

# Sharing Windows Mobile Ink with the Desktop

Mark Arteaga  
OpenNETCF Consulting  
September 2007

## Sharing Windows Mobile Ink with the Desktop

*Introduction*

*Ink Serialized Format Data*

*Sharing Ink With A Custom Application*

*Sharing Ink with One Note 2007*

*What's Next*

*Conclusion*

## Introduction

In the previous article [LINK] we discussed technical information on the Mobile Ink Library and WISP Lite. In this article we will discuss how the Ink Serialized Format (ISF) data can be shared on both a Windows Mobile 6 application and an application running on the desktop.

## Ink Serialized Format Data

Ink Serialized Format (or ISF) is the binary format in which the ink data is persisted as. This format is directly compatible to the Tablet PC ink format and can be interchanged on both the Tablet PC and a Windows Mobile 6 device. Using ISF data that was saved on a Tablet PC on a Windows Mobile device, Windows Mobile will ignore any properties in the ISF stream that are not supported but still preserve the properties when saved again.

As a developer using the Mobile Ink Library you have the option to persist the ink data as either as:

1. Ink Serialized Format - The ink data using ISF. This is the most compact representation of ink data.
2. Base64 Ink Serialized Format - The ink data using ISF and Base64 encoded.
3. Gif - The ink data in Graphics Interchange Format (Gif)
4. Base64 Gif - The ink data in Graphics Interchange Format (Gif) and Base64 encoded

In the next sections we'll look at sharing ISF data in a custom desktop application and a Windows Mobile application and also importing ISF data created on a Windows Mobile device into One Note 2007.

## Sharing Ink With A Custom Application

Since ISF data produced on the desktop is binary compatible on a Windows Mobile 6 Device and vice-versa I decided to take the common scenario of signature capture on a mobile device and display the data on a custom application written using the .NET Framework running on the desktop.

The Mobile Ink Library includes a sample called InkSerializationCF which captures a users signature and embeds an ExtendedProperty within the ISF stream. We will use this sample to show how the ISF data can be shared with the desktop.

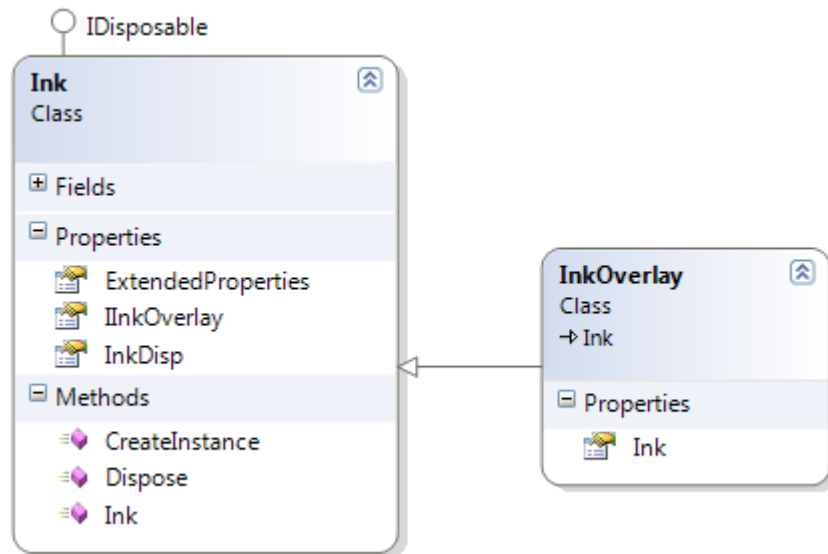
Note, if you want the try this sample on the desktop you will need to install a few SDKs. Here is a chart summarizing what is need on different operating systems.

Operating System	SDK Requirements
Windows Vista Windows XP Tablet PC 2005	<a href="#">Windows Vista SDK</a>
Windows XP	<a href="#">Windows XP Tablet PC SDK</a> <a href="#">Windows Vista SDK</a>

