

The Evolution of Tablet PC Technologies in Microsoft Windows Vista

Although a relatively new platform, **Tablet PC** continues to revitalize and broaden its appeal to developers. By expanding existing features, adding new investments, and simplifying deployments, developing for **Tablet PC** becomes even more alluring with **Windows Vista**.

What's Coming in Windows Vista?

In Windows Vista, Tablet PC shifts into the mainstream. Ink becomes more ubiquitous, integrating directly into the presentation subsystem in Windows Presentation Foundation. Ink Analysis offers new and exciting ways to parse handwriting. There are more APIs for COM developers. Touch screens enter into the Tablet PC realm. Tablet PC Input Panel makes some APIs accessible and integrates new functionality. All Tablet PC Platform components support partial trust. Tablet PC technologies deliver native 64 bit platform components. And Tablet PC technologies are ubiquitous across all Windows Vista editions.

Windows Presentation Foundation Integration

Tablet PC's integration into the WinFX™ Runtime Components isn't just about bringing ink to mainstream applications. If you develop for Tablet PC you'll find a deep synergy between Windows Presentation Foundation and Tablet PC values. The flowed-layout features make it possible to design UI that works well in either portrait or landscape orientation, and Windows Presentation Foundation's device-independent coordinate space makes it easier to work with a wide variety of display technologies, accommodating the broad spectrum of Tablet PC form factors. XAML opens the door to creative UI design, including customization of the look of the entire palette of standard framework controls—and 3-D animation!

In Windows Vista, there are three parallel technology stacks for the Tablet PC platform: COM, Windows Forms, and the new Windows Presentation Foundation. The COM and the Windows Forms object models are similar. The Windows Presentation Foundation object model is somewhat different. The core Tablet PC concepts continue to be the same, though: stylus, ink, rendering, recognition, and ink controls.

Digital ink collection and rendering—features traditionally available only on the Tablet PC Platform—have been incorporated into Windows Presentation Foundation as first-class members of the framework. You need no separate runtime dependencies to support basic ink functionality in your applications—ink is available everywhere WinFX is installed.

Hello, <InkCanvas>!

The central tier of Tablet PC support in Windows Presentation Foundation is the <InkCanvas> element. Thanks to the flexible layout, rendering, and hit-testing inherent in all Windows

Presentation Foundation framework elements, InkCanvas encapsulates all of the various modes of operation provided by the InkCollector, InkPicture, and InkOverlay controls of the classic Tablet PC Platform API. In many cases, adding support for ink to your Windows Presentation Foundation application is as simple as adding an <InkCanvas> element to your markup language (code-named "XAML"). Users without Tablet PCs are able to view others' ink annotations and, by using the mouse in lieu of a stylus, they can scribble themselves. Simply create a new Windows Presentation Foundation Application in Visual Studio 2005, edit the skeleton XAML by replacing the <Grid> tags with an <InkCanvas /> tag, and, Voila!

Once instantiated, you make use of a number of elements to control InkCanvas object's appearance and behavior. For example, you can enable annotation on a background image by setting the Background property, of type Brush:

```
<InkCanvas xmlns="http://schemas.microsoft.com/winfx/avalon/2005">
<InkCanvas.Background>
<ImageBrush ImageSource="myBackground.jpg"/>
</InkCanvas.Background>
</InkCanvas>
```

One of the fundamental ways Windows Presentation Foundation differs from the Win32 user interface model is in how sibling elements can overlap or be contained within one another's boundaries. To create the functionality of a classic InkCollector that collects and renders ink behind child windows, simply rearrange the Windows Presentation Foundation elements as siblings, with appropriate relative z-ordering so that the child appears on top of the InkCanvas:

```
<Window xmlns="http://schemas.microsoft.com/winfx/avalon/2005">
<InkCanvas />
<TextBox /> <!-- ink will appear behind this textbox -->
</Window>
```

A Richer User Experience

Windows Presentation Foundation also integrates rich media into the user experience. For example, you can use InkCanvas to implement real-time ink collection and rendering over full motion video. Simply wrap a MediaElement child. This was very difficult to accomplish using the traditional platform. Windows Presentation Foundation unifies all aspects of the presentation tier, so the following XAML just works:

```
<Window xmlns="http://schemas.microsoft.com/winfx/avalon/2005">
<InkCanvas>
<MediaElement Source="myMovie.wmf" />
</InkCanvas>
</Window>
```

