

Using the OpenNETCF Mobile Ink Library for Windows Mobile 6

Mark Arteaga
OpenNETCF Consulting
September 2007

Contents

Introduction
Technical Overview
Mobile Ink Library Custom Controls
Samples Available with the Mobile Ink Library
Conclusion

Introduction

With the recent release of the Mobile Ink Library for Windows Mobile 6, some people are probably thinking 'well what can I use that for'. In this article we will be going over some technical information and some possible uses for inking on the Windows Mobile 6 platform.

Technical Overview

Windows Mobile 6 Inking

In the latest release of Windows Mobile, there is a new API called Windows Mobile Ink. What is Windows Mobile Ink? Where does it come from? Why do you need it? In a nutshell, Windows Mobile Ink is a new Inking API only available for the Windows Mobile 6 platform called WISP Lite. It is based on Windows Inking Services Platform (WISP) for the Tablet PC.

Desktop developers have managed classes available under the Microsoft.Ink namespace to allow them to develop managed applications to take advantage of 'inking' features available on Tablet PC. This article does not deal with the desktop so I'll leave it up to the user to search the many articles available on inking on the Tablet PC.

The Windows Mobile 6 SDK contains some examples of using WISP Lite with native code. Unfortunately if you are a managed developer, there are no managed classes available for your .NET Compact Framework applications. The Mobile Ink Library addresses this issue and provides a library of classes which wrap the COM interfaces and the controls that are available on the Windows Mobile 6 platform.

Differences from the Desktop and Device

The Mobile Ink Library and WISP Lite allows a developer to add inking functionality to their Windows Mobile application. WISP Lite (as the name implies) is a 'lighter' version or a subset of WISP available on the desktop. For the managed desktop developer, there is the Microsoft.Ink namespace which provides inking capabilities in your .NET Framework application. For the native developer there is also a COM API available to add inking capabilities to a native application.

Currently for inking capabilities on the Windows Mobile 6 platform there are only COM interfaces and a 'windowed control' named InkCanvas available to native developers. Managed developers using .NET Compact Framework don't have any managed classes available but can wrap the COM interfaces and the InkCanvas control using version 2 of .NET Compact Framework. The Mobile Ink Library has already

done this and is available under the `OpenNETCF.WindowsMobile.Ink` namespace. This gives the managed developer the ability to take advantage of this new APIs available.

Both the desktop and the Windows Mobile versions of inking APIs allow the developer to save the ink in a format called Ink Serialized Format or ISF. The ISF data output, which can be saved as Base64 or Binary, is compatible on both the desktop and mobile device. If you try to open an ISF stream on mobile device that was saved on the desktop, the mobile device will ignore any properties in the stream that are not supported on Windows Mobile but still preserve the properties when saved again. See this [page](#) for more details on the differences between the desktop and device.

COM Interfaces

There are various COM interfaces available for Windows Mobile Ink and for details on using it with native code see '[Using the Automation Library with Windows Mobile Ink](#)'. The Mobile Ink Library wraps the appropriate interfaces to allow a managed developer to ink enable a .NET Compact Framework application. Two interfaces that are primarily used and will be covering next are `IInkDisp` and `IInkOverlay`.

IInkOverlay

An `IInkOverlay` object is used to attach to a window (or control) using the `IInkOverlay.hWnd` property. When you 'attach' a window to an `IInkOverlay` object, the window becomes 'ink enabled' and allows the collection of ink data.

`IInkOverlay` also allows you to change the size, shape and color of the ink using the `IInkOverlay.DefaultDrawingAttributes` property. The following example changes the size of the pen width using a `TrackBar.Value` property.

```
IInkDrawingAttributes attrs = overlay.DefaultDrawingAttributes;
attrs.Width = (float)trackBar1.Value;
overlay.DefaultDrawingAttributes = attrs;
```

An example use of `IInkOverlay` is attaching it to a `PictureBox` and adding ink notes to an image. The following code sample uses the `OpenNETCF.WindowsMobile.Ink.Ink` class to achieve this:

```
PictureBox pb = new PictureBox();
//Create a new InkOverlay object
OpenNETCF.WindowsMobile.Ink.Ink ink = new OpenNETCF.WindowsMobile.Ink.Ink();
//We need to set the handle (or control) the inkoverlay object is bound to
ink.IInkOverlay.hWnd = pb.Handle;
ink.IInkOverlay.Enabled = true;
```

To change the editing mode of the Ink object the `EditingMode` property can be set with one of the following values:

```
public enum InkOverlayEditingMode
{
    Ink = 0,
    Delete = 1,
    Select = 2
}
```

`IInkOverlay` exposes a property called `Ink` which gets or sets the `IInkDisp` object that is associated with the `IInkOverlay` object.