### Changes Since First Release

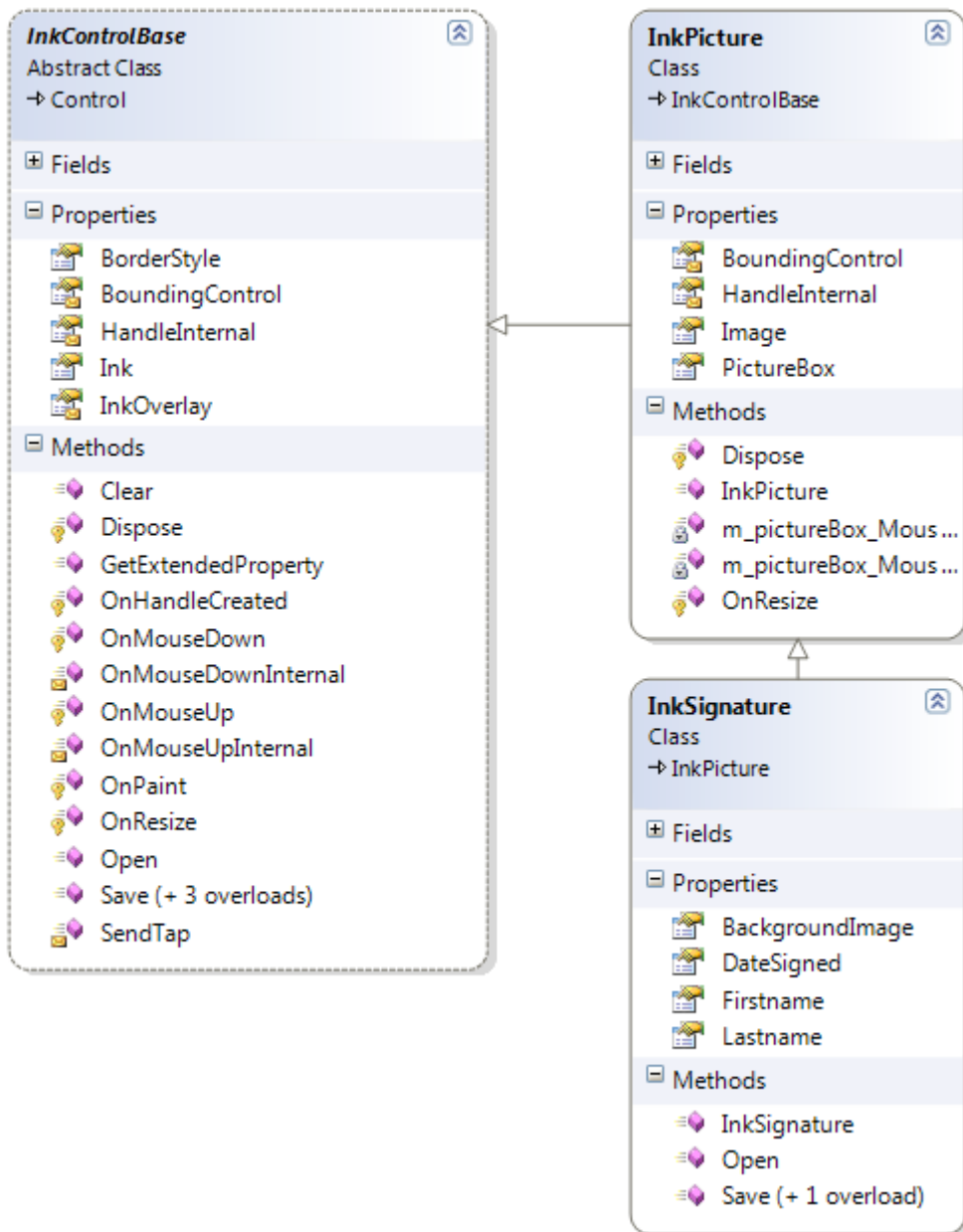
Since the Mobile Ink Library was first released, I have gone ahead and made some changes to align the classes more with the Microsoft.Ink classes available.

The first change was to the OpenNETCF.WindowsMobile.Ink.InkOverlay class. This class has been renamed to OpenNETCF.WindowsMobile.Ink.Ink class. The InkOverlay class has been marked obsolete and is still available.



In the above diagram you will notice in the OpenNETCF.WindowsMobile.Ink.Ink class we also added a new property called ExtendedProperties. These are the custom properties you can add to an ISF stream. In the case of the signature sample application we add the date the signature was saved, the first name and last name of the person signing.

Next is the InkControlBase class. We have added a new property called Ink which exposes the Ink object discussed above. Since we are using the OpenNETCF.WindowsMobile.Ink.InkPicture control as a base class for the OpenNETCF.WindowsMobile.Ink.InkSignature control and since InkPicture inherits from InkControlBase, this will help us keep the same object model as the Microsoft.Ink.InkPicture control. Here is a class diagram of the controls in the Mobile Ink Library:



Having a similar object model allows us to use the same InkSignature control on both the Windows Mobile application and the Desktop application.