Likewise, the selection-mode behaviors of InkCanvas are extended to work with all Windows Presentation Foundation child elements, not just ink strokes. This makes it trivially easy to implement complex mixed-ink-and-text editing features similar to those found in Windows Journal and Microsoft Office OneNote®. The following XAML defines a selection-mode editing surface where the two child elements can be selectively resized and repositioned alongside ink content:

```
<Page xmlns="http://schemas.microsoft.com/winfx/avalon/2005">
<InkCanvas EditingMode="Select" >
<TextBox ... />
<MediaElement ... />
</InkCanvas>
</Page>
```

A Reformulated Object Model

The ink object model has been redesigned based on developers' feedback (This new Stroke/StrokeCollection object model resides in the System.Windows.Ink namespace). Many difficult scenarios (such as undo/redo functionality for point-erase operations on strokes) are significantly easier. Note that the recognition APIs have largely been factored out of the Windows Presentation Foundation object model (only the gesture recognizer remains) in favor of the new and improved Ink Analysis SDK

Although the object model has changed slightly, ink serialization format (ISF) has not. Windows Presentation Foundation ink continues to interoperate with classic platform ink, by virtue of their common serialization format.

Building Your Own Stylus Input

All elements deriving from UIElement expose a set of stylus-related events, as well as keyboard and mouse events. The stylus events are similar to mouse events but they provide richer information from the stylus device (or touch screen) and fire at a higher frequency. This enables developers to forego InkCanvas and build their own ink collection and rendering surfaces from scratch, a flexibility required of most professional design and illustration software.

The UIElement stylus events bubble-up and tunnel-down through the element tree, like all routed events in the Windows Presentation Foundation framework, and they originate from the threading context of the source UIElement in order to mitigate race conditions with their corresponding mouse events. For applications with higher fidelity needs—such as real-time rendering of ink—a plug-in model exists whereby applications can arrange to receive stylus input data on a background thread. This is similar to the RealTimeStylus (RTS) plug-in model.

Ink Analysis: The Unified Recognition API

Ink Analysis fuses ink to text recognition and ink layout recognition to improve accuracy, and the APIs are available to all three of the technology stacks.

To date, developers programmatically interpret ink by using two separate APIs of the Tablet PC Platform SDK:

- InkDivider—Identifies writing from drawing and groups basic grammatical units of strokes.
- RecognizerContext—Converts ink to text for several supported languages.

Each of these APIs are separate, yet functionally related. Both analyze what the user has written.

The Ink Analysis APIs are a functional superset of the InkDivider and RecognizerContext APIs and remain very similar. For example, to just get to the recognized strings, a Windows Forms application can do so in four steps:

```
// Create a new InkAnalyzer and an association to a single Ink object.
// Multiple ink objects cannot be referenced from a single InkAnalyzer.
// In this example, the Ink belongs to an InkOverlay object.
myInkAnalyzer = new InkAnalyzer(myInkOverlay.Ink);

// Add some or all of the strokes to be analyzed to the InkAnalyzer object.
myInkAnalyzer.AddStrokes(myInkOverlay.Ink.Strokes);

// To Analyze the strokes, call the InkAnalyzer.Analyze method.
myInkAnalyzer.Analyze();

// Once the results are finished, you can simply access
// the recognized string value by calling the GetRecognizedString method.
string results = myInkAnalyzer.GetRecognizedString();
```

InkAnalysis Tree Structure

The Ink Analysis APIs give you access to the full set of results as a dynamic tree. This tree consists of two main types of data: ContextNode objects, which describe the semantic classification of the strokes, and AnalysisHints objects, which enable developers to provide contextual clues to scope down the possible classification and recognition values for a region of input.

The InkAnalyzer uses several types of context nodes to illustrate the structure, such as:

- WritingRegionNode - Represents regions of strokes with similar paragraphs.
- ParagraphNode - Represents the strokes for several similar handwritten lines.
- LineNode - Represents the strokes for a single handwritten line.
- InkWord - Represents the strokes for a single handwritten word or character.

Tap-to-Expand-Selection Example

Consider the scenario in which the user taps to select various levels of ink, one where the selection grows incrementally based on the number of taps.

# of Taps	Selected Structure
1	Stroke
2	Ink Word
3	Line
4	Paragraph
5	Writing Region

Use the groupings from the InkAnalyzer to adjust the selection. For one tap, hit test to find the appropriate stroke. For two taps, look for the strokes within the InkAnalyzer's collection of InkWord ContextNode objects.

