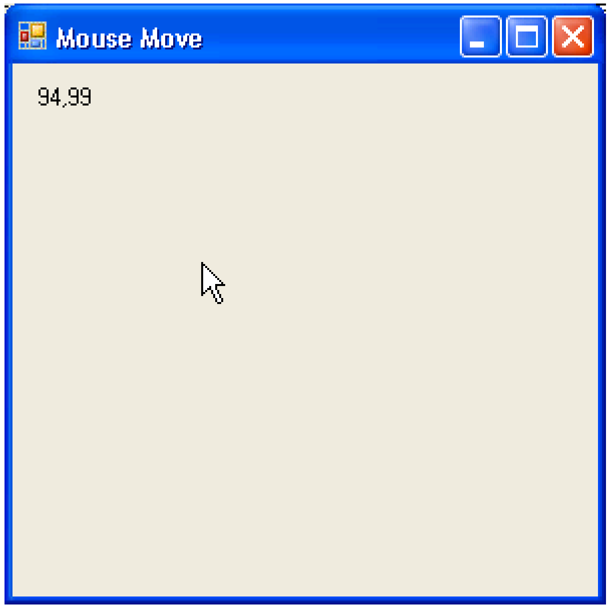
MouseMove Example

```
public Form1()
       {
           InitializeComponent();
       }
       protected override void OnPaint(PaintEventArgs e)
       {
           Graphics g = e.Graphics;
           g.DrawString(s, Font, Brushes.Black, 10, 10);
       }
       protected override void OnMouseMove(MouseEventArgs
           e)
       {
          s = e.X.ToString() + "," + e.Y;
```

MouseMove Example

```
 Invalidate();
         }
         private string s;
    }
}
```

Result



```
94,99
```

Dragging an Object

- A common task in graphical applications is to drag a graphic object around the client area.
- This is not as difficult as you might assume.
- The first task is to identify the object to be dragged using hit testing.
- The next step is to determine the new position for the object.
- *MouseMove* events are used to report the new location continuously.

How to Display the Object

- Two alternatives are possible while we are dragging the object:
    1. Display the object in the new position continuously as we drag it.
    2. Display the object only when we let up on the mouse button.
- Sometimes the outline of the object is displayed rather than the entire object.
- This is desirable when the objects are complex and require too much CPU time to redraw continuously.

Dragging Example

- This example allows dragging a filled rectangle.
- The left mouse button is used.
- Note that the cursor switches to a cross while dragging.
- State information needs to be kept to know when to ignore the *MouseMove* events.

Dragging Example

```
DragRect1 - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace DragRect1
{
    public partial class Form1 : Form
    {
        public Form1()
```

Dragging Example

```
        {
            InitializeComponent();
        }
        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.FillRectangle(Brushes.Red, rect);
        }
        protected override void OnMouseDown(MouseEventArgs
              e)
        {
            if (e.Button == MouseButtons.Left)
            {
```

Dragging Example

```
                Point p = new Point(e.X, e.Y);
                if (rect.Contains(p))
                {
                    dragging = true;
                    offset = (Size)(rect.Location - new
                         Size(e.X, e.Y));
                    Cursor = Cursors.Cross;
                }
            }
        }
        protected override void OnMouseUp(MouseEventArgs
                     e)
```

Dragging Example

```
        {
            if (e.Button == MouseButtons.Left)
            {
                dragging = false;
                Cursor = Cursors.Default;
            }
        }
        protected override void OnMouseMove(MouseEventArgs
            e)
        {
            if (dragging)
            {
                Invalidate(rect); //old position
```

Dragging Example

```
                rect.Location = new Point(e.X, e.Y) +
                   offset;
                Invalidate(rect); //new position
            }
        }
        private Rectangle rect = new Rectangle(0, 0, 100,
                 100);
        private bool dragging = false;
        private Size offset;
    }
}
```