### Sharing the InkSignature Control

Now that some appropriate changes have been made to the Mobile Ink Library classes to align with the Microsoft.Ink classes, we can now look at how we can share ISF data generated from a Windows Mobile 6 device within a desktop application.

The InkSignature control is a custom control that will capture signature data. It derives from InkPicture and since we change the classes to align more with the desktop we just have to change the internal implementation of InkSignature to be able to share the control on both the desktop and the device. (For more information on sharing code between .NET Compact Framework and .NET Framework see this [article](#) in [MSDN Magazine](#).) By using conditional compile statements we are able to have code in the same source file for both the device and the desktop.

To implement the code sharing we will be working with two separate solutions. The first is InkSerializationCF which is a .NET Compact Framework project for Windows Mobile 6 device. The second is InkSerializationFx which is the .NET Framework application that runs on the desktop. The InkSerializationFx application will only be used to view any signatures saved via the InkSerializationCF application.

The main change to accomplish the code sharing is to the InSignature.Open(string) method. Here we have used conditional compiles to allow us to use on both the desktop and device.

```
#if !Fx
public override void Open(string file)
#else
public void Open(string file)
#endif
{
#if !Fx
    base.Open(file);
#else
    this.InkEnabled = false;
    StreamReader sr = new StreamReader(file);
    byte[] data = Encoding.ASCII.GetBytes(sr.ReadToEnd());
    sr.Close();
    this.Ink = new Microsoft.Ink.Ink();
    this.Ink.Load(data);
    this.Refresh();
#endif

    //See if the ISF file opened is valid
    object ret = this.Ink.ExtendedProperties[new
Guid(m_sigValidationGuid)].Data;
    if (ret == null)
    {
#if !Fx
        this.Clear();
#else
        this.InkEnabled = false;
        this.Ink = new Microsoft.Ink.Ink();
        this.InkEnabled = true;
#endif
        this.m_firstname = "";
        this.m_lastname = "";
        this.m_dateSigned = DateTime.MinValue;
        throw new Exception("Invalid signature file!");
    }

    //Retrieve the firstname extended properties
    ret = this.Ink.ExtendedProperties[new Guid(m_firstNameGuid)].Data;

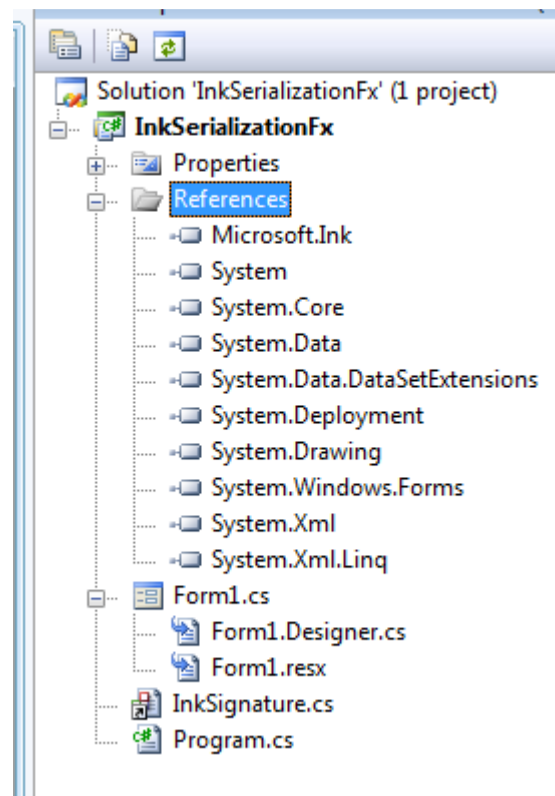
    this.m_firstname = ret.ToString();

    //Retrieve the lastName extended properties
    ret = this.Ink.ExtendedProperties[new Guid(m_lastNameGuid)].Data;
    this.m_lastname = ret.ToString();

    //Retrieve the date the sign was signed
    ret = this.Ink.ExtendedProperties[new Guid(m_dateSignedGuid)].Data;
    this.m_dateSigned = DateTime.FromFileTime(long.Parse(ret.ToString()));
}
}
```

The Open() method will extract the signature information, and all the ExtendedProperties that contains the date signed, and first and last name. From the above code you will notice not many changes are needed to be able to use the same source file on the desktop and on the device.

In the InkSerializationFx project, we need to add a reference to Microsoft.Ink (since OpenNETCF.WindowsMobile.Ink is not supported on the desktop).