InkDisp

IInkDisp is responsible for collecting the ink strokes from a user. It also has the ability to Save() and Load() the ink strokes collected by an IInkDisp. To create an instance of IInkDisp, use OpenNETCF.WindowsMobile.Ink.InkDisp class has one static method called CreateInstance().

Usually you will want to create a new IInkDisp when you want to clear existing ink in an IInkOverlay.Ink. An example of this is:

```
IInkDisp inkDisp = InkDisp.CreateInstance();  
overlay.Enabled = false;  
overlay.Ink = inkDisp;  
overlay.Enabled = true;
```

Note, to set the IInkOverlay.Ink property, you must first set IInkOverlay.Enabled to false and then once you set the IInkOverlay.Ink property set the IInkOverlay.Enabled back to true.

When saving the ink strokes, the Save() method returns a byte[] and there are four possible options defined in OpenNETCF.WindowsMobile.Ink.InkPersistenceFormat:

- InkSerializedFormat - The ink data using ISF. This is the most compact representation of ink data.
- Base64InkSerializedFormat - The ink data using ISF and Base64 encoded.
- Gif - The ink data in Graphics Interchange Format (Gif)
- Base64Gif - The ink data Graphics Interchange Format (Gif) and Base64 encoded

These options allow the developer to save in various formats depending on their application requirements. For example, if you are emailing the collected ink, the receiver may not be able to render the ISF format so you will probably want to send it as a Gif image attached to the email.

IInkDisp has a property call ExtendedProperties which allows the you to save custom data within the ink data. For example, using ExtendedProperties you can 'embedded' some data to know when the ink data was last saved. One caveat is ExtendedProperties are only persisted when saving as either InkSerializedFormat or Base64InkSerializedFormat so if you save as a Gif or Base64Gif, this data will be lost.

That summarizes IInkOverlay and IInkDisp which are probably the two main classes needed for inking. I'll leave it up to the reader to explore some of the other methods and properties available in these interfaces and some of the other interfaces available.

InkCanvas Control

The InkCanvas control is a windowed control and allows the developer to avoid using the COM Interfaces for inking in their application. In addition to the COM Interfaces wrapped by Mobile Ink Library, the library also includes a managed wrapper appropriately called InkCanvas (found under OpenNETCF.WindowsMobile.Ink.InkCanvas).

InkCanvas simplifies capturing ink within your application. It uses windows messages to request operations and notify the user of any changes like adding or deleting ink. If using the IInkDisp COM object is desired, the OpenNETCF.WindowsMobile.Ink.InkCanvas.Ink property can be used. InkCanvas also has the following properties which may be useful

- Mode - Gets or sets the editing mode
- Ink - Gets the IInkDisp object

- BackColor - Gets or Sets the background color
- ZoomLevel - Gets or sets the zoom level
- CanvasSize - Gets or sets the canvas size
- RecognizedText - Gets the recognized ink as text
- PenStyle - Gets or sets the style of pen to use for example pen or highlighter

There are also various methods available in the InkCanvas control which is pretty self explanatory so I'll leave it up to the reader to explore.

Ink Recognition

One of the great benefits of the Windows Mobile Ink API is the ability to convert the ink data to a text string via ink recognition.

The following code sample will convert ink data to a text string using the COM objects:

```
private Ink m_ink;

public override string ToString()
{
    //Get the appropriate COM objects required.
    IInkDisp disp = m_ink.InkDisp;
    IInkStrokes strokes = disp.Strokes;
    IInkRecognitionResult results = strokes.RecognitionResult;

    //Get the recognized string. An alternate way of doing this is
    strokes.ToString()
    string ret = results == null ? "" : results.TopString;

    //Release the COM objects
    if (results != null)
        Marshal.ReleaseComObject(results);
    Marshal.ReleaseComObject(strokes);
    Marshal.ReleaseComObject(disp);

    //Return the recognized text
    return ret;
}
```

If you use the InkCanvas control the RecognizedText property can be used. Internally the InkCanvas control does the following to recognize the ink data:

```
public string RecognizedText
{
    get
    {
        return Ink.Strokes.RecognitionResult.TopString;
    }
}

public Interfaces.IInkDisp Ink
{
    get
    {
        if (OpenNETCF.Windows.Forms.StaticMethods.IsDesignTime)
            return null;
        IntPtr pInk;
```

```

NativeMethods.SendMessage(childHandle, (int)ICM.GETINK, 0, out pInk);
if (pInk == IntPtr.Zero)
    return null;
return (Interfaces.IInkDisp)Marshal.GetObjectForIUnknown(pInk);
}
}

