Building Tablet PC Applications with Visual FoxPro

Mike Stewart, Visual FoxPro Team, Microsoft Corporation

Learn how to use the Microsoft® Tablet PC Platform SDK in Visual FoxPro applications to work with "digital ink", accept handwritten input, handle the events raised by Tablet PC controls, and place the "ink" in save files permanently.

introduction

The controls and APIs required to take advantage of many of the Ink features of Microsoft Windows® Tablet PC Edition can be found in the Tablet PC Platform SDK. You can download the Tablet PC Platform SDK from http://www.micro soft.com/ windowsxp/tabletpc/developers/ Download default.asp. It provides the Ink objects in two ways: via the Component Object Model (COM) Automation and via the Common Language Runtime (CLR) of the Microsoft .NET Framework via managed APIs. Because Visual FoxPro does not compile to the CLR, this article focuses on the COM Automation APIs.

The Tablet PC views Ink differently than previous devices, which also included Ink as a functionality. Earlier devices like the Apple Newton or Microsoft Windows CE/ Pocket PC devices focused on recognizing ink as handwriting. With the Tablet PC, this philosophy has changed so that ink is now considered data input, converting to text is considered a secondary focus. For example, you can take notes in Windows Journal, thanks to the included tablet PC and Microsoft's OneNote, without the need to convert those notes to text. This means that the tablet PC's recognition capabilities are very good, and that in case you need to convert ink to text sen, a very high recognition rate is achieved.

The objects of interest to VFP developers are InkEdit, InkPicture, and InkOverlay. The InkEdit control corresponds to the VFP edit box. The difference is that InkEdit accepts and recognizes digital ink. The InkPicture control is exactly what you expect: an image can be placed in this control and the user can paint on the image with the digital ink. The control lies invisibly over a window and can accept and process Ink.

System requirements

- **Microsoft Tablet PC Platform SDK**
- **Microsoft Windows XP.** You can also use the Tablet PC Platform SDK on Microsoft Windows XP, but Microsoft XP Tablet PC Edition is highly recommended.
- **Microsoft Visual FoxPro 8.0 or higher here.** It is also possible with earlier versions of VFP to use almost all functionalities of the Tablet PC Platform SDK. However, various bugs affecting the cooperation with the Microsoft XP Tablet PC Edition have been eliminated in VFP 8.0.
- **Digitizers.** Various manufacturers offer digitizers for the USB interface

to, for example Wacom (http:// www.wacom.com) or applications can be developed that use the built-in digitizer of a Tablet PCx.

A) Only on a tablet PC?

The Tablet PC Platform SDK only requires Microsoft Windows XP. However, the functionality of the controls is significantly expanded through the use of Windows XP Tablet PC Edition. If the SDL is not installed on a Tablet PC, there will be no recognition engine and some controls will not accept Ink. So it is advisable to have access to a Tablet PC, at least for testing and debugging.

Ideally, you can code and debug your applications on a tablet PC. But who wants to write code with a pen? The keyboards that some Tablet PCs have don't have either the full size. There are various possibilities to circumvent this problem. One option is to use a USB keyboard and mouse (or use a docking station if one is available for your tablet PC). Another option is to use Windows XP's Remote Desktop feature to access the Tablet PC from a desktop computer.

design decisions

A common question is, "How do I know if my application will run on a Tablet PC?" You can get the answer by calling the Win32® function GetSystemMetrics with the parameter SM_TABLETPC(86). A non-zero return value indicates that Windows XP Tablet PC Edition is running; the return value 0 indicates that this is not the case.

```
#define SM_TABLETPC 86
Declare Integer GetSystemMetrics in Win32API Integer retVal =
GetSystemMetrics(SM_TABLETPC)

If retVal <> 0
        Wait Window "Running on a Tablet PC"
else
        Wait Window "*Not* running on a Tablet PC"
final
```

However, a return value other than 0 does not yet indicate whether all components of the Tablet PC are installed and working. To determine whether a component in is installed, try to create an instance of the component and check for errors during the creation.

An example:

```
Try
        oInkEdit = Newobject("InkEd.InkEdit.2")
Catch To oException Wait
        Window "InkEdit control is not installed!"
end try
```

A Tablet PC can have two display modes: landscape and portrait. Landscape format is mostly used when the tablet PC is docked or as standard laptop is used; in portrait format it is used when used as a tablet.