Then we also have to add a using statement with a conditional compile so InkSerializationCF does not use it:

```
#if Fx
using Microsoft.Ink;
using System.IO;
#endif
```

A conditional compile statement was also added to the Save() method since the InkSerializationFx does not allow any modification of the signature data.

```
#if !Fx
public void Save(string filename, string firstName, string lastName,
InkPersistenceFormat format)
{
...
}
#endif
```

The InkSerializationCF sample was also modified to save the ISF data in different formats. One thing to note is when sharing the ISF data with the desktop, you can only use the ISFBase64 format as the regular ISF stream will not load.

After all this talk about code let's see the results. Here are the results of a signature on a Windows Mobile 6 device.



First Name

mark

Last Name

arteaga

Signature

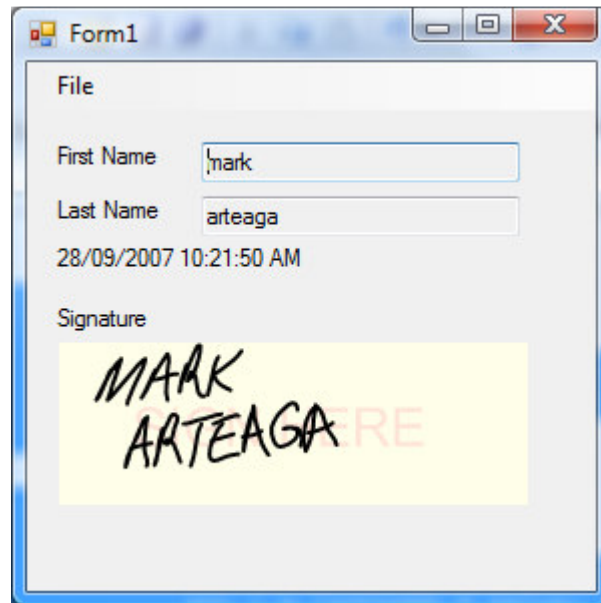


OK



File

And the results on the desktop.



For the desktop signature application, the signature is read-only as on the desktop you don't want the signature to be modified. You can add more ink to the ISF file and if you try to load it on the device, you will see those changes. I will leave it up to the reader to explore that scenario.

Now that you have knowledge on sharing ink data between a desktop application and a Windows Mobile device, next we will look at sharing the ISF data with One Note 2007.

## Sharing Ink with One Note 2007

Since ISF data is compatible on the desktop and a Windows Mobile 6 device, I decided to explore one of the comments in the 'Readme.txt' file in the Windows Mobile 6 SDK which said:

“These ink notes can also be imported to OneNote on the desktop.”

Well, not being a regular user of One Note but seeing the value this can possibly have in an application I decided to explore this comment a bit.

## Integration with OneNote

When InkNotesCF was first ported from the native sample it allowed the user to save the note in a certain Xml format as follows:

```
<?xml version="1.0"?>
<Import xmlns="http://schemas.microsoft.com/office/onenote/2004/import">
  <EnsurePage path="Pocket Notes\InkFiles.one" guid="3855d272-28de-4a3f-b157-4eca51c5d15a" title="\Storage Card\note2003.ink" />
  <PlaceObjects pagePath="Pocket Notes\InkFiles.one" pageGuid="476d389f-a48b-4891-b665-148a6ed7d9d1">
    <Object guid="">
      <Position x="0" y="0" />
      <Ink>
        <Data></Data>
      </Ink>
    </Object>
  </PlaceObjects>
</Import>
```

As you can see from the Xml namespace the version of OneNote that was used when the native sample was created was OneNote 2003. This Xml format was also ported to the managed InkNoteCF sample.

With the release of OneNote 2007, the COM API has changed which also resulted in a new Xml schema to import ink data into OneNote. We will not go through the changes but for more information on OneNote 2007 from a developers perspective see [‘What’s New for Developers in OneNote 2007 \(Part 1\)’](#) and [‘What’s New for Developers in OneNote 2007 \(Part 2\)’](#)