```

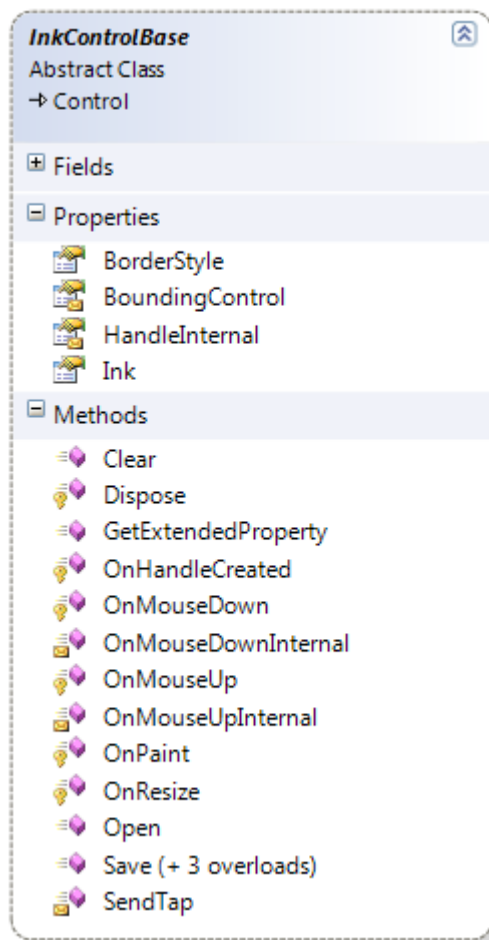
Essentially both methods would produce the same results for the same ink data.

Mobile Ink Library Custom Controls

The Mobile Ink Library consists of four controls to make it easier for the developer to ink enable a .NET Compact Framework application. Using the IInkOverlay is pretty straight forward, but to simplify the life of the developer various controls were created to ink enable .NET Compact Framework applications using Windows Mobile 6.0.

InkControlBase

InkControlBase is the abstract base class in which the InkRecognizer, InkPicture and InkSignature inherit from. The following is the class diagram of the control:



As you can see from the diagram InkControlBase inherits from System.Windows.Forms.Control. 'Inking' the control is enabled by the Ink property which wraps the IInkOverlay interface (for more information on IInkOverlay see the [above](#)).

Since `IInkOverlay` requires a handle to the window (or control) to attach to, we override the `OnHandleCreated()` in the `InkControlBase` class to attach the control. The following code is used to accomplish this:

```
protected override void OnHandleCreated(EventArgs e)
{
    base.OnHandleCreated(e);
    if (StaticMethods.IsRunTime)
    {
        if (m_inkOverlay == null)
        {
            //Create the IInkOverlay object
            m_inkOverlay = new Ink();

            //Listen for all stroke events
            m_inkOverlay.IInkOverlay.SetEventInterest(InkCollectorEventInterest.Stroke,
            true);

            //We need to set the handle (or control) the inkoverlay object is
            bound to
            m_inkOverlay.IInkOverlay.hWnd = this.HandleInternal;
            m_inkOverlay.IInkOverlay.Enabled = true;
        }
    }
}
```

You will notice the following line:

```
m_inkOverlay.IInkOverlay.hWnd = this.HandleInternal;
```

`HandleInternal` is an internal virtual property and returns the `Handle` of the `InkControlBase` control used to attach to the `IInkOverlay`.

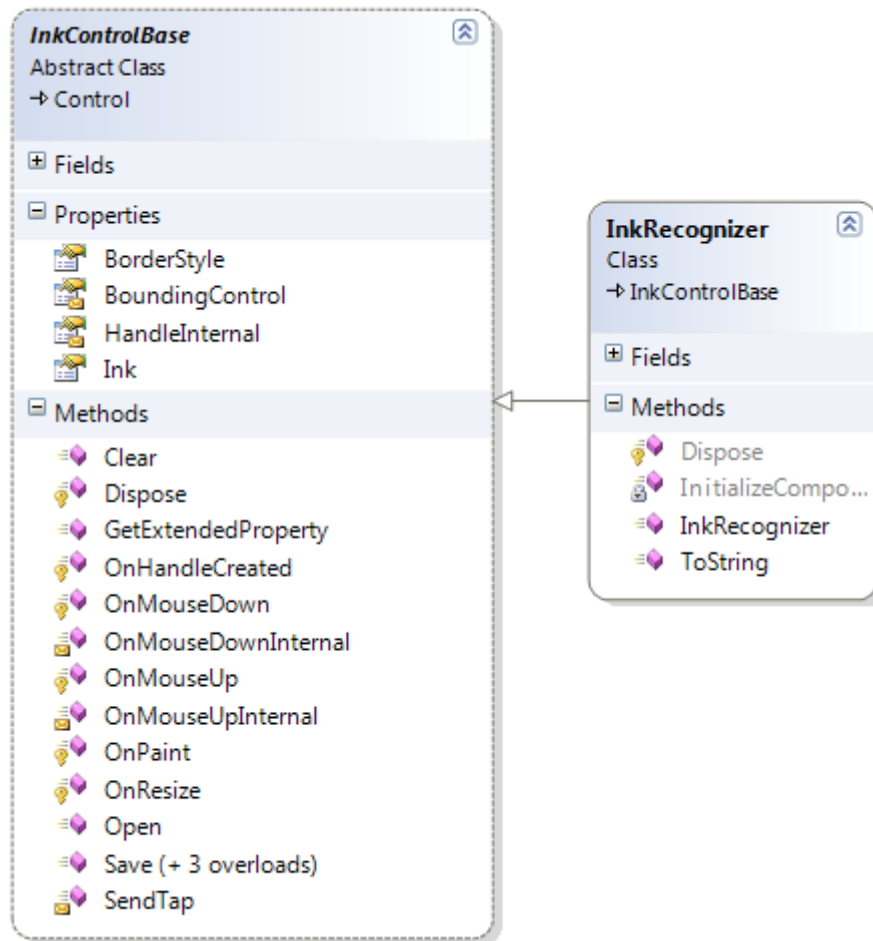
```
/// <summary>
/// Gets the internal handle to associate the IInkOverlay object with
/// </summary>
internal virtual IntPtr HandleInternal
{
    get
    {
        return this.Handle;
    }
}
```