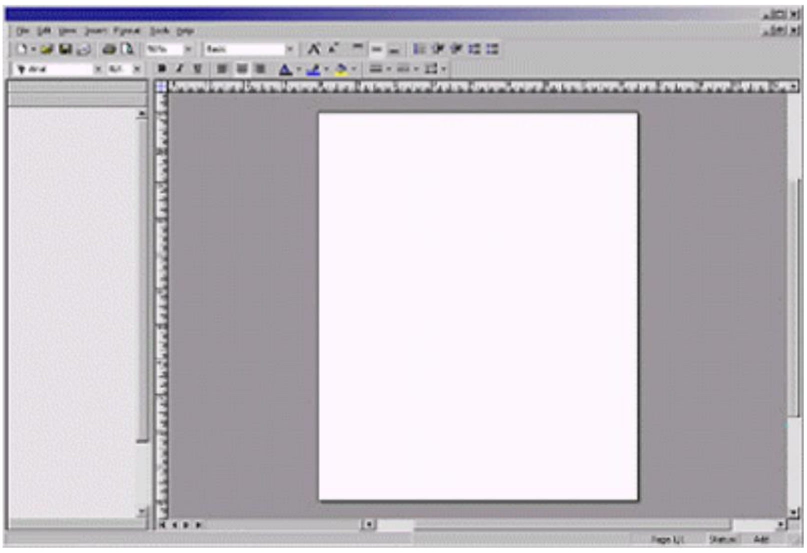


Figure 1. The landscape format.

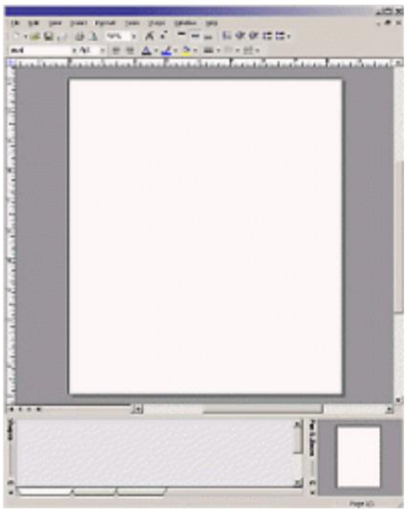


Figure 2. The portrait format.

Your application may need to determine whether the user's Tablet PC is running in portrait or landscape mode so that the screen elements are resized accordingly. By using VFP's SYSMETRIC() function with parameters 1 or 2, you can determine the width and height of the display. If the width is less than the height, the tablet PC is in portrait format.

There are other decisions you must make when designing your application. Read the entry "Planning Your Application" in the Tablet PC SDK help file.

collect ink

The simplest way to accept pen input is to use the InkCollector object provided by the Tablet PC Platform SDK. It's extremely easy to deploy, requiring just three simple steps to set up and activate. InkCollector is added to a form via the form's Windows Handle (hWnd) and allows the entire form to be pen-enabled.

B) Ink Collection on one enable form

1. Create a new object with the InkCollector.
2. Add the InkCollector to the form's Windows handle.
3. Enable the InkCollector by setting its Enabled property to .T.

put.

Run the following code and start typing.

```
Local oInkCollector as MSINKAUT.inkcollector.1 Local oForm1 as Form oForm1
= NewObject("form")


*-- Create a new InkCollector, and set the *-- window handle to be the form
on which *-- we want to work with the InkCollector oInkCollector =
NewObject("msinkaut.inkcollector.1") oInkCollector.hWnd = oForm1. HWnd


*-- That's it, we're done setting it up. Enable *-- the InkCollector so we can start
using it.
oInkCollector.Enabled = .t.

*-- InkCollector is turned on, show the form oForm1.Show Read Events
```

Note that this is the Tablet PC specific code:
oInkCollector = NewObject("msinkaut.inkcollector.1")
oInkCollector.hWnd = oForm1.HWnd

If you've used Windows Journal, you might expect to be able to mark, change, and delete ink in a Tablet PC application. Using the Ink Collector object, you can draw on your form, accept what is written, and then erase the ink. However, you cannot use the InkCollector method in your VFP application to mark, modify, or delete ink. You use the In kOverlay class for this. InkOverlay is an advanced InkCollector so you have full functionality.