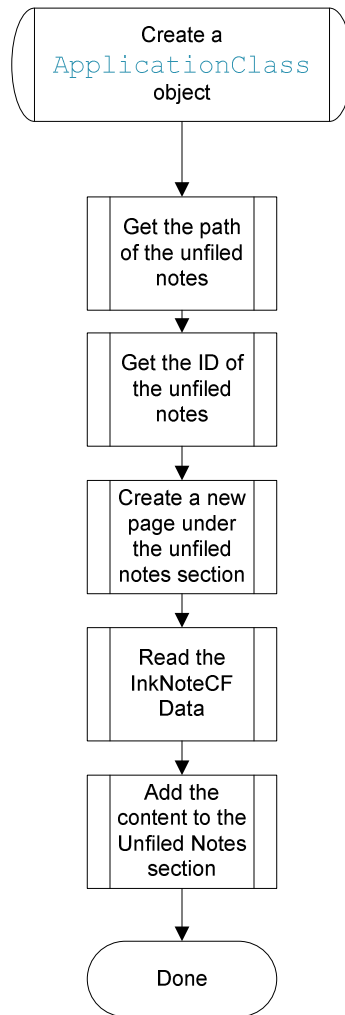
The new Xml structure for importing data into one is as follows:

```
<one:Page
xmlns:one="http://schemas.microsoft.com/office/onenote/2007/onenote"
ID="{0}">
  <one:Title>
    <one:OE author="Mark Arteaga" lastModifiedBy="Mark Arteaga">
      <one:T><![CDATA[\Storage Card\note2003.ink]]></one:T>
    </one:OE>
  </one:Title>
  <one:Outline>
    <one:OEChildren>
      <one:OE>
        <one:InkParagraph>
          <one:InkWord>
            <one:Data></one:Data>
          </one:InkWord>
        </one:InkParagraph>
      </one:OE>
    </one:OEChildren>
  </one:Outline>
</one:Page>
```

As you can see, the new Xml format (or schema) has changed from OneNote 2003. If you would like to explore the schemas you can download the [‘2007 Office System: XML Schema Reference’](#). I’ll leave it up to the reader to explore the schemas and other technical information.

To import the ink data into OneNote you will need to use the OneNote COM API. In the sample program that imports the data we use a C# application and it automatically generates a interop assembly called Microsoft.Office.Interop.OneNote. (for more information on this see [this section](#) in the [‘What’s New for Developers in OneNote 2007 \(Part 1\)’](#) article)

The basic flow to create a new note in OneNote is this:



Within the import application, the ink data generated by InkNoteCF is saved to the ‘Unfiled Notes’ section of OneNote. The actual code looks like this:

```

Microsoft.Office.Interop.OneNote.ApplicationClass onApp = new
ApplicationClass();
try
{
    //Get the path for the unfiled notes
    string unfiledNotesPath;
    onApp.GetSpecialLocation(SpecialLocation.slUnfiledNotesSection, out
unfiledNotesPath);

    //Get the ID of the unfiled notes
    string UnfiledNotesID;
    onApp.OpenHierarchy(unfiledNotesPath, "", out UnfiledNotesID,
CreateFileType.cftNone);

    //Create a new page
    string pageID;
    onApp.CreateNewPage(UnfiledNotesID, out pageID,
Microsoft.Office.Interop.OneNote.NewPageStyle.npsBlankPageWithTitle);

    //Read the file data
    StreamReader sr = new StreamReader(txtInkNote.Text);
    string data = sr.ReadToEnd();
  