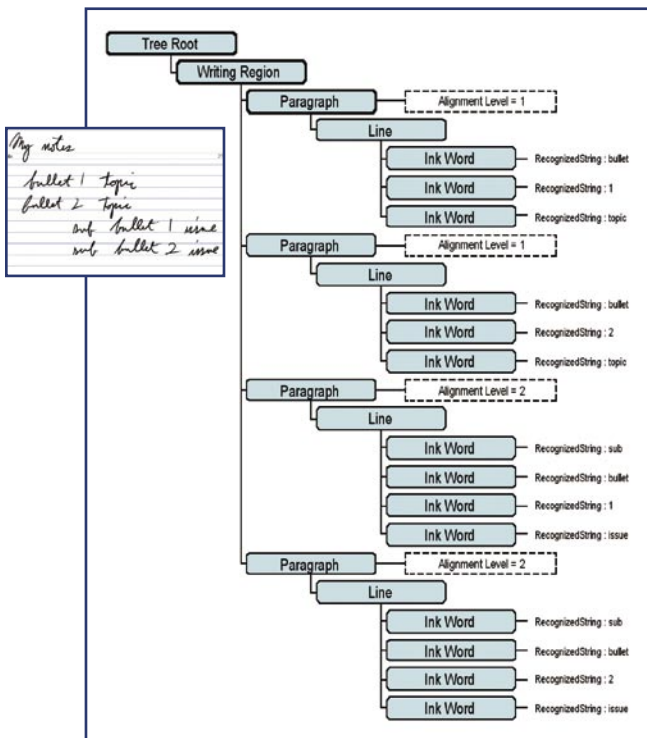
```
ContextNodeCollection inkWordNodes;
inkWordNodes = analyzer.FindNodesOfType(ContextNodeType.InkWord,
tappedStroke);
```

If inkWordNodes is not empty, the context node also references any other strokes that make up the handwritten ink word. You then set the selected strokes to the InkWord's Strokes collection.

ContextNode Properties

Each ContextNode contains type-specific properties. The recognized string is the most obvious property exposed on writing type context nodes, but there are several others such as shape names, rotated bounding boxes, and alternates.

One such property, AlignmentLevel, exists only on the Paragraph context nodes. This property shows the hierarchy for an outline. For any given Paragraph object an AlignmentLevel property indicates the indents each paragraph has within a WritingRegion. The following diagram shows a results structure and AlignmentLevel property values for a small bulleted list.



The combination of the analysis results tree and AlignmentLevel property enables you to work with the nested relations in an ink outline. You can select any level of the outlined lists. For instance, you can select the second bullet and all nested sub-bullets. This detailed access to the results provides rich ink features such as hiding and showing a sub-bullet item within an outline.

Analysis Hints

Analysis hints are special ContextNode objects—AnalysisHintNode objects—within the analysis tree, created by the application. Each AnalysisHintNode has a Location property, which indicates what area of the page the analysis hint should be applied. The application can scope the expected input down by setting properties like WordList or Factoids or writing Guide structures and the likes on the AnalysisHintNode. When the InkAnalyzer analyzes some ink, it checks for user-created AnalysisHintNode objects within the tree. If any hint nodes intersect the location of the stroke(s) being analyzed, the InkAnalyzer uses the defined contextual information in the hint node when analyzing the strokes. A great use for this is in form-filling applications.

For each input field, you can easily create the AnalysisHintNode, setting the appropriate properties on it.

```
// Create a rectangle to represent the bounds of this hint.
Rectangle hintBounds = new Rectangle(3969, 2646, 17198, 3440);

// Construct the hint based on those bounds.
AnalysisHintNode hint =
    analyzer.CreateAnalysisHint(new AnalysisRegion(hintBounds));

// Set the hint's name.
hint.Name = "1.Time";

// Set the factoid to be Time. Recognizers
// interpret whatever appears in this region as a time (1:00, 2:00, etc.)
hint.Factoid = Factoid.Time;
```

In addition to increasing recognition accuracy, the AnalysisHintNode also tells what strokes associated with the area occupied by the AnalysisHintNode. You get quick and accurate recognition results for an input field.

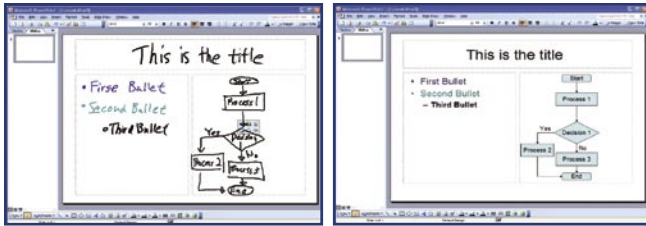
```
// Get the time field AnalysisHint ContextNodes
ContextNodeCollection analysHints = analyzer.GetAnalysisHints("1.Time");

// Get the first hint with name "1.Time" out of the
// collection of hints and cast to an AnalysisHintNode
AnalysisHintNode hint = (AnalysisHintNode)analysHints[0];

// Get the Recognized string value out of the hint
string result = hint.GetRecognizedString();
```

Putting it All Together: Hand-Drawn Slides

You see the power of Ink Analysis when you combine features like shape recognition, outline interpretation, and analysis hints to create tools where users can convert hand-written slide decks to clean, professional presentations.



