Ink and Windows Presentation Foundation

Lecture #4: Ink and WPF

Joseph J. LaViola Jr.

Fall 2009

From Last Time

- Windows Presentation Foundation (WPF)
  - integration of
    - ink
    - 2D Graphics
    - 3D Graphics
    - video and audio
  - uses visual tree model
    - component based
- XAML and C# code
- Important control – *InkCanvas*

Important Ink Components

- InkCanvas – System.Windows.Controls
  - receives and displays ink strokes
  - starting point for ink applications
  - stores ink in *Strokes*
- System.Windows.Ink Namespace
  - contains classes to interact with and manipulate ink
  - examples
    - *Stroke*
    - *InkRecognizer*
    - *InkAnalyzer*
    - *GestureRecognizer*

Dealing with InkCanvas

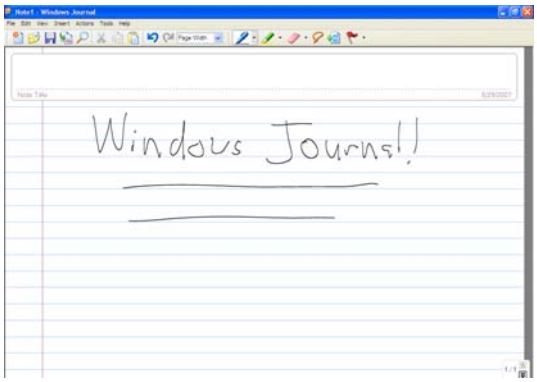
- *InkCanvas* collects Strokes
- Strokes contain *StylusPoints*
- *StylusPoints* contain X,Y, Pressure
  - can also be converted into Geometry objects
- Strokes contain
  - digitizer packets
  - drawing attributes
  - application-defined data
- *InkCanvas* has several stylus level events
  - *StrokeCollected, StylusInAirMove*, …

Strokes and Geometry

- Strokes
  - perform hit tests
  - get geometry, bounds, Bezier points
  - add properties
  - transformations
- Geometry
  - lose pressure and stylus specific data
  - Within scope of 2D graphics API
  - get area
  - create shapes
- No Cusp or self-intersection detection

More InkCanvas Features

- Enough support to implement Windows Journal
- Modes
  - Ink
  - InkandGesture
  - GestureOnly
  - EraseByStroke
  - EraseByPoint
  - Select
  - None



Drawing Attributes

- Can access on stroke level using *Drawing Attributes* property
- Can access on global level using the InkCanvas *DefaultDrawingAttributes* property
- Example attributes
  - color
  - Bezier curves
  - height and width of ink stroke
  - ignoring pressure

InkCanvas Example

```
<Window x:Class="WpfApplication4.InkTest"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="InkTest" Height="300" Width="300"
    Visibility='Visible'>
    <Grid>
      <InkCanvas
        Name='_ink'
        StrokeCollected='Collected'
        Background='Beige' />
        <Canvas Name='_overlay' />
    </Grid>
</Window>
```

```
private void Collected(object sender, InkCanvasStrokeCollectedEventArgs e)
       {
           _overlay.Children.Clear();
           Brush fill = new SolidColorBrush(Color.FromArgb(120, 255, 0, 0));
           foreach (StylusPoint pt in e.Stroke.StylusPoints)
           {
               double markerSize = pt.PressureFactor * 35.0;
               Ellipse marker = new Ellipse();
               Canvas.SetLeft(marker, pt.X - markerSize / 2);
               Canvas.SetTop(marker, pt.Y - markerSize / 2);
               marker.Width = marker.Height = markerSize;
               marker.Fill = fill;
               _overlay.Children.Add(marker);
           }
       }
```

Examples adapted from *Essential Windows Presentation Foundation* by Chris Anderson, Addison Wesley, 2007.

DrawingAttributes Example

```
<ComboBox Name="cbInkColors" SelectionChanged="cbInkColors_SelectionChanged" Width="35"
    SelectedIndex="0">
    <ComboBoxItem Background="Black" Foreground="White">Black</ComboBoxItem>
    <ComboBoxItem Background="Red" Foreground="White">Red</ComboBoxItem>
    <ComboBoxItem Background="Blue" Foreground="White">Blue</ComboBoxItem>
    <ComboBoxItem Background="Purple" Foreground="White">Purple</ComboBoxItem>
    <ComboBoxItem Background="Green" Foreground="White">Green</ComboBoxItem>
</ComboBox>

private void cbInkColors_SelectionChanged( object sender, SelectionChangedEventArgs e )
{
    if ( cbInkColors.SelectedIndex < 0 )
    {
           return;
    }

    if ( icMain != null && icMain.IsLoaded )
     {
           Brush brush = ( (ComboBoxItem)cbInkColors.Items[cbInkColors.SelectedIndex] ).Background;
           Color color = ( (SolidColorBrush)brush ).Color;
           daPen.Color = color;
           icMain.DefaultDrawingAttributes = daPen;

           cbHighlightColors.SelectedIndex = -1;
    }
}
```