```

```
data = string.Format(data, pageID);

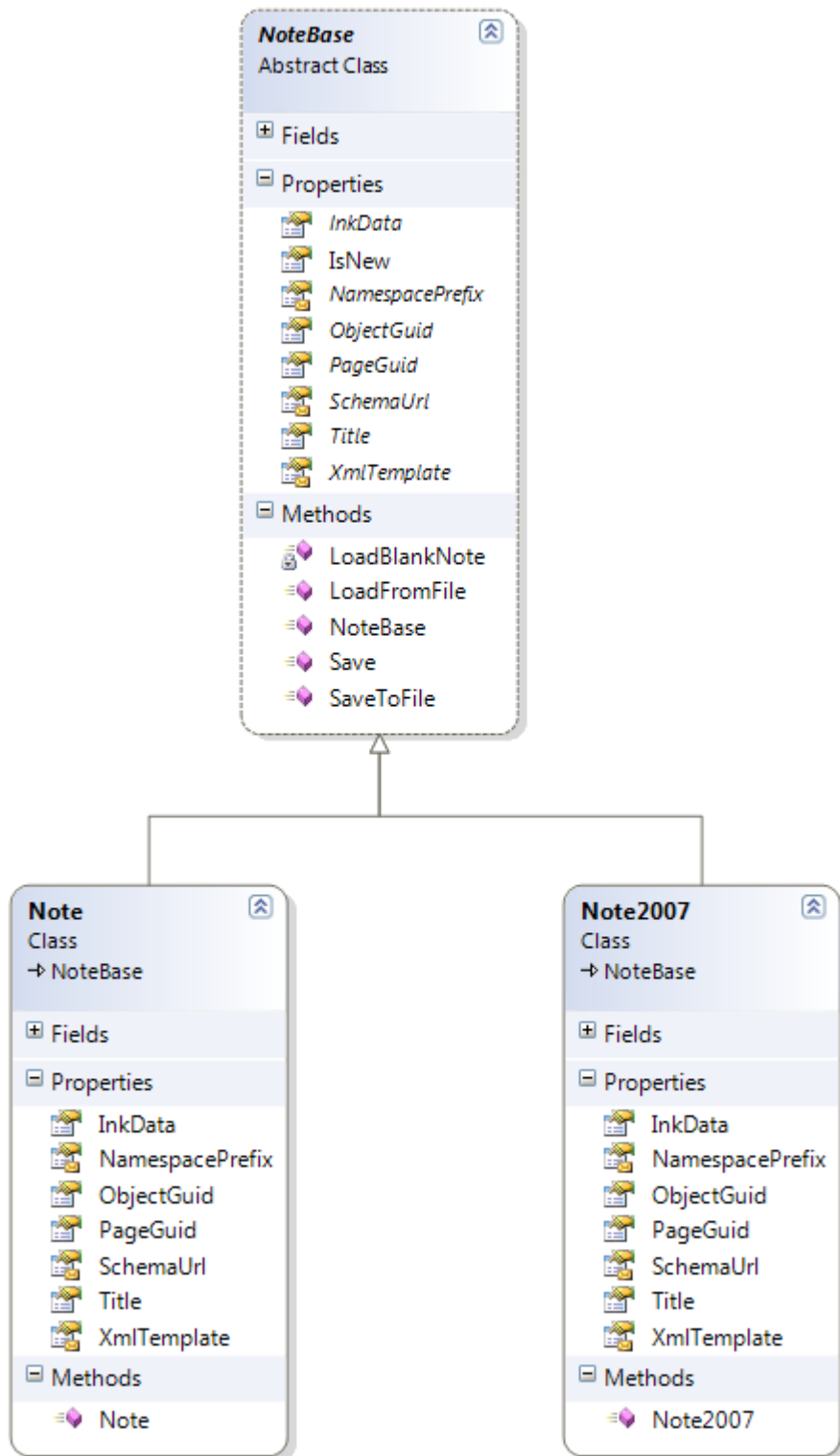
//Add the content to the one note page
onApp.UpdatePageContent(data, DateTime.MinValue);
}
catch (Exception ex)
{
    MessageBox.Show(ex.ToString());
}
}
```

As you can see from the code, it's not very difficult to import data into OneNote 2007.

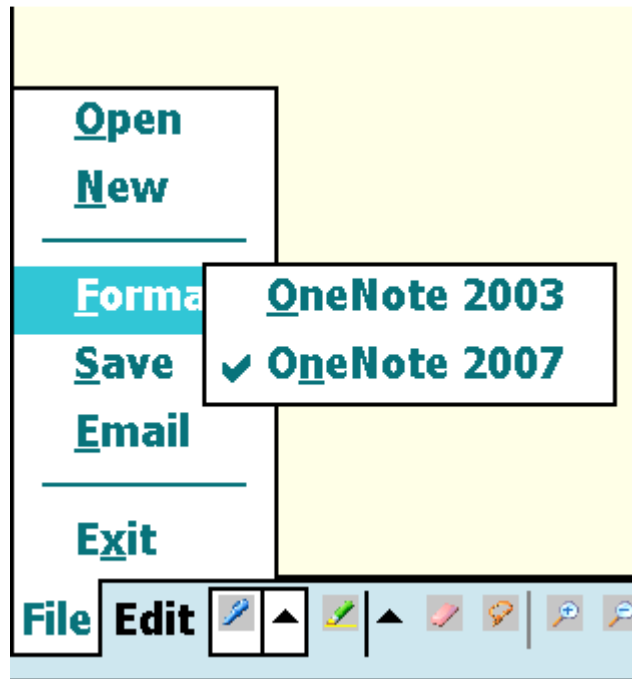
### Changes To InkNoteCF

When the InkNoteCF sample was first released, it only supported saving the data in OneNote 2003 format. Since I did not have OneNote 2003 and only had 2007, I refactored the code to allow the user to save in either the OneNote 2003 or OneNote 2007 format. In this section we will go over some of the changes done. (Note, if you are interested in importing the ink to OneNote 2003 see [this article](#) on MSDN)

The first thing that was modified was the Note class. The original one only had support for OneNote 2003 so a new NoteBase class was created to accommodate the two different versions. The new class hierarchy is show in the following diagram.



