Using Ink Services in Your Application

Carbon > Events & Other Input

2003-07-24

Contents

Figures, Tables, and Listings

Introduction to Using Ink Services in Your Application

Ink supports text input using a stylus and a graphics tablet. When a user prints text with the stylus, Ink processes the text and flows the recognized text to the current insertion point, just as if the user had typed it on a keyboard. When users work with a graphics tablet, there's no need to put down the stylus and return to the keyboard just to enter a title, caption, or filename. Users can also write keyboard equivalents with the stylus that enable them to open and close windows, and otherwise control an application without lifting the stylus from the tablet. Users can turn Ink handwriting recognition on or off, control where inking is permitted, and enable or disable recognition of predefined editing gestures.

When Ink was introduced in Mac OS X version 10.2, it provided automatic support for Ink input into applications. As long as the user enables Ink in System Preferences, an application receives Ink input as text without needing any modifications. As English text is written on a tablet, it is automatically recognized and entered as a stream of key down events into a document or text field.

Note: Ink is available in Carbon and Cocoa applications. It is not supported in Classic applications.

With the introduction of the Ink Services application programming interface (API) in Mac OS X version 10.3, developers can further integrate Ink into their application and create novel solutions for Mac OS X. Some of the key features the new API provides are:

- The ability to programmatically enable or disable handwriting recognition
- Access to a list of alternate interpretations for Ink input
- Support for deferred recognition and recognition on demand
- Support for direct manipulation of text using gestures
- Access to Ink data at multiple levels (points and recognized text)

With the release of Mac OS X version 10.3, the Ink recognition engine supports English, French, and German. The language that is recognized depends on the user's language setting in the Ink pane of System Preferences.

Who Should Read This Document

Any developer whose application receives text input should read this document to find out how the Ink Services API can benefit an application. The document is of most benefit to developers who have specific needs related to Ink input. Some specialized needs include the following situations:

- You want to implement a handwriting recognition solution for an input device. For example, you are a hardware vendor who wants to supply end-user software for your tablet or other piece of hardware.
- You need to provide a customized solution for recognizing the end of a phrase or word. For example, you want all input in a given window to be treated as a single phrase.
- You are writing a text editor and want to provide support for direct manipulation of text with Ink gestures.

- You want to implement a correction model. For example, you want to provide users with a list of alternate interpretations for a word.

Organization of This Document

The remainder of this document is organized into the following chapters:

- "Ink Services Concepts" (page 9), describes the Ink user interface, provides an overview of how Ink works in Mac OS X, and introduces the key concepts you need to understand the Ink Services API.
- "Ink Services Tasks" (page 25), provides information on how to accomplish the most common programming tasks using the Ink Services API.
- "Glossary" (page 41), defines Ink terminology.

The ideal way to proceed through the document depends on your programming experience and the tasks you want Ink Services to perform. Because the Ink Services API is new to Mac OS X, all developers should first read "Ink Services Concepts" (page 9). If you need more than the automatic support provided by Ink Services, you should read the how-to sections in "Ink Services Tasks" (page 25) that seem most appropriate to your application.

See Also

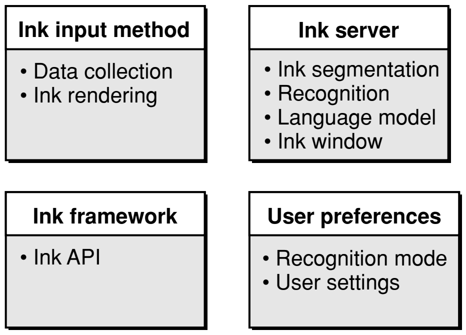
If you plan to use the Ink Services API in your application, you should read *Ink Services Reference*, as this document provides a complete reference to the Ink Services API.

Ink Services Concepts

Ink input is available for any application that accepts text input, as long as an Ink input device is connected to the computer and the user turns on handwriting recognition in the Ink pane of System Preferences. An application doesn't need to perform any tasks to support Ink input. However, there may be special situations for which an application needs to use the Ink Services application programming interface (API) to provide a customized Ink-input solution. After reading this chapter, you should be able to determine whether your application needs to use the Ink Services API.

Ink has four components (shown in Figure 1-1)—Ink input method, framework, server, and user preferences. The Ink input method is the component responsible for collecting Ink, drawing Ink, managing phrase termination, and posting Ink events. The Ink framework provides the application programming interface and services for the other Ink components. The Ink server manages the recognizer (including character and word segmentation), the language model used for recognition, and the Ink window that can be used for Ink input. The user preferences component manages the user settings that control the Ink recognition mode as well as a variety of options that can be set by the user.

Figure 1-1 Ink components



The remainder of this chapter provides an overview of the Ink user interface, describes how Ink technology works in Mac OS X, and discusses the concepts you need to understand the Ink Services API.