Classes inheriting from `InkControlBase` have the option to override this property to return a different handle. We will visit this again when we discuss the remaining controls.

`InkControlBase` provides all the basic functionality such as `Clear()`, `Open()` and `Save()` methods. These methods internally use the `Ink` property to make the required calls to `IInkOverlay`.

InkRecognizer

`InkRecognizer` class was created to allow the developer to recognize ink inputted by a user into plain text and inherits from `InkControlBase`.



From the class diagram, you will see that InkRecognizer does not do much except override the ToString() method. The ToString() method will return the text string that is recognized by the InkOverlay object using the following code:

```

public override string ToString()
{
    if (StaticMethods.IsDesignTime || this.InkOverlay == null)
        return base.ToString();
    else
    {
        //Get the appropriate COM objects required.
        IInkDisp disp = Ink.InkDisp;
        IInkStrokes strokes = disp.Strokes;
        IInkRecognitionResult results = strokes.RecognitionResult;

        //Get the recognized string. An alternate way of doing this is
        strokes.ToString()
        string ret = results == null ? "" : results.TopString;

        //Release the COM objects
        if (results != null)
            Marshal.ReleaseComObject(results);
        Marshal.ReleaseComObject(strokes);
        Marshal.ReleaseComObject(disp);
    }
}
  
```

```

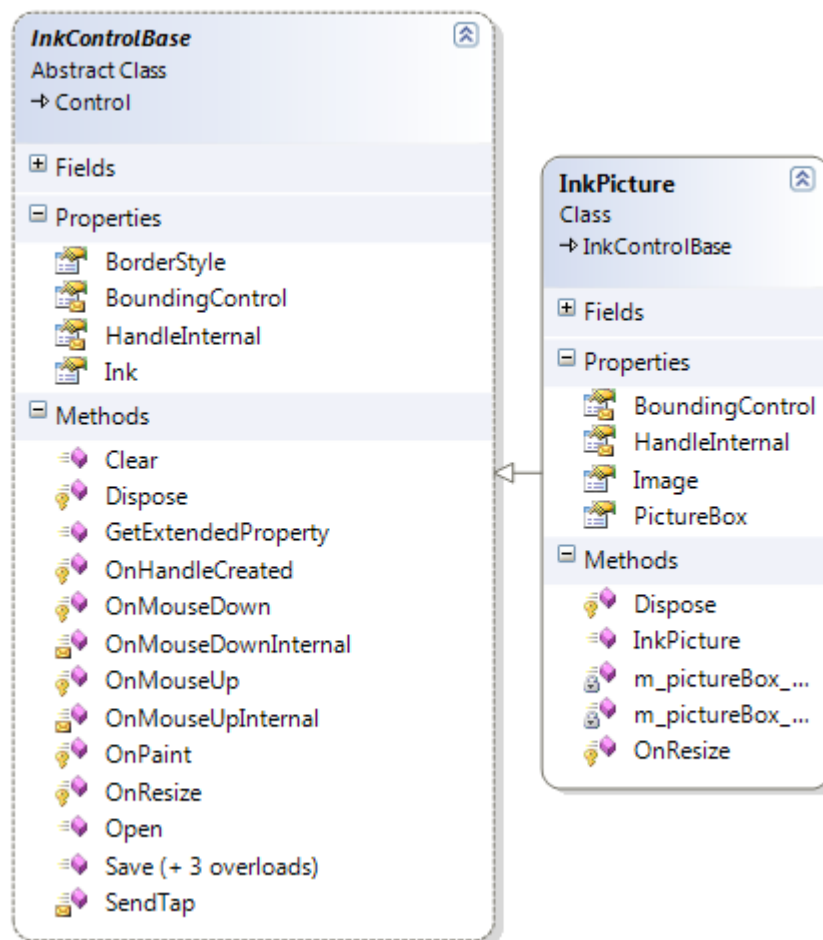
        //Return the recognized text
        return ret;
    }
}