C) The InkOverlay class

Using the InkOverlay class is as easy as using the InkCollector class. Due to the additional functionalities, however, there is a lot more to consider with this class. Let's start with a simple form that includes InkOverlay and that adds the ability to see what the user writes. Note that to activate the Recognize button you have to press Alt-R. You will soon find out why this is so.

```
*---------------------------------------------------
*--          Program: InkOl1.prg
*--
*--          Author:             Mike Stewart
*--          Comments: Simple InkOverlay demo using an InkOverlay class with a
*--                               recognizer button.
*--
*---------------------------------------------------
PUBLIC oform1

oform1=NEWOBJECT("InkOverlayDemo1") oform1.Show

RETURN

DEFINE CLASS InkOverlayDemo1 AS form

    Top = 0 Left
    = 0 Height =
    416 Width = 659
    DoCreate = .T.

    Caption = "Form1"
    Name="Form1"

    ADD OBJECT inkRecognize AS Commandbutton WITH ;
```

```
                top = 368, ;
                Left = 20, ;
                Caption = "\<Recognize ", ;
                Name="Inkbutton1"

        ADD OBJECT edtrecognized AS editbox WITH ;
                Height = 121, ;
                Left = 20, ;
                top = 236, ;
                Width = 589, ;
                name="edtRecognized"

        PROCEDURE Init
                PUBLIC oink As msinkaut.inkoverlay.1 oInk =
                NEWOBJECT("msinkaut.inkoverlay.1")

                WITH oInk *--
                        Point it to the window for which you *-- want to capture ink.
                        Because VFP does *-- not have hWnds for individual controls,
                        *-- this can only be done at the form level. .hwnd = thisform.HWnd



                        *-- The Attachmode property *-- determines
                        whether the
                        *-- InkOverlay sits in front *-- or behind the
                        controls on
                        *-- the shape.
                         .AttachMode = 0 && IOAM_Behind

                        *-- Set everything before enabling, *-- or else error
                        occurs. .enabled = 1


*-- Set to collect both ink and gestures
                 .CollectionMode = 2 && ICM_InkAndGesture
                ENDWITH
        END PROC

        PROCEDURE inkRecognize.Click
        thisform.edtRecognized.value = ;
        oInk.ink.strokes.tostring
        END PROC
FINAL DEFINE
```

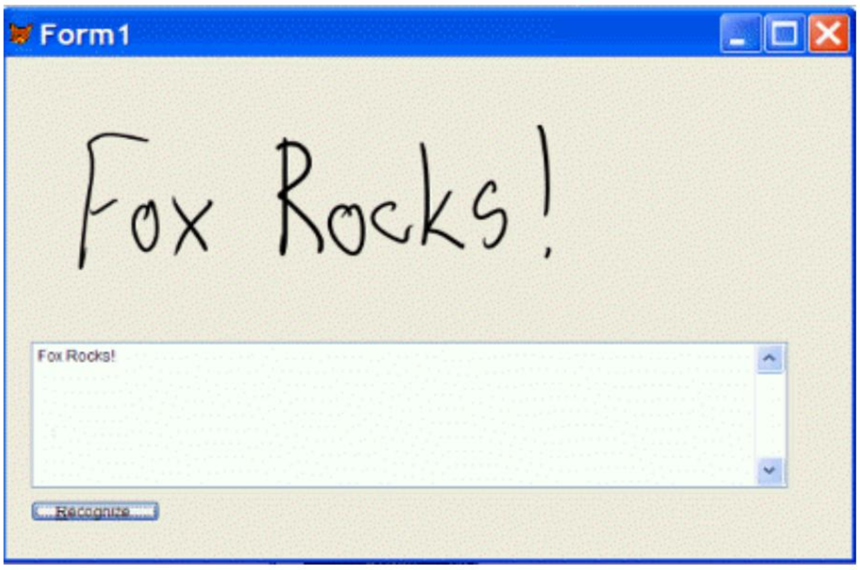


Figure 3. The form with the InkOverlay class after the Recognize button is clicked.

With just 38 lines of code, we have a working form that accepts Ink and has the ability to recognize the input and write it to a VFP control that is bound to a field and writes its value to a table can be.

If you run the code above, you will have noticed that it was difficult to click the Recognize button with the pen or the mouse.
This is because when you add the In kOverlay object to it, the entire form is spanned. This behavior corresponds to that of the InkCollector class. Open minded

Apparently, this doesn't work for most applications, since the majority of forms contain text boxes and other control elements in which the user's input is recorded.