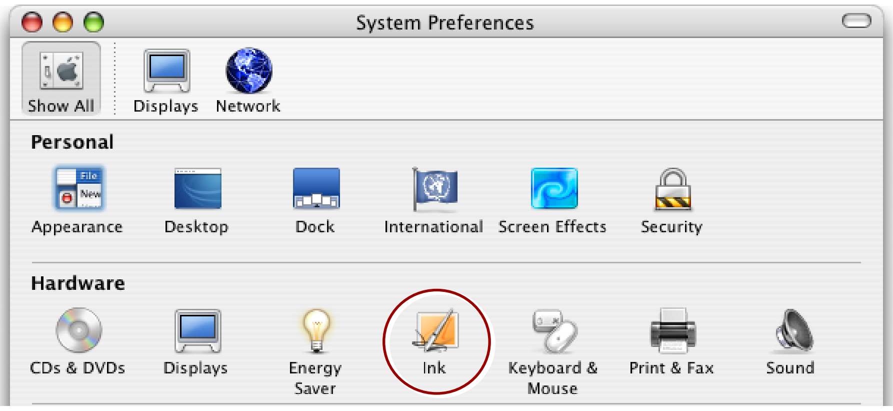
Ink User Interface

Three items make up the Ink user interface: the Ink preferences pane, the Ink window, and Ink writing guides. The Ink preferences pane allows the user to turn handwriting recognition on and off. This preferences pane is only available if the computer detects an Ink input device, such as a tablet. The Ink window provides a writing area as well as controls that allow users to quickly toggle handwriting on or off and to control a number of other aspects of Ink. The Ink window is not available unless handwriting recognition is turned on. The Ink writing guides are visual aids that can help users to enter text horizontally. Each of these user interface items are described in the sections that follow. The user interface described here is available in Mac OS X version 10.3.

Ink Preferences Pane

Users access the Ink preferences pane by opening System Preferences and clicking the Ink icon shown in Figure 1-2. The Ink icon is visible in System Preferences in Mac OS X version 10.2 and later if there is an Ink input device attached to the computer and an appropriate device driver has been installed.

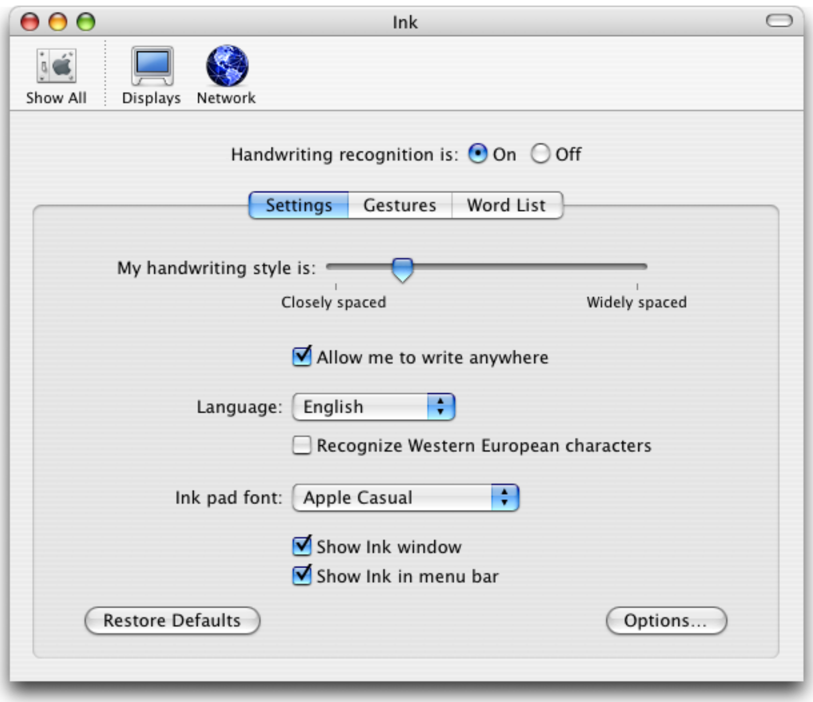
Figure 1-2 The Ink icon in System Preferences



Settings

A user can turn handwriting recognition on or off by clicking the appropriate radio button in the Ink preferences pane, as shown in Figure 1-3. The figure also shows that Ink preferences has three panes within it—Settings, Gestures, and Word List. A user can navigate between the panes by clicking the appropriate tab.

Figure 1-3 Settings in the Ink preferences pane



Handwriting recognition is: [✓ On] Off

My handwriting style is: Closely spaced — Widely spaced

[✓] Allow me to write anywhere

Language: English

[ ] Recognize Western European characters

Ink pad font: Apple Casual

[✓] Show Ink window

[✓] Show Ink in menu bar

The options available in the Settings pane allow a user to:

- Identify handwriting style as closely or widely spaced
- Specify whether to allow Ink input anywhere or to limit Ink input only to Ink-aware applications. In Mac OS version 10.2, the only Ink-aware application is the Ink window.
- Choose a language (English, French, or German) for the recognizer to use
- Specify whether to recognize Western European characters. Users should turn on this option if they plan to enter ligatures or characters that use diacritical marks.
- Choose a font to use in the Ink window
- Specify whether to show the Ink window
- Specify whether to show the Ink icon in the menu bar