```

From the code above you will notice that Ink property is used to access the IInkDisp, IInkStrokes and IInkRecognitionResults to return the recognized text. (For more information on text recognition see the [above](#)). The BasicRecognitionCF sample demonstrates the use of the InkRecognizer control.

InkPicture

InkPicture control allows the developer to add ink to a picture using the standard System.Windows.Forms.PictureBox control.



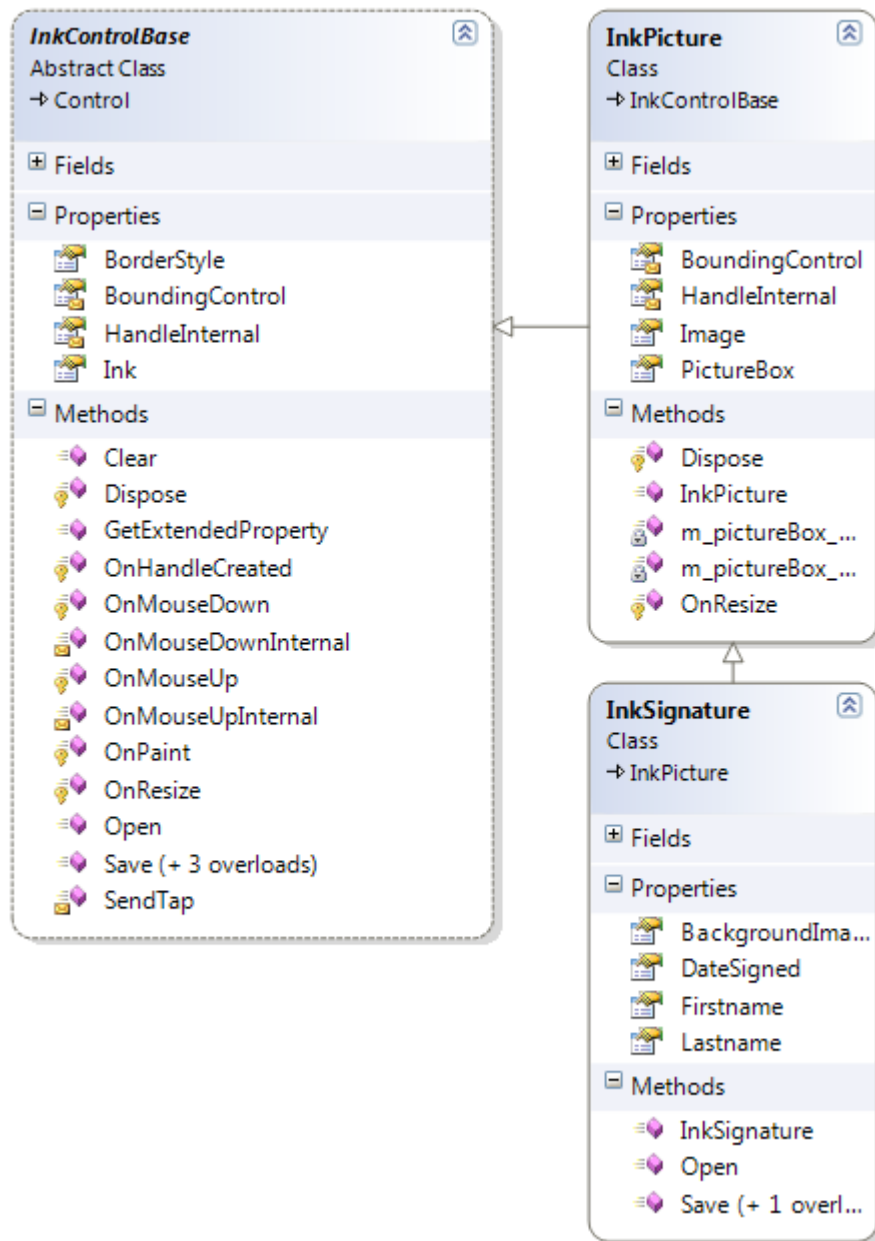
The InkPicture control is not as straight forward as the InkRecognizer.