There is a simple solution to this problem. It consists in defining a rectangle for InkOverlay. The Tablet PC Platform SDK provides the InkRectangle object and the In kOverlay.SetWindowInputRectangle method for just this purpose.
The changes to the above code are minimal. First, create the SetRectangle method using the following code.

```
Procedure SetRectangle LOCAL
oInkRectangle as MSINKAUT.inkrectangle oInkrectangle =
NEWOBJECT("msinkaut.inkrectangle")

*-- Set the bottom of the rectangle *-- to be the top of
the edit box.
*-- Doing this means the InkOverlay *-- will not overlap
our controls. oInkRectangle.Bottom =
thisform.edtRecognized.Top

*-- The rest of the dimensions of the *-- rectangle will
match those of the *-- form. oinkrectangle.Top =
thisform.Top oinkrectangle.Left = thisform.Left
oinkrectangle.Right = thisform.Width - thisform.Left


*-- Call the method by passing our InkRectangle *-- and we're done.

ThisForm.InkOverlay.SetWindowInputRectangle(oInkRectangle)
```

Then call this method from within the form's Init method:

```
 .AttachMode = 1 && IOAM_InFront
this.setrectangle
```

Run the code and you'll notice that the input range for the pen is now constrained to the edit box area.

Another solution is to set the InkOverlay.AttachMode property to 0 (IOAM_Behind). This will place In kOverlay behind the controls on the form so you can use the controls as normal. A problem with this solution is that the user can write between the controls, but not on them.

A well-defined inking surface using the SetInputWindowTec tangle method could be the better solution for your users.

Selecting and Deleting Ink

The following code shows how to add a combo box for choosing between three cursors:

- Ink Cursor – for drawing and To write.
- Selection Lasso - used for marking Ink.
- Eraser - used for erasing ink.

First, add a combobox to the class definition:

```
ADD OBJECT inkPenType AS combobox WITH ;
     RowSourceType = 1, ;
     RowSource = "Ink,Select,Delete", ;
     Height = 24, ;
     Left = 36, ;
     top = 428, ;
     Width = 168, ;
     name="inkPenType"
```

Now add the following code to the combo box's InteractiveChange event to set the EditingMode property based on the user's selection: PROCEDURE inkPenType.InteractiveChange

```
     DO case
     CASE this.Value = "Ink"
          thisform.inkoverlay.EditingMode = 0 && IOEM_Ink
     CASE this.Value = "Select"
          thisform.inkoverlay.EditingMode = 2 && IOEM_Select
     CASE this.Value = "Delete"
          thisform.inkoverlay.EditingMode = 1 && IOEM_Delete *-- Lastly, set the EraserMode based
          on *-- whether entire strokes should be erased *-- when the pen touches the screen, *-- or
          if pixels should be erased: *- - Set EraserMode to 0 to erase strokes,

          *--                                  or 1 to erase pixels *--
          thisform.inkoverlay.EraserMode = 0 thisform.inkoverlay.EraserMode
          = 1
     ENDCASE
END PROC
```

By selecting the appropriate option in the combo box, the user can now change the behavior of the pen.

Saving and loading Ink

The user probably wants to save what he has written and reload it at a later point in time.

First of all, let's add some buttons to the class definition of the form, one for saving the

Inks and one to load Ink into the form:

```
ADD OBJECT cmdSave AS commandbutton WITH ;
     top = 360, ;
     Left = 357, ;
     Height = 27, ;
     Width = 84, ;
     Caption = "\<Save", ;
     Name="cmdSave"

ADD OBJECT cmdLoad AS commandbutton WITH ;
     top = 360, ;
     Left = 456, ;
     Height = 27, ;
     Width = 84, ;
     Caption = "L\<load", ;
     Name="cmdLoad"
```

Now add code to the Save event of the button:

```
PROCEDURE cmdSave.Click lsInk =
     ThisForm.inkoverlay.Ink.Save()
     STRTOFILE(lsInk, GETFILE("isf"))
END PROC
```

The InkOverlay's Ink object uses its Save method to return a byte array representing the Ink on the InkOverlay. Using FoxPro's STRTOFILE() function, this byte array

(the

FoxPro treats it as a string) can then be saved to disk.