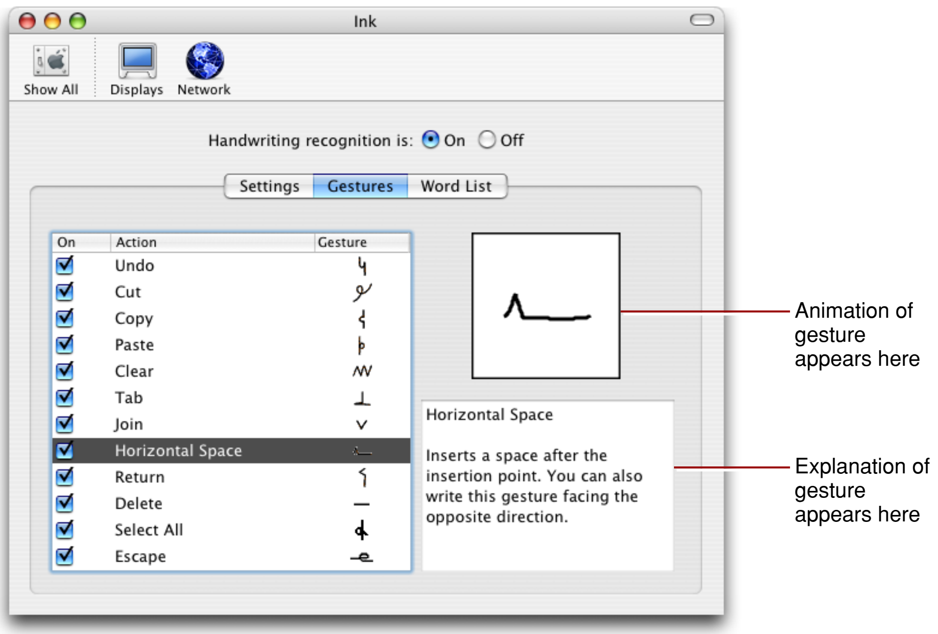
A user who wants to customize Ink further can click the Options button shown in Figure 1-3 to:

- Set how quickly recognition begins after lifting the stylus. A **stylus** is the hand held instrument used to enter data on a graphics tablet. It is also called a **pen**.
- Specify how far the stylus must move before Ink input begins
- Set how long the stylus must be held still before it can be used as a mouse instead of as a pen
- Specify that a phrase is terminated when the stylus moves away from the tablet
- Hide the pointer that normally appears when writing
- Specify to play a sound while writing

Gestures

Gestures are pen strokes that specify editing actions. The list of available editing actions appear when the user clicks the Gestures tabs, shown in Figure 1-4. Recognition for all gestures associated with editing actions is on by default; the user can disable an action by clicking the checkbox next to the action. When a user chooses an action, the gesture associated with the action is drawn in the box to the right of the action list. A user can learn how to draw the more complex gestures (such as Select All) by watching the animated drawing of the gesture in the box. The description of the action appears below the gesture.

Figure 1-4 Gestures in the Ink preferences pane


Ink
Show All
Displays
Network
Settings
Gestures
Word List
Animation of gesture appears here
Explanation of gesture appears here

Handwriting recognition is: [✓ On] Off

On	Action	Gesture
✓	Undo	
✓	Cut	
✓	Copy	
✓	Paste	
✓	Clear	
✓	Tab	⊥
✓	Join	v
✓	Horizontal Space	
✓	Return	
✓	Delete	—
✓	Select All	
✓	Escape	

Horizontal Space

Inserts a space after the insertion point. You can also write this gesture facing the opposite direction.

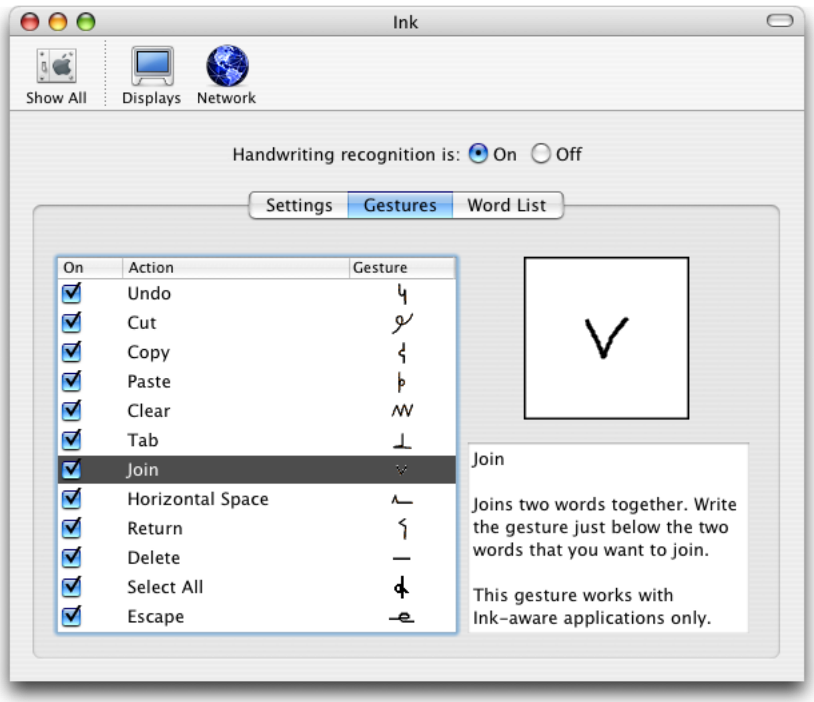
All gestures, except Join (which is described later), can be used in an untargeted manner. **Untargeted gestures** apply to the current selection, if there is one, or to the insertion point, if there is no selection. Most gestures can also be used in a targeted manner. **Targeted gestures** apply either to the item under a predefined hot spot or the entire gesture bounds. Not all gestures that can be targeted have a hot spot. An application can use the hot spot location (if available) and gesture bounds to determine the area to which a targeted gesture should apply.