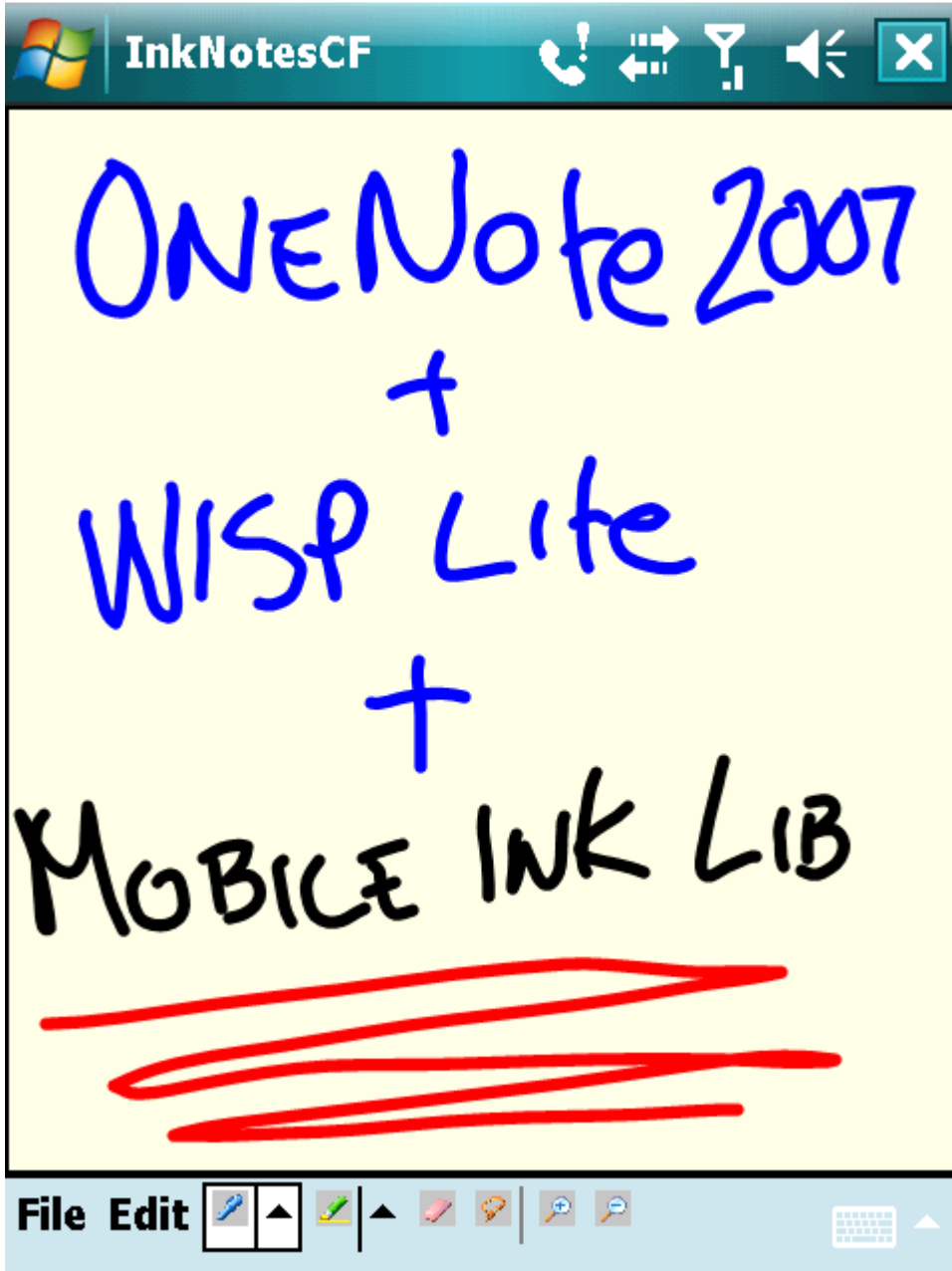
In the main user interface, a new menu item called 'Format' was added to allow the user to save the ink note as either OneNote 2003 or OneNote 2007 format.