The first difference is InkPicture overrides HandleInternal. If you recall, HandleInternal provides the handle to which the IInkOverlay object will be attached to. InkPicture internally has an m_pictureBox field which is the standard PictureBox control used to display an image and we use the m_pictureBox.Handle to attach to the IInkOverlay object.

The InkPicture control exposes the internal PictureBox control as a property to allow the developer to modify any properties for the picture box. It also exposes an Image property to get or set the image that is displayed in the PictureBox.

InkSignature

Capturing signatures is a very common requirement for Windows Mobile line of business applications and using and using the Windows Mobile Ink API (or WISP Lite) is a perfect solution. Since the InkPicture control has the main requirements for capturing signatures (which is really just capturing ink data) it can be used as the base class for InkSignature.



Since this is a signature control, we may want to give some sort of indication to the user to sign. To do this we can add an image to the control using the Image property. Below is an example of what that could possibly look like:



The InkSignature control also exposes a FirstName, LastName and DateSigned property. These class properties are saved as ExtendedProperties within the Ink Serialized Format (ISF) file when the signature data is saved. Be aware that these properties are only saved when the ink data is saved using ISF and won't be saved if saved to a GIF file. The InkSerializationCF Sample demonstrates the use of the InkSignature control.

InkControlBase Workarounds

During the development of these controls, it was found that InkOverlay behaved differently in managed code than it did in native code. Whenever a user started inking on the signature control, the InkSignature control could never lose focus by tapping on any part of the screen (for example the Textbox). Specifically, the Capture property of the control was set to True whenever a user started inking which caused other controls to not receive any Click events. It was also noticed that the InkControlBase control would never receive focus when clicked on because of InkOverlay.

To work around this a workaround was implemented in the OnMouseDown and OnMouseUp of the InkControlBase.

Focus Issue

First we'll look at OnMouseDown which resolves the 'focus' issue:

```
protected override void OnMouseDown(MouseEventArgs e)
{
    base.OnMouseDown(e);
    if (this.ClientRectangle.Contains(e.X, e.Y))
        OnMouseDownInternal(e);
}
```

From the code above you can see that on mouse down we check to see if we are in the bounds of the control, if we are we call OnMouseDownInternal().

```
internal virtual void OnMouseDownInternal(MouseEventArgs e)
{
    this.BoundingControl.Focus();
}
```

```
}
```

OnMouseDownInternal() will call the Focus() method on the BoundingControl. In the case of InkRecognizer it calls the InkControlBase implementation which returns this.

```
internal virtual Control BoundingControl
{
    get
    {
        return this;
    }
}
```

In the case of InkPicture and InkSignature, we override the BoundingControl property and return m_pictureBox.

```
internal override Control BoundingControl
{
    get
    {
        return this.m_pictureBox;
    }
}
```

This allows us to set focus to the underlying control and still have the control fire off events like GotFocus.

Capture Issue

The next issue faced was the Capture issue. It seemed IInkOverlay always set Capture to true for the bounded control. To work around this issue we had to override OnMouseUp as follows:

```
protected override void OnMouseUp(MouseEventArgs e)
{
    if (!this.ClientRectangle.Contains(e.X, e.Y))
        OnMouseUpInternal(e);

    base.OnMouseDown(e);
}
```

Again from the above code you can see that we call OnMouseUpInternal() where we set the Capture property to false and then send a 'manual tap' so the user does not have to tap the screen twice.

```
internal virtual void OnMouseUpInternal(MouseEventArgs e)
{
    //HACK When using the IInkOverlay object in managed
    //code seems that the control Capture property is set to true.
    //When trying to click outside the bounds of the
    //control (ie a button) the control would not
    //receive the mouse click
    this.BoundingControl.Capture = false;
    SendTap(Control.MousePosition.X, Control.MousePosition.Y);
}
```

The SendTap() method makes a native call to mouse_event to manually send another tap to the screen since the first tap (or click) was consumed by the InkBaseControl.