The Gestures pane doesn't indicate whether a gesture is targeted or not; this information is not available to users and depends, in general, on whether targeted gestures are supported or not in each application. If an application is not specifically set up to handle Ink events (that is, the application is not Ink-aware) the gesture is always treated by the system as an untargeted gesture (except for the Join gesture). An application that is Ink-aware can use the Ink Services API to obtain information about the hot spot. The application can then use that information to apply the editing action specified by the gesture to the appropriate area. See "Supporting Text Editing With Ink Gestures" (page 28) for more information.

A targeted gesture can also be tentative. A **tentative gesture** is Ink that the system treats tentatively as a gesture until your application either confirms the Ink is indeed a gesture or informs the system the Ink is not a gesture. There is only one tentative gesture—the Join gesture, shown in Figure 1-5. Note in the figure that this gesture looks similar to the letter "v." The Join gesture can only be used in an Ink-aware application, because it is the application that must decide whether the Ink is a gesture or text.

When Ink Services receives Ink input that appears similar to that shown in Figure 1-5, it has no way to determine whether the Ink input should be interpreted as the Join gesture or the letter “v.” Ink Services tentatively interprets the Ink as a gesture and passes the gesture to your application. You application must make the determination as to whether the Ink input is the Join gesture or the letter “v.” If you determine the Ink input is the Join gesture, it is up to your application to apply the gesture to the appropriate text. If you determine the Ink input is not the Join gesture, your application informs Ink Services of this by returning `eventNotHandledErr`, which in turn posts the Ink as the letter “v.” See “Supporting Text Editing With Ink Gestures” (page 28) for details on how to handle the Join gesture.

Figure 1-5 The Join gesture is indistinguishable from the letter “v”



Handwriting recognition is: [✓ On] Off

On	Action	Gesture
✓	Undo	
✓	Cut	
✓	Copy	
✓	Paste	
✓	Clear	
✓	Tab	
✓	Join	
✓	Horizontal Space	
✓	Return	
✓	Delete	
✓	Select All	
✓	Escape	

Join

Joins two words together. Write the gesture just below the two words that you want to join.

This gesture works with Ink-aware applications only.

All gestures except the Join gesture fall into the category of non-tentative gestures. Non-tentative gestures are always treated as gestures, regardless of where they are drawn because Ink Services interprets them unambiguously as gestures.

Table 1-1 lists gestures and the categories they can fall into. Notice that some gestures can be targeted or untargeted while others are always untargeted. Join, the only tentative gesture, is always targeted.

Table 1-1 Gesture categories

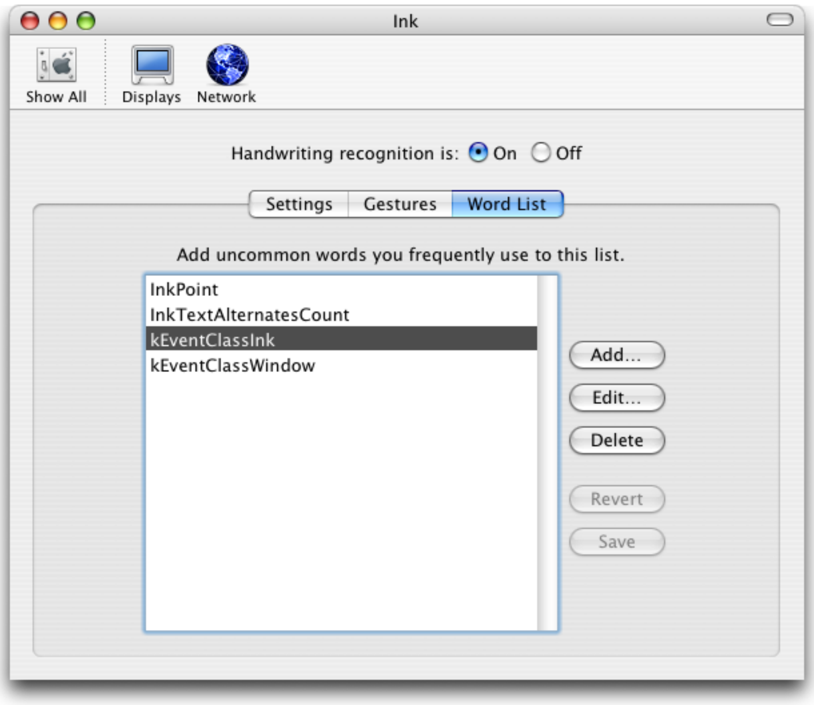
Gesture	Targeted	Untargeted	Tentative
Undo	Never	Always	No
Escape	Never	Always	No
Select All	Never	Always	No
Delete	Never	Always	No