To reload the saved Ink object using the Load button, use the Ink.Load method:

```
PROCEDURE cmdLoad.Click
        lsInk = CREATEBINARY(FILETOSTR(GETFILE("isf")))
        thisform.inkoverlay.Enabled = 0 thisform.inkoverlay.ink.Load(lsInk)
        thisform.inkoverlay.Enabled = 1


        *--Tell control to redraw, or changes will not be made visible.
        *-- Use the InkRectangle object we *-- created
        during form Init.
        ThisForm.inkoverlay.Draw(ThisForm.InkRectangle)
END PROC
```

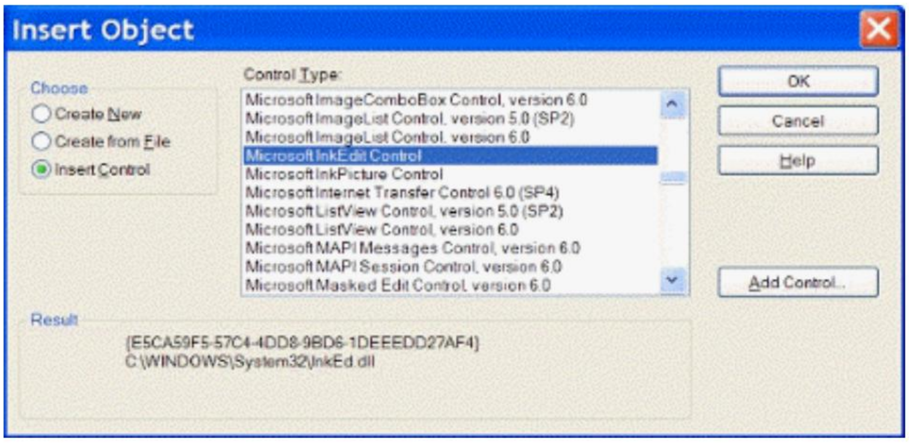
Note the use of the CREATEBINARY() function when reading the ink file into a FoxPro variable. This is required to put them in a format that the InkOverlay object will accept. Also note that the InkOverlay must first be disabled before the Ink is loaded. Otherwise an error would occur.

Try this: save some ink to a file, exit the form, run it again and load the ink you created. Now click the Recognize button. The InkO verlay recognizes the Ink! This is because the InkOverlay not only stores the location of the pixels so it can be viewed again later, but it also stores the metadata needed for recognition.

What is included in this metadata? The file's metadata includes information such as the date and time stamp, as well as a unique identifier for each stitch.

The InkEdit control

The InkEdit control is an advanced RichText control. It was designed to provide an easy way to accept, view, and recognize Ink. To implement an InkEdit control on a VFP form, drag an ActiveX control from the form's Controls toolbar. When prompted, highlight the InkEdit control.



Choose
- Create New
- Create from File
- [✓ Insert Control]

Control Type:
- Microsoft ImageComboBox Control, version 6.0
- Microsoft ImageList Control, version 5.0 (SP2)
- Microsoft ImageList Control, version 6.0
- Microsoft InkEdit Control
- Microsoft InkPicture Control
- Microsoft Internet Transfer Control 6.0 (SP4)
- Microsoft ListView Control, version 5.0 (SP2)
- Microsoft ListView Control, version 6.0
- Microsoft MAPI Messages Control, version 6.0
- Microsoft MAPI Session Control, version 6.0
- Microsoft Masked Edit Control, version 6.0

Result
{E5CA59F5-57C4-4DD8-9BD6-1DEEEDD27AF4}
C:\WINDOWS\System32\InkEd.dll

Figure 4. Adding a Microsoft InkEdit control to a form.

When you run the form, the InkEdit control accepts the ink and recognizes it after the allotted time of 200 milliseconds. You can change the time-to-recognition using the InkEdit control's RecognitionTimeout property. InkEdit has a default value of Ole control1 in the Text property, but you can easily change this later.

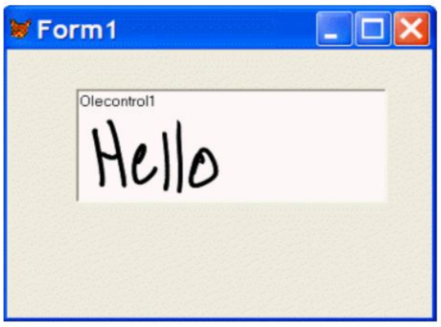


Figure 5. Before detection.