I'll leave it up to the user to delve into the code and to see how the saving code works. In a nutshell, it just calls the `XmlDocument.Save()` method.

### Running the Code

After all this background information you are probably thinking, what is the result? Well, here are the screen shots of the application on the device and in OneNote 2007.



ONENote 2007

+

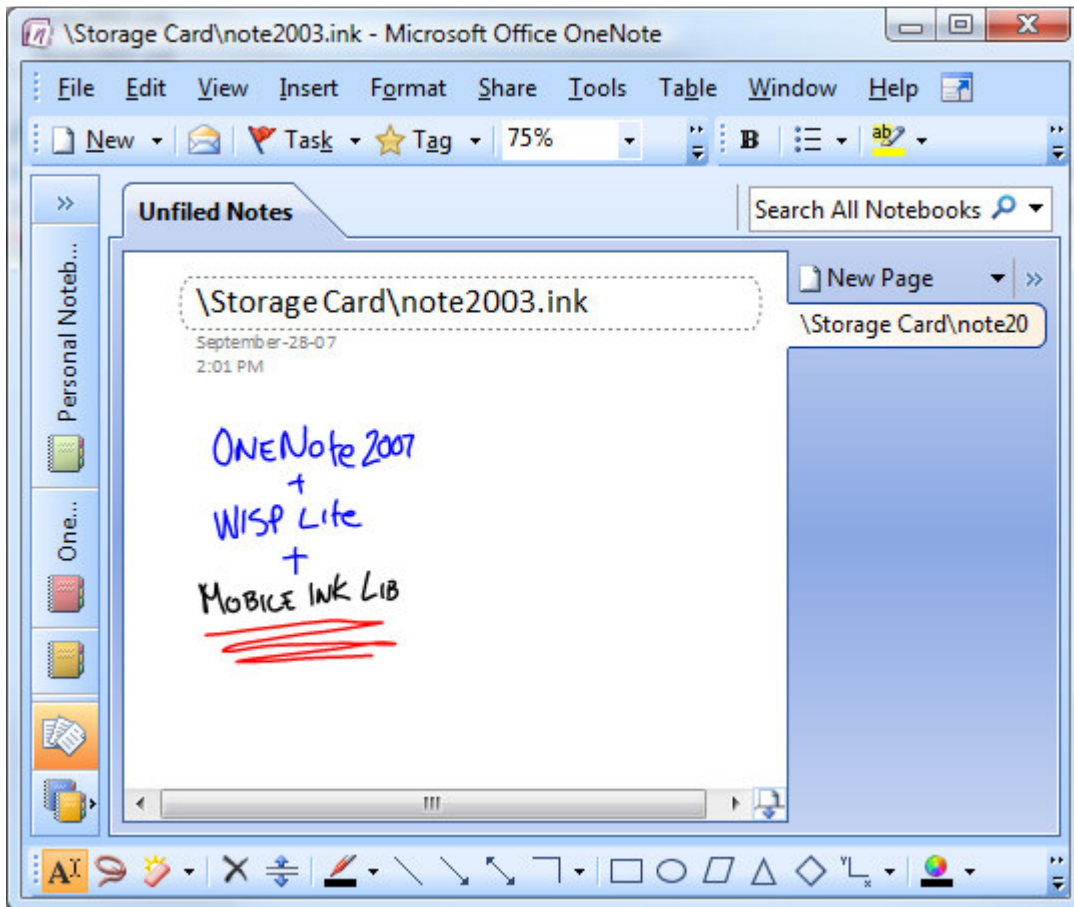
WISP Lite

+

MOBILE INK LIB

File Edit





## What's Next

From here the Mobile Ink Library is in a good state to be used in managed applications. We are hoping to improve it some more by wrapping some of the WISP Lite interfaces and align the object model more closely with the desktop. At this point we have no plans to do that but invite the community to contribute to the project. You can download the source code for Mobile Ink Library and the Samples at [www.opennetcf.com/opensource/mobileink.ocf](http://www.opennetcf.com/opensource/mobileink.ocf).

## Conclusion

The new WISP Lite API available in Windows Mobile 6 gives the native developer the ability to 'ink enable' a Windows Mobile application. Using the Mobile Ink Library from OpenNETCF, .NET Compact Framework developers can now also 'ink enable' their applications. In this article we discussed how using the Mobile Ink Library, you can easily share ink data with a custom application or with OneNote 2007. Have fun Inking!