COM Real Time Stylus

The managed RealTimeStylus API exposes the raw data stream coming from digitizers at a low level, providing full control of the pen and how it interacts with applications. Many developers asked for a COM version that would not require interoperability with the .NET Framework. We are delivering the COM RealTimeStylus implementation in Windows Vista.

Touch Input

To date, Tablet PC technologies have optimized for electromagnetic digitizers. Many of you requested Tablet PC capabilities for touch digitizers (resistive and capacitive). We are expanding support to these digitizers in Windows Vista. There are unique usability issues when using touch digitizers, such as fingers blocking the accurate positioning of the cursor and lack of hover capabilities. To address these, we created a new UI element that enables you to drive Windows with a finger. We support touch digitizers at the API level as well. Thus, any existing stylus application can take advantage of input from a touch digitizer.

New Input Panel APIs

In Windows Vista, we have deprecated the IPenInputPanel API and replaced it with the new and improved ITextInputPanel API. The ITextInputPanel API overcomes several of the IPenInputPanel API's shortcomings and enables the following new scenarios in addition to the previously supported IPenInputPanel scenarios:

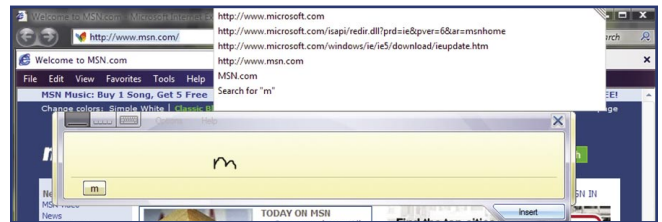
- Absolute positioning of the in-place Tablet PC Input Panel
- Forcing Input Panel to show in its fully expanded state, in addition to the hover target state
- Selecting any of three Input Panel input areas (Lined Pad, Character Pad, or Keyboard)
- Collecting the Ink objects entered through Input Panel, in addition to the recognized text

Additionally, we added a completely new IHandWrittenTextInsertion API, enabling an application to mimic text insertions through Input Panel. Thus, users can enjoy Input Panel's correction experience in a custom inking surface or even the keyboard.

Input Panel AutoComplete Integration APIs

Autocompletion of text vastly speeds up input when applications can access the functionality. This is especially true for Tablet PC. In Windows Vista, Input Panel includes a new API set that enables applications to integrate their autocomplete lists directly with Input Panel. Access your Most Frequently Used (MFU), Most Recently Used (MRU), and other types of auto complete lists to speed and

enhance recognition accuracy. By using the ITipAutocompleteClient and ITipAutocompleteProvider interfaces—implemented by Input Panel and the application, respectively—an application repositions an auto complete list so that it appears relative to Input Panel. You can also update lists as a user writes in Input Panel.



Extended Partial Trust (Windows Journal Reader Extension)

When we released the Windows Journal Reader Supplemental Component in 2004, it did not support partial trust scenarios. Our aspiration has always been to fully support partial trust for all Tablet PC platform components. In Windows Vista, we have again reached the state where all our components support partial trust.

Native 64 Bit Components

64 bit computing is becoming ubiquitous. Tablet PC technologies fully align with this by delivering native 64 bit platform components enabling 64 bit ink applications.

Enabled Unicode Ranges

We have added the ability to set and get the enabled Unicode ranges on a RecognizerContext object for both COM and managed versions. By using the new IInkRecoContext2 interface, you can limit the expected Unicode ranges for recognition. For example, CHS/CHT recognizers currently support over 10,000 characters, but only 1,000 of those are common. Use the SetUnicodeRanges function to limit the set of recognized characters to the 1,000 common ones, improving accuracy.

Available for All Windows Vista Editions

To date, developers have had problems redistributing Tablet PC technology. In part to address this, Tablet PC developer redistributables will be included in all Windows Vista editions. For all future Windows releases, Microsoft continues to drive towards the direction of making Tablet PC technology ubiquitous.

Conclusion

A lot of changes are occurring throughout the developer ecosystem. Tablet PC is rolling into the Windows Vista era with new features, greater integration, and better deployment possibilities. We're hoping that these changes enable developing scenarios for the Tablet PC Platform that are exciting, effective, profitable, and enjoyable.