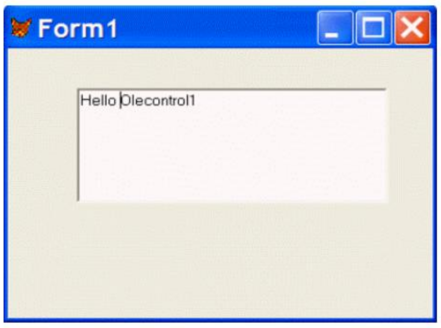


Figure 6. After detection.

Because the InkEdit control is based on the RichText control, users can type in the In kEdit control just as they would in the standard text box or edit box. By using the control

InkEdit allows you to create applications that give the user the option of entering data either with the keyboard or with the pen.

The InkPicture control

The InkPicture control combines most of the attributes of the InkOverlay control with the ability to overlay ink over a graphics file. The following code demonstrates how to create a form with an InkPicture control and allows the user to select the graphic to load.

The InkPicture control can also be dragged onto a form just like an InkEdit control. In Figure 4, it's the control just below the InkEdit

.

```
PUBLIC ofrminkpicture

ofrminkpicture=NEWOBJECT("frminkpicture") ofrminkpicture.Show

RETURN

DEFINE CLASS frminkpicture AS form
        Top = 0
        Left = 0
        Height = 460
        Width = 469
        DoCreate = .T.
        Caption = "InkPicture Form"
        AllowOutput = .F.
        name="frmInkPicture"

        ADD OBJECT olecontrol1 AS olecontrol WITH ;
                top = 12, ; Left =
                7, ; Height = 444, ;
                Width = 360, ; OleClass
                = "msinkaut.inkpicture.1"

                OleLCID = "1033"
                name="olecontrol1"

        ADD OBJECT cmdload AS commandbutton WITH ;
                top = 24, ;
                Left = 374, ;
                Height = 27, ;
                Width = 84, ;
                Caption = "\<Load Image", ;
                Name="cmdLoad"

        ADD OBJECT cmdcolor AS commandbutton WITH ;
                top = 60, ;
                Left = 374, ;
                Height = 27, ;
                Width = 84, ;
                Caption = "\<Color", ;
                name="cmdColor"


        PROCEDURE cmdLoad.Click WITH
                thisform.olecontrol1
                        Try
                                .Picture =;
LoadPicture;
 (GetFile("bmp;jpg;jpeg;gif",;
 "Graphics files"))
                        Catch To oException
                                MessageBox("No file selected")
                        end try
                ENDWITH
        END PROC

        PROCEDURE cmdcolor.Click
                thisform.olecontrol1.DefaultDrawingAttributes.Color =;
GetColor()
        END PROC
FINAL DEFINE
```

The OleControl is defined in the following code part: OleClass = "msinkaut.inkpicture.1"
OleLCID = "1033"

After the control is defined, it gives the user the option to load a graphic file into the Click method of the Load button. There is only one time for that le code required. Of course, you can also include error handling in the code.

```
PROCEDURE cmdLoad.Click
WITH thisform.olecontrol1 Try

                .Picture =;
LoadPicture;
 (GetFile("bmp;jpg;jpeg;gif",; "Graphics files"))

        Catch To oException
                MessageBox("No file selected")
        end try
ENDWITH
END PROC
```

You can also add a Color button to allow the user to change the color of the ink other This ability is also available in the InkOverlay control.

```
PROCEDURE cmdcolor.Click
thisform.olecontrol1.DefaultDrawingAttributes.Color =; GetColor()

END PROC
```

The DefaultDrawingAttributes collection offers a variety of ways to change the way the Ink is drawn. In this example, Color is the only element of the collection, and the code simply calls FoxPro's GETCOLOR() function to take the user's choice of color.

Like the InkOverlay control, the InkPicture control can recognize, save, and load ink. The code for accomplishing this task with the InkOverlay control goes directly through the InkPicture control.

Summary

These are just some of the things you can do on the Tablet PC with the Tablet PC SDK and Visual FoxPro. The purpose of this article is to make you aware of the options available to you when writing applications for the Tablet PC. For a complete listing of the properties, events, and methods of the controls discussed here, see the Tablet PC SDK Help file.