The Mouse and Keyboard

Mouse
- A pointing device with one or more buttons
- Important input device, but optional
- User moves physical mouse =>
  - Windows moves small bitmapped image (mouse cursor) on display
  - "Hot spot" points to a precise location on display
  - Hot spot position constantly updated by low-level logic inside Windows

Mouse Actions
- Button Down, Button Up
- Wheel movement
- Moving mouse
- Clicking
  - Pressing and releasing a mouse button
- Dragging
  - Moving mouse while a button is pressed down
- Double Clicking
  - Clicking a button twice in succession
  - Must occur within a set period of time and with mouse cursor in approximately the same place
  - Form's SystemInformation class has two properties that give this information:
    - int DoubleClickTime
    - Size DoubleClickSize

Information about Mouse
- More Form's SystemInformation Properties:
  - bool MousePresent
  - int MouseButtons
    - Specifies which button was pressed
    - Enumeration values: None, Left, Right, Middle
  - bool MouseButtonsSwapped
  - bool MouseWheelPresent
  - int MouseWheelScrollLines

Mouse Events
- The "Control" Class defines 9 mouse events and 9 corresponding protected event handler methods
  - Form class is derived from Control class
  - Only one control or form receives mouse events
    - The one that has its Enabled and Visible properties set to true
    - If multiple controls are stacked, the enabled visible control on top receives the event
    - A Form object receives mouse events only when mouse is over its client area
      - But mouse can be "captured" by a control => it can receive mouse events when mouse is not over it

Some Basic Mouse Events and Handler Methods
- MouseDown OnMouseDown( )
- MouseUp OnMouseUp( )
- MouseMove OnMouseMove( )
- MouseWheel OnMouseWheel( )
  - Delegate for each event: MouseEventHandler
  - 2nd argument for each handler: MouseEventArgs
- Click OnClick( )
- DoubleClick OnDoubleClick( )
  - Delegate for each event: EventHandler
  - 2nd argument for each handler: EventArgs
**MouseEventArgs Properties**

- Give access to read-only data that comes with mouse event
  - int X  Horizontal position of mouse
  - int Y  Vertical position of mouse
  - MouseButtons Button
    - Enumeration possibilities:
      - None, Left, Right, Middle
      - Indicates which button or buttons are currently pressed
      - Each button corresponds to a bit set
  - int Clicks
  - int Delta

**Click/DoubleClick EventArgs Static Properties**

- Give access to read-only data that comes with mouse Click and DoubleClick events
  - Point MousePosition
    - Result in screen coordinates
    - To convert to client area coordinates, use PointToClient()
  - MouseButtons MouseButtons

**Sketching Example Program**

- Sketch-dotNet
  - Sketching revisited
    - Using C# and the .NET Framework Class Library
  - But if window is exposed, the sketch disappears

- Two ways to avoid this:
  1. Save the points in each sketch and redraw all line segments in response to Paint event
  2. Draw the sketch on a shadow bitmap that the program draws on while it’s drawing on the screen
     - Then redraw the bitmap in response to Paint event

**Saving the Sketch points**

- Could use an array:
  - Point[ ] apts = new Point[?????]
  - But how big?

- Better to use a C# dynamic "ArrayList"
  - A class defined in System.Collections namespace
    - Also has data structures like: Queue, Stack, SortedList, Hashtable
  - To create a new ArrayList:
    - ArrayList arrlst = new ArrayList();
    - Could hold any data type(s)
  - To add elements, e.g., a Point p:
    - arrlst.Add(p);
    - Can also Insert( ) and Remove( ) elements
  - Accessing an element: use an indexer as for an ordinary array
    - Point p = (Point) arrlst[2];
    - Note typecast
    - Needed because indexer returns an object of type Object
  - Number of objects in an ArrayList: arrlst.Count

**New Sketch-dotNet using an ArrayList**

- A single run can have many sketches
  - One for each time left mouse button goes down
  - So use one ArrayList to store the points for each sketch
  - When finished (when mouse button goes up), convert it into an array of Points
  - Use a second ArrayList to store the array of points for each sketch (i.e., an ArrayList of sketches)
- Each time left mouse button goes down, start a new sketch’s ArrayList
- Each time mouse moves with left button down, draw line segment and add the point to current sketch’s ArrayList
- In response to Paint event, use DrawLines(…) to draw all the line segments in each ArrayList
  - g.DrawLines(Pen pen, Point[ ] a_pts); // a_pts is an array of Points
- See Sketch-dotNet-ArrayList example program
  - Here we’re really storing the drawing in a Metafile format