Gesture	Targeted	Untargeted	Tentative
Clear	Sometimes	Sometimes	No
Cut	Sometimes	Sometimes	No
Copy	Sometimes	Sometimes	No
Paste	Sometimes	Sometimes	No
Horizontal Space	Sometimes	Sometimes	No
Tab	Sometimes	Sometimes	No
Return	Sometimes	Sometimes	No
Join	Always	Never	Yes

Word List

Users can improve handwriting recognition, particularly for unusual words such as technical jargon, by entering words into the Word List, shown in Figure 1-6. When the user clicks Add, a sheet appears that has a text field into which the use can type a word. A user can edit or delete words in the list by clicking the appropriate button.

Figure 1-6 A custom word list in the Ink preferences pane



Handwriting recognition is: [✓ On] Off

Add uncommon words you frequently use to this list.

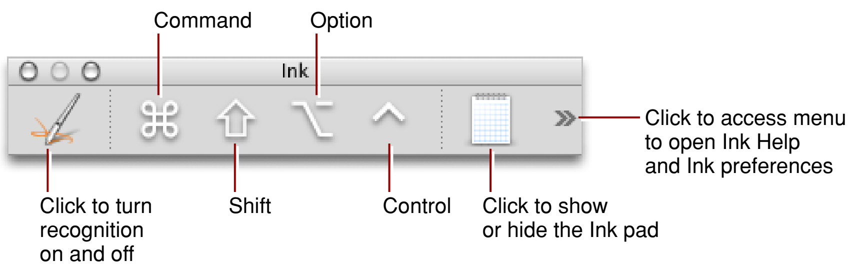
InkPoint
InkTextAlternatesCount
kEventClassInk
kEventClassWindow

Ink Window

The Ink window has two areas—the Ink toolbar and the Ink pad. When a user turns on handwriting recognition in the Ink preferences pane, the Ink toolbar appears, as shown in Figure 1-7. A user can use the Ink toolbar to:

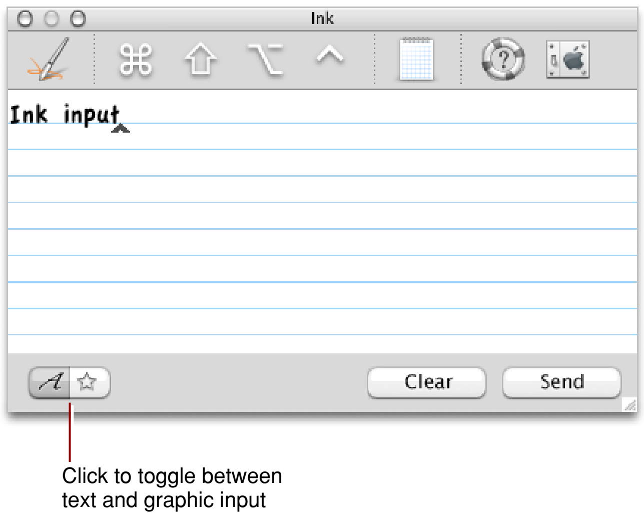
- Turn recognition on or off. This is handy if the user needs to quickly switch between using the stylus to print and using it to control the pointer.
- Show and hide the Ink pad
- Open Ink Help
- Open the Ink preferences pane
- Activate Command, Shift, Option, and Control

Figure 1-7 The Ink toolbar


Command
Option
Ink
Click to access menu to open Ink Help and Ink preferences
Click to turn recognition on and off
Shift
Control
Click to show or hide the Ink pad

When the users clicks the Ink pad icon, the Ink pad appears as shown in Figure 1-8. The Ink pad provides an area for users to print or sketch and then insert the printed text or drawn art into a document. A user can toggle between entering printed text and graphics by clicking the buttons in the lower-left of the Ink pad. The Clear button erases anything currently displayed while Send pastes the entered text or sketch into the insertion point of the currently active Mac OS X application.

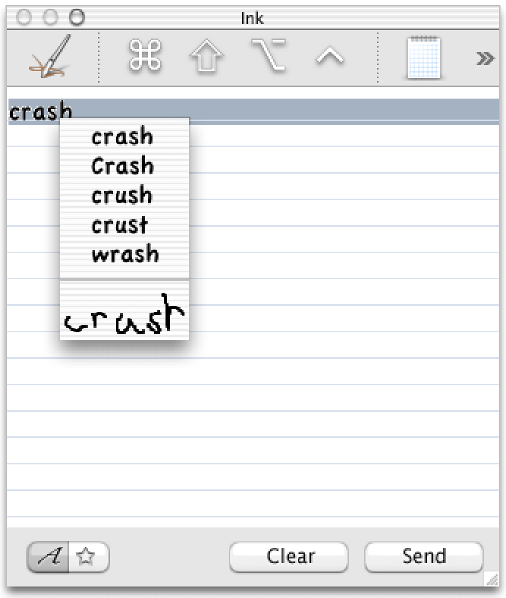
Figure 1-8 The Ink window with the Ink pad open



The Ink window provides a list of alternate interpretations for each unit of text (typically a word) entered by the user. The list is in the form of a menu that the user can access by placing the pointer over a word and then pressing the Control key while clicking the mouse. Figure 1-9 shows a contextual menu with a list of alternates. The original Ink is listed as the last item.