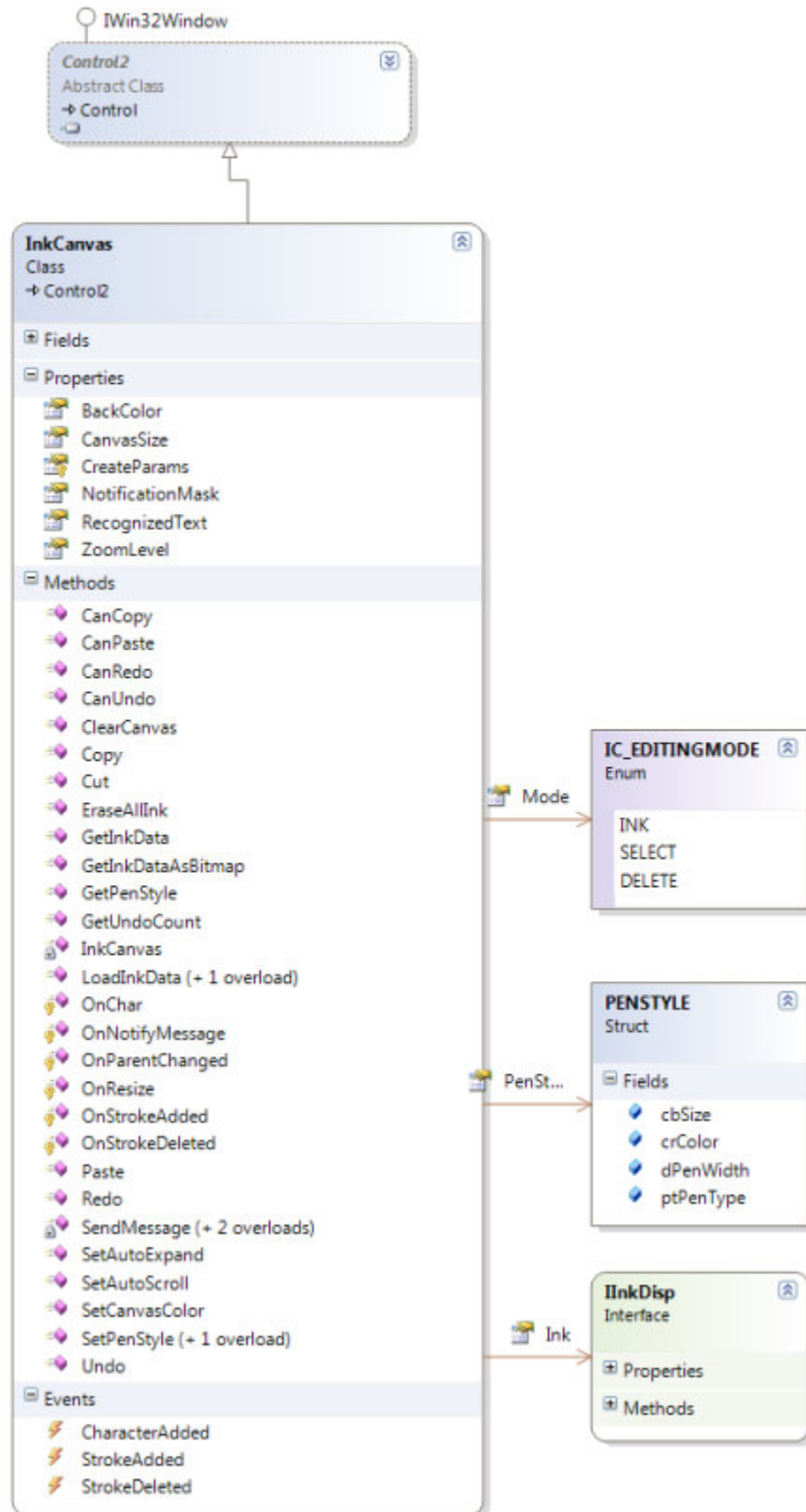
```
internal void SendTap(int x, int y)
{
    NativeMethods.mouse_event(
```

```
        NativeMethods.MOUSEEVENTF.LEFTDOWN |
NativeMethods.MOUSEEVENTF.ABSOLUTE,
        (int)((65535 / Screen.PrimaryScreen.Bounds.Width) * x),
        (int)((65535 / Screen.PrimaryScreen.Bounds.Height) * y),
        0,
        0);
    NativeMethods.mouse_event(NativeMethods.MOUSEEVENTF.LEFTUP, 0, 0, 0, 0);
}
```

InkCanvas

The InkCanvas control was discussed briefly in the technical overview but we will go through it again since it's a custom control part of the Mobile Ink Library. The InkCanvas control is a managed wrapper to the native InkCanvas control available on Windows Mobile 6. The InkCanvas control gives you the ability to add inking capabilities to your Compact Framework application on the Windows Mobile 6 platform without the need to use any of the COM interfaces.

Here is the class diagram of InkCanvas:



From the above class diagram you can see there are three different modes that the InkCanvas can be set to.

- INK - Sets the InkCanvas to inking mode and allows the collection of ink data
- SELECT - Sets the InkCanvas control select ink strokes collected
- DELETE - Sets the InkCanvas to delete ink strokes as the user taps on the stroke

There is also a PenStyle property which allows you to change the size, color, width and pen type of the stroke being collected. If you need to interact with the IInkDisp interface, you can also use the Ink property and use the COM interfaces available. The main method of communication with the native InkCanvas is via windows messages. Since the InkCanvas control inherits from the OpenNETCF.Windows.Forms.Control2 class, Control2 helps simplify the creation of native control and capture any messages such as StrokeAdded. This allows us to raise relevant events such as StrokeAdded, StrokeDeleted and CharacterAdded. The InkNotesCF Sample demonstrates the use of InkCanvas control.