**New Sketch-dotNet using a Shadow Bitmap**

- Store the window client area as a shadow bitmap
  - Draw on it and on screen when mouse moves with its left button down
  - Draw the shadow bitmap on the screen when a Paint event occurs
  - Note that with this technique all of the information on the original points is lost
  - See the Sketch-dotNet-Bitmap example program
**Some Other Mouse Events and Event Handlers**
- **MouseEnter**  `OnMouseEnter( )`
  - Mouse cursor has been moved onto form’s client area
- **MouseLeave**  `OnMouseLeave( )`
  - Mouse cursor is no longer on top of client area
- **MouseHover**  `OnMouseHover( )`
  - Mouse cursor has entered client area and has stopped moving
  - Only happens once between MouseEnter and MouseLeave events
- Delegate for each: `EventHandler`
- Argument for each: `EventArgs`
- See Mouse-Enter-Leave-Hover example program

**The Mouse Cursor**
- A little bitmap on screen that indicates the location of the mouse
- Can change its appearance
- It’s an object of type ‘Cursor’ defined in `System.Windows.Forms`
- Get a mouse cursor from the ‘Cursors’ class
  - Consists of 28 static read-only properties that return predefined objects of type ‘Cursor’, e.g.:
    - Arrow, Cross, Default, Hand, Help, Beam, WaitCursor, etc.
- Some static read/write properties of ‘Cursor’ class:
  - `Cursor Current`
  - `Point Position`
  - For example to display the hourglass cursor on the form:
    - `Cursor.Current = Cursors.WaitCursor;`
- Some Cursor methods:
  - `Show( )`, `Hide( )`
- See MouseCursors example program

**The Keyboard**
- A shared resource in Windows
  - All applications receive input from same keyboard
  - But any keystroke has a single destination
    - The destination is always a ‘Control’ (e.g. a Form)
  - Object that receives a keyboard event has the “input focus”
    - the active Form
  - Usually the topmost form
  - If form has a caption bar, it is highlighted
  - Form.IsActive property returns the active form
  - this.Form.Activate() method can be used to make this form the active form

**Keys and Characters**
- Think of keyboard in two ways:
  - A collection of distinct physical keys
    - Code generated by a key press or release identifies the key
  - A means of generating character codes
    - Code generated identifies a character in a character set
      - Traditionally 8-bit ASCII code
      - In Windows, extended to 16-bit Unicode
      - Keyboard combinations (Shift, etc.) taken into account

**Types of Keys**
- Keyboard divided into four general groups of keys
  - **Toggle keys**: Pressing key changes state
    - Caps Lock, Num Lock, Scroll Lock, Insert
  - **Modifier keys**: Pressing key affects interpretation of other keys
    - Shift, Ctrl, Alt
  - **Noncharacter keys**: Not associated with displayable characters; direct a program to carry out certain actions
    - Function keys, PgUp, PgDn, Home, End, Insert, Delete, Arrow keys
  - **Character keys**: Letters, numbers, symbol keys, spacebar, Backspace, Tab key
    - Generate ASCII/Unicode codes when pressed

**Keyboard Events & Data**
- **KeyDown**, `KeyEventArg`  `KeyEventArgs`
  - When a key is pressed (WM_KEYDOWN)
- **KeyPress**, `KeyPressEventArg`  `KeyPressEventArgs`
  - When a character-generating key is pressed (WM_CHAR)
  - Occurs after a KeyDown event
- **KeyUp**, `KeyEventArg`  `KeyEventArgs`
  - When a key is released (WM_KEYUP)
- Note KeyUp/KeyDown and KeyPress event data is different
  - KeyUp/KeyDown events provide low-level information about the keystroke — which key
  - KeyPress provides the character code
    - Keyboard combinations taken care of
KeyDown/KeyUp Events

KeyDownEventArgs Properties:
- Keys KeyCode Identifies which key
- Keys Modifiers Identifies shift states
- Keys KeyData Combines KeyCode & Modifiers
  - Keys: a huge enumeration, some examples:
    - Keys.A, Keys.z, Keys.D0 (zero key), Keys.F1, Keys.Add,
  - See Online Help on “Keys enumeration”
- bool Shift True if Shift key is pressed
- bool Alt True if Alt key is pressed
- bool Handled Set by event handler (initially false)
- int KeyValue Returns KeyData as an integer

KeyPress Event

KeyPressEventArgs Properties:
- char KeyChar Unicode/ASCII character code
- bool Handled Set by handler (initially false)

Two Example Programs

Key:
- Assembles incoming characters from keyboard into a string that is displayed on the form’s client area
  - Handles Backspace key by removing last character from string
- Handles KeyPress event

KeyArrow:
- Moves an image on the form’s client area in response to keyboard Left/Right/Up/Down arrow key presses
- Handles KeyDown event