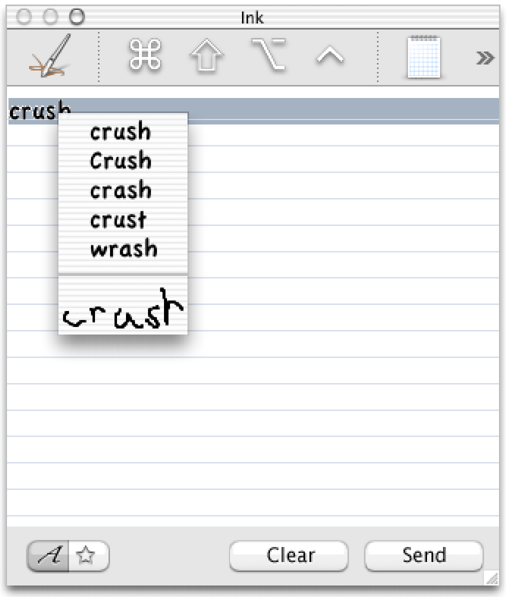
Menu items for a set of alternates whose first letter is an alphabetical character always include an alternate whose first letter is the opposite lettercase. Hence the second item shown in the menu in Figure 1-9 is Crash. Menu items for a set of alternates whose first letter is a nonalphabetical character do not include a lettercase alternate.

Figure 1-9 A menu with a list of alternate interpretations



When the user chooses a word from the list, Ink Services automatically reorders the menu items. The text that was first in the list moves to the second or the third position, depending upon whether the first letter is alphabetical or nonalphabetical. For example, for the list of words shown in Figure 1-9, if the user chooses crush, the menu items are reordered as shown in Figure 1-10.

Figure 1-10 The reordered list of alternate interpretations



Notice that the list of alternates is kept to a maximum length of five. An uppercase alternate for crush is added to the menu while the uppercase alternate Crash is dropped.

For a nonalphabetic first character, however, such as a number, the original moves to the second position. So for the following items:

1239, 1234, 1289, 1284

If the user chooses 1234, the list is reordered as follows:

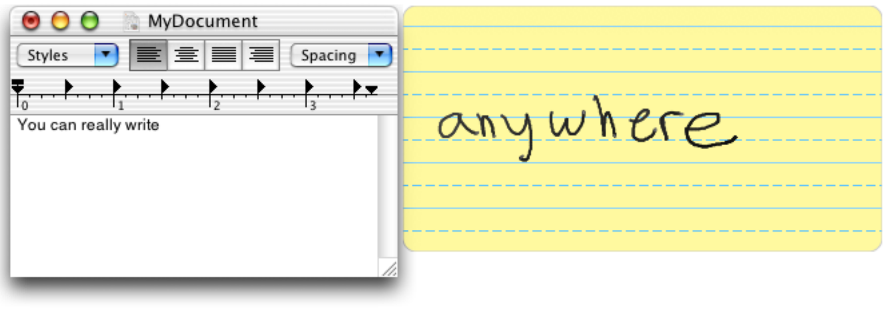
1234, 1239, 1289, 1284

An application may provide support for alternate word lists using the Ink Services API. For more information, see "Implementing a Correction Model" (page 33).

Ink Writing Guides

A user can enter pen input directly into any application that accepts text input. Ink Services automatically draws the user's strokes as Ink, recognizes the text, and sends the recognized text to the application. All the user needs to do is position the stylus and start printing. Ink provides writing guides to facilitate text entry, as shown in Figure 1-11. A user may write almost anywhere on the screen and the recognition results flow to the insertion point in the frontmost application. The only exceptions are specially designated controls and screen areas, such as the Dock, the menu bar, window title bars, and scroll bars.

Figure 1-11 Ink writing guides facilitate pen input into an application



The Ink writing guides looks similar to a translucent piece of yellow, lined, writing paper and is positioned wherever the user writes. The user's printed input appears in the writing guides until Ink Services applies recognition to the Ink text, at which point the Ink phrase is terminated. After the Ink text is recognized, the text appears in the application window as typed text. It is possible for an application to control whether Ink Services draws the writing guides.

How Ink Works in Mac OS X

The flow of Ink from stylus and tablet to an application is shown in Figure 1-12 (page 19). When the user writes a line of text using a stylus, Ink performs a series of operations, as follows:

- Raw data from the hardware is acquired and processed by the tablet driver, which emits pen events using IOKit and Human Interface Device (HID) Manager calls.

  Note: A **pen event** is a shorthand convention for referring to a mouse event that contains tablet data.