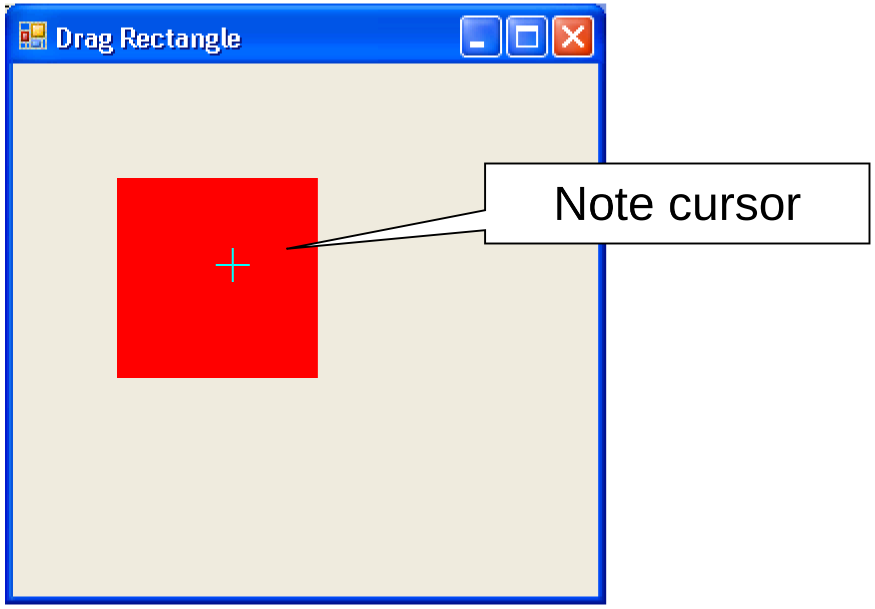
Hit Testing Made Easy

- The *Rectangle* class has a nice method to determine if a point is inside the rectangle.
- *Contains(p)* returns *true* if *p* is inside the rectangle.

Steps to Drag

1. Hit test on *MouseDown*. If inside the rectangle then save *offset* from upper left hand corner.
2. Set the cursor to a cross.
3. Set flag to indicate dragging in progress.
4. *MouseMove* event calculates the new upper left corner using the mouse position and the previously calculated offset.
5. Invalidate the client area that needs to be updated (old and new positions).
6. *MouseUp* restores the cursor and clears the dragging flag.

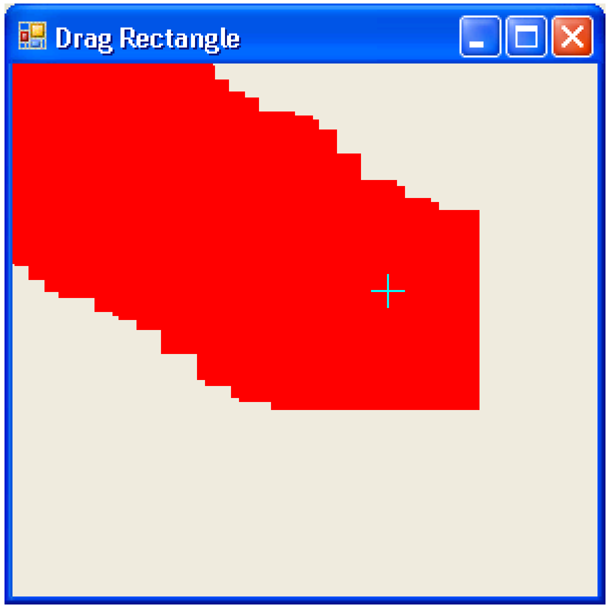
Output



Invalidation Notes

- It is important to invalidate both the previous and new positions of the dragged object.
- The following shows what happens if we drag the rectangle without invalidating the previous position.

Oops!