DrawingAttributes with Data Binding Example

```
<Grid.Resources>
    <x:Array x:Key="penAttributes" x:Type="{x:Type DrawingAttributes}">
           <DrawingAttributes Color="Black" Width="2" Height="2" IsHighlighter="False" />
           <DrawingAttributes Color="Red" Width="2" Height="2" IsHighlighter="False" />
           <DrawingAttributes Color="Blue" Width="2" Height="2" IsHighlighter="False" />
           <DrawingAttributes Color="Purple" Width="2" Height="2" IsHighlighter="False" />
           <DrawingAttributes Color="Green" Width="2" Height="2" IsHighlighter="False" />
    </x:Array>
    <DataTemplate DataType="{x:Type DrawingAttributes}">
           <Label Name="label" Foreground="White" Content="Pen">
                  <Label.Background>
                         <SolidColorBrush Color="{Binding Path=Color}" />
                  </Label.Background>
           </Label>
    </DataTemplate>
</Grid.Resources>

...

<ComboBox Name="cbInkColors" Width="45" SelectedIndex="0" ItemsSource="{StaticResource
    penAttributes}" />
```

Creating Your Own InkCanvas

- *InkCanvas* handles approx. 90-95% of what you need
- Can develop custom *InkCanvas*
  - *InkPresenter* – System.Windows.Controls
  - *DynamicRenderer* – System.Windows.Input.StylusPlugins
  - Stylus events
- See Windows SDK documentation

Stylus Descriptions

- Other data besides x,y points and pressure
  - xtilt, ytilt
  - Barrel button
- Can request data globally using *DefaultStylusPointDescription* on *InkCanvas*
- Per stroke with *Reformat* method on *StylusPointCollection*

Stylus Description Example

```
public InkTest()  {
  InitializeComponent();
  _ink.DefaultStylusPointDescription = new StylusPointDescription(
    new StylusPointPropertyInfo[] {
      new StylusPointPropertyInfo(StylusPointProperties.X),
      new StylusPointPropertyInfo(StylusPointProperties.Y),
      new StylusPointPropertyInfo(StylusPointProperties.NormalPressure),
      new StylusPointPropertyInfo(StylusPointProperties.BarrelButton), });
}
```

Asks for information on x,y, pressure, and if the barrel button is pressed.

Gesture Recognition

- Built in Gesture recognition engine
  - handwriting recognition and ink analysis are separate (outside of InkCanvas)
- 41 distinct gestures (found in ApplicationGesture enum)
  - check
  - square
  - triangle
  - arrows
  - scratchout
  - etc…

Gesture Recognition Example

```
<Window x:Class="WpfApplication6.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title='GestureTester'>
    <StackPanel>
        <InkCanvas Height='200' Name='_ink'
         Gesture='InkGesture'
         EditingMode='InkAndGesture' />
        <ListBox Name='_seen' />
    </StackPanel>
</Window>
```

```
public partial class Window1 : Window  {
       public Window1()  {
           InitializeComponent();
           _ink.SetEnabledGestures(new ApplicationGesture[] {
                                   ApplicationGesture.AllGestures,});
       }

       private void InkGesture(object sender, InkCanvasGestureEventArgs e) {
         _seen.Items.Add(e.GetGestureRecognitionResults()[0].ApplicationGesture);
       }
}
```


Property Data

- Data can be added to many classes (Stroke, StylusPoint)
- Adding Property Data to a Stroke
  - Guid (Globally Unique Identifier) for each property you want to add
  - Create a Guid
    - Can pass in a string of 32 hexadecimal characters
  - Example:

```
Guid guid = new Guid( "aaaaaaaa-bbbb-cccc-ddddeeee-
   ffffffffffff" );
List<String> property = new List<string>() { "one", "two",
   "three" };
stroke.AddPropertyData( guid, property );
```

Property Data – Cont.

- Check For Property Data
  - Pass in the Guid you want to check for
  - Example:

```
if ( stroke.ContainsPropertyData( guid ) ) ...
```

- List Property Data IDs
- Retrieve Property Data
  - Example:

```
List<String> property = stroke.GetPropertyData( guid ) as
   List<String>;
```


Collecting Timing Information

```
// Create a guid for the date/timestamp.
Guid dtGuid = new Guid("03457307-3475-3450-3035-640435034540");
DateTime now = DateTime.Now;

// Check whether the property is already saved
if (thisStroke.ContainsPropertyData(dtGuid)) {
  // Check whether the existing property matches the current date/timestamp
  DateTime oldDT = (DateTime)thisStroke.GetPropertyData(dtGuid);
  if (oldDT != now) {
    // Update the current date and time
    thisStroke.AddPropertyData(dtGuid, now);
  }
}
```

This snippet works on a Stroke by Stroke basis. Can you think of how to do this on a point by point basis?

InkAnalyzer

- Basic Usage
  - Add strokes to analyzer
  - Analyze or BackgroundAnalyze
  - When analysis is done, look at results
    - ContextNodes form a tree structure
- A ContextNode can be one of several subclasses
  - InkWordNode, InkDrawingNode, InkBulletNode, WritingRegionNode, ParagraphNode, LineNode, etc.
  - Determine its type and cast to appropriate subtype to get results
    - Example:

```
if ( node is InkWordNode ) ...
```

    - Example:

```
if ( node.Type == ContextNodeType.InkWord ) ...
```

    - Check subnodes too

Other Useful Things

- List<Type>
- Hashtable
- Queue<Type>

- Accessors

```
public int ID
{
     get{ return myID; }
     set{ myID = value; }
}
```

- foreach

```
foreach ( Stroke stroke in icMain.Strokes ) ...
```

- is

```
if ( ContextNode node is InkWordNode ) ...
```

- as

```
InkWordNode word = node as InkWordNode;
```

Assignments

- Readings
  - Windows SDK documentation
    - Windows.System.Control.InkCanvas
    - Windows.System.Ink