- Pen events are passed to the Core Graphics layer. Unlike ordinary mouse events, pen events are usually routed to the frontmost application, even if the event occurred somewhere other than in that application's windows. This allows the user to write anywhere on the screen. The exception to this occurs when pen events are located within certain "instant mousing" areas, such as window title bars, scroll bars, the Dock, and the main Menu Bar.

  An **instant-mousing area** defines an area in which stylus input is interpreted as mouse input; the system "instantly" interprets the stylus as a mouse in these special places and Ink is not generated. For any other location, a user can signal Ink to interpret stylus input as mouse input by pressing and holding the stylus still briefly. It not possible for a user to start Ink input with the stylus placed over an instant-mousing area.

- Pen events (mouse plus tablet data) are then passed to Ink Services, where the various components (see Figure 1-1 (page 9)) perform the tasks listed in the Ink Input Method box shown in Figure 1-12 (page 19).

- The input method component of Ink Services accumulates pen events into strokes and phrases.

  Strokes are collections of points (onscreen locations), spanning the time from which the pen is pressed to the tablet hard enough to generate a mouse-down event until the time at which the pen is lifted from the tablet enough to generate a mouse-up event, and corresponds to the common concept of strokes one might write with paper and pencil.

Phrases are collections of strokes, spanning the time from when the first stroke is started to the time at which the phrase is terminated due to one of several common events—a timeout following a pen lift (but staying in proximity), a lift that takes the stylus out of tablet proximity (the limited range over which the tablet can sense the stylus), a recognizer-generated break (due to the user introducing an extra large horizontal space or beginning a new line), or a direct request by the application to terminate the phrase. An Ink phrase can represent a letter, word, or longer unit.

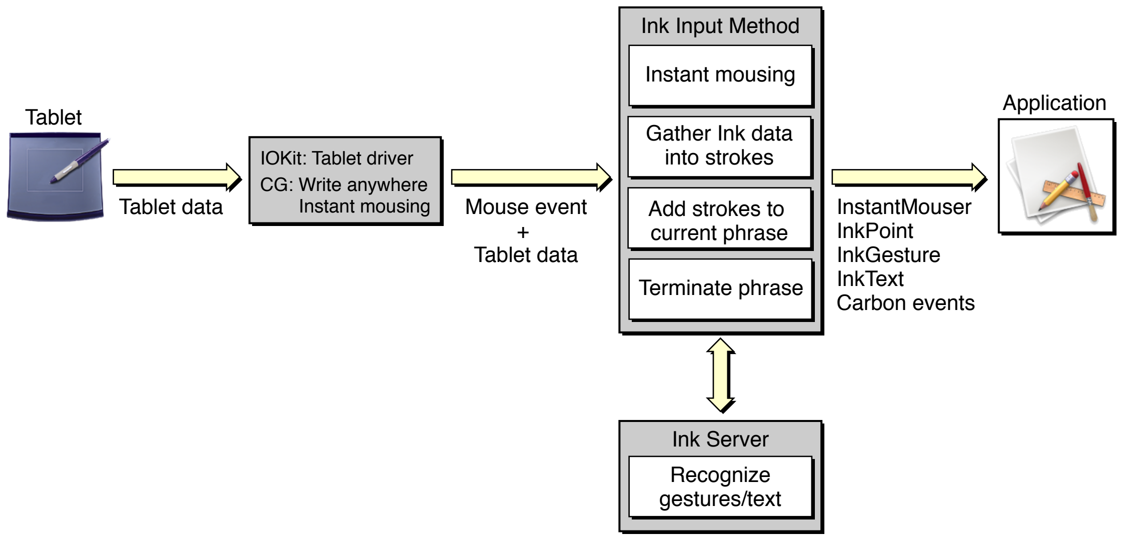
- The Ink server hosts the recognizer which processes accumulated phrases. Ink Services is set up by default to recognize both text and editing gestures. See "The Ink Recognizer" (page 20) for more information on how the recognizer works.

  If a user toggles the write-anywhere mode off, then pen events are ignored by Ink Services, everywhere except in ink-aware applications. These points are not drawn by Ink Services nor interpreted by the recognizer unless the application calls the Ink Services function `InkSetApplicationWritingMode` with the `iWhere` parameter set to `kInkWriteAnywhereInApp`. As long as Ink is turned on, Ink will always be drawn and recognized if the user writes directly into the Ink pad in the Ink window. Similarly, as long as Ink is turned on, Ink drawing and recognition services will always be provided for your ink-aware application once you enable them by calling `InkSetApplicationWritingMode`, regardless of the user's write-anywhere preference setting.

- When the Ink input is recognized, Ink Services generates one or more Carbon events of class `kEventClassInk` and the appropriate event kind `kEventInkText` or `kEventInkGesture`. Your application can obtain the data it requires to accomplish its goals by installing appropriate handlers for Ink-related Carbon events. When your application is the active application (that is, the one that has keyboard focus) it can receive the Carbon events, extract the relevant event parameters, and process the data accordingly. If your application does not handle the events, then Ink Services handles them.

  For more details, see "Ink-Related Carbon Events" (page 21).