Mouse Captures

- Mouse events normally go to the window that is clicked in. In other words if you have more than one application on the screen the events are directed to the appropriate application.
- Clicking on title bars, borders etc. direct mouse events to specific handlers in the application or the operating system.
- This presents a problem when dragging objects since going out of the bounds of the client area would result in a potentially missed mouse event, e.g., the *MouseUp* event.
- The .NET FCL automatically captures all mouse messages while a button is held down and directs them to the same window (form) as the *MouseDown* event.

Mouse Captures – Contd.

- Even if you let up the mouse button outside the client area of your form you will still get the *MouseUp* event as if you let on the button inside your form.
- In earlier programming frameworks such as MFC, the programmer needed to explicitly make function calls to capture the mouse.

The Wheel

- The mouse wheel, if one is available, can be used for anything the programmer wants.
- .NET provides some built in capabilities for the mouse wheel.
- If a control derives from *ScrollableControl* and the *VScroll* property is set to true, the mouse wheel scrolls through the control vertically.
- If *VScroll* is set to false and the *HScroll* property is set to true, the mouse wheel scrolls through horizontally.

Working With the Wheel

- The wheel is a relative device and there is no absolute positioning available.
- We can go in one direction or another.
- The concept of a *detent* is defined for the wheel even if the wheel itself does not exhibit this behavior.
- The *MouseWheel* event is triggered when the mouse wheel is moved in either direction.
- The *Delta* property of *MouseEventArgs* provides the number of detents the wheel has rotated over. Unfortunately the value returned bears little correlation with reality. The value is either +120 or -120 unless you rotate the wheel extremely rapidly. In that case it is a multiple of 120.
- The best way to use the wheel is to merely detect if it is greater or less than zero and either increment or decrement the value being controlled.

Wheel Example

- The following example changes the shade of the background color using the wheel.
- The RGB value for shades of gray is one in which all three components are equal and range between 0 and 255.

Wheel Example

```
MouseWheel - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace MouseWheel
{
    public partial class Form1 : Form
    {
        public Form1()
```

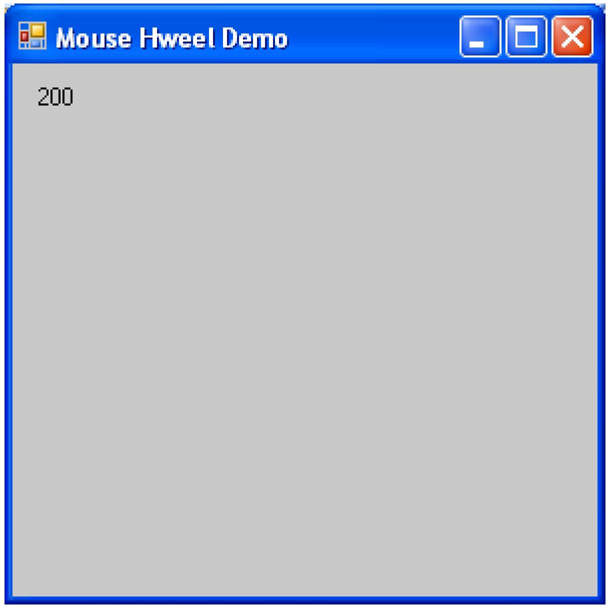
Wheel Example

```
        {
            InitializeComponent();
            BackColor = Color.FromArgb(128, 128, 128);
        }
        protected override void OnPaint(PaintEventArgs e)
        {
            e.Graphics.DrawString(shade.ToString(), Font,
                 Brushes.Black, 10, 10);
        }
        protected override void
          OnMouseWheel(MouseEventArgs
                 e)
        {
            int i = e.Delta;
```

Wheel Example

```
            if (i > 0) ++shade;
            else --shade;
            if (shade < 0) shade = 0;
            if (shade>255) shade=255;
            BackColor = Color.FromArgb(shade, shade,
                  shade);
            Invalidate();
        }
        private int shade=128;
    }
}
```

Output



```
200
```