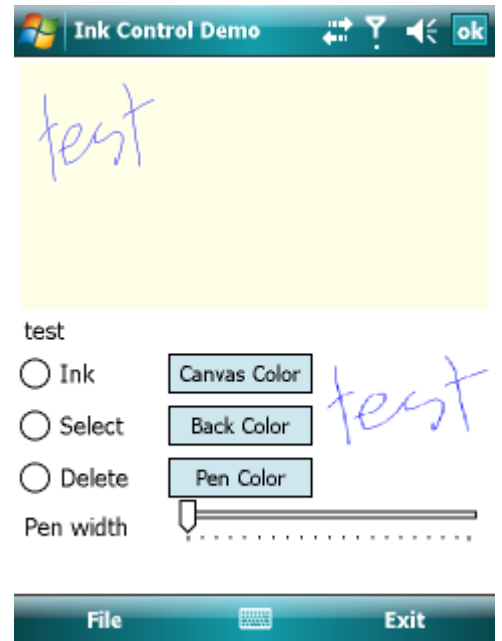
Samples Available with the Mobile Ink Library

Also included with the installation are five sample applications demonstrating the use of the library. The following is a brief description of the samples.

InkControlDemo

InkControlDemo demonstrates the various features of the InkCanvas control available in the Mobile Ink Library. It demonstrates how to:

- Add, select and delete ink data
- Change the color of the ink canvas, back color and pen color
- Change the width of the pen
- Text Recognition
- Extracting an image of the ISF ink data



InkObjDemo

InkObjDemo demonstrates the following:

- How to use the IInkOverlay COM interface within a managed application
- How to save the ink data as Ink Serialized Format (ISF) from the IInkOverlay COM Object
- How to load and Ink Serialized Format file into an IInkOverlay object
- Clearing the ink collected by IInkOverlay
- Changing the width of the pen for the ink



BasicRecognitionCF

BasicRecognitionCF was ported from the BasicRecognition native sample available with the Windows Mobile 6 Professional SDK. It allows a user to input text with the stylus, and recognize the text. It demonstrates the following:

- Using the InkRecognizer custom control
- Saving and Loading ISF files
- Ink recognition



InkSerializationCF

InkSerializationCF was ported from the InkSerialization native sample available with the Windows Mobile 6 Professional SDK. It allows a user to sign on a region and save the signature along with the persons name in the ISF file. It demonstrates using the following:

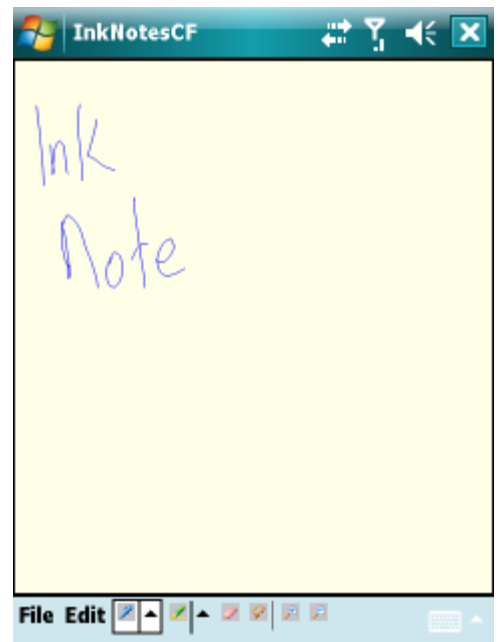
- Using the InkSignature custom control
- Saving and loading a signature file (ISF file)
- Embedding ExtendedProperties within an ISF binary stream



InkNotesCF

InkNotesCF was ported from the InkNotes native sample available with the Windows Mobile 6 Professional SDK. It allows a user to view and edit ink notes. These ink notes can also be imported to OneNote 2007 or 2003 on the desktop. It demonstrates using the following:

- InkCanvas control
- Changing the pen used for inking
- Erasing Ink
- Selecting Ink
- Zooming in and out
- Saving and loading ISF files
- Emailing the note as a GIF attachment



Conclusion

In this article we took a look at the Windows Mobile Ink API for Windows Mobile 6 and the OpenNETCF Mobile Ink Library. The Mobile Ink Library opens the doors to .NET Compact Framework developers to ink enable their Windows Mobile 6 applications. In the next article we'll look at using the same Ink Serialized Format (ISF) data on both a Windows Mobile 6 application and a desktop application.