Figure 1-12 The flow of Ink from stylus to application



The Ink Recognizer

The Ink recognizer is at the heart of Ink Services. It is the algorithmic component of Ink Services that identifies written text and gestures. Built using neural-network technology, the architecture of the recognizer integrates multiple representations of the input data. This design, combined with the training regimen used to build the recognizer, provides robust, accurate character recognition despite individual differences in the writing styles of users.

Three outcomes are possible from the Ink recognizer. The first two outcomes are the ones you are likely to see; the third is a rare event.

- The Ink input is recognized as a gesture.

  When the recognizer determines with a high confidence level that the Ink input is a gesture, Ink Services generates a gesture event.

  Recall that Ink input in the form of the standalone letter "v" is tentatively treated as a gesture. If your application determines the Ink input is not the Join gesture, not handling the gesture (returning `eventNotHandledErr`) notifies Ink Services that the Ink input should be interpreted as text by the recognizer. See "Gestures" (page 12) for more information.

- The Ink input is recognized as text.

  The recognizer ranks text interpretations according to a confidence level. Up to five interpretations are returned to an application through the Ink text event (in the `InkTextRef` parameter). If the Ink text event is not handled by the application, then for compatibility with non-ink-aware applications, only the top-choice, highest confidence interpretation is returned in a `kEventTextInputUnicodeForKeyEvent` event. If that event is not handled, then the text is returned to the application receiving the input as raw `keyDown` events.

  Using the Ink Services API, your application can obtain a list of interpretations, in ranked order, from the recognizer, and then use the list to implement a correction model. For more information, see "Implementing a Correction Model" (page 33).

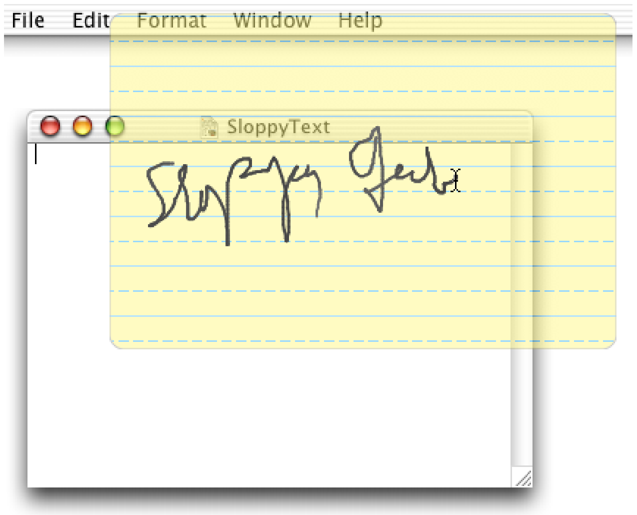
- The Ink input is not recognized.

  In the rare case that handwritten input cannot be recognized, the recognition system returns a diamond character that indicates the text is not recognized.

  However, in the event of misrecognition it is more often the case that the Ink input is recognized as text, but that the text has no meaning to the user. For example, the Ink input shown in Figure 1-13 would be recognized as "54M^ NG" or some other meaningless text. With a little practice most users improve their printed input to achieve a high recognition rate.

  Note that the Ink input shown in the Figure 1-13 is script, not print. The recognizer is optimized for printed text.

Figure 1-13 Ink input that is difficult to recognize



Ink-Related Carbon Events

Ink Services notifies your application of Ink-related events by generating Carbon events. The events listed in Table 1-2 (page 21) are generated by the system, in the order listed, until an event in the chain is handled by an event handler provided by your application. Once your application handles one of these events (by returning any result other than the result code `eventNotHandledErr`), the chain is terminated and no further events are generated by Ink Services for that data.

Table 1-2 Carbon events generated by Ink input

Event category	Carbon event	The event is received by your application …
pen	`kEventMouseDownkEventMouseDragged`	along with tablet coordinates and a pressure value, only when recognition is disabled.
instant mousing	`kEventAppIsEventInInstantMouser`	only at the start of a phrase.
Ink point	`kEventInkPoint`	during a phrase.
gesture	`kEventInkGesture`	only when a gesture is recognized.
text	`kEventInkText`	when text is recognized.
Unicode text	`kEventTextInputUnicodeForKeyEvent`	when Unicode text is recognized and when `kEventInkText` is not handled.
key	`kEventRawKeyDown`	when text is recognized and the previous events are not handled.

The Ink-specific events most important to any application that uses Ink Services—instant mousing, Ink point, Ink gesture, Ink text —are described in the sections that follow. For more information, see "Obtaining Parameters from Ink Text and Gesture Events" (page 25).

Instant Mousing Events

The event `kEventAppIsEventInInstantMouser` is dispatched only when the stylus is initially pressed to a tablet and before Ink Services determines whether the user is writing or not. Instant mousing areas are those areas where you do not want the tablet to start Inking. Once the user begins to write, Ink input does not generate instant-mousing events until the current phrase is terminated.

Instant mousing in standard Carbon and Cocoa controls, drag regions, and so forth is handled for you automatically. If your application implements its own custom controls that you want to be treated as instant mousing areas, it must provide a handler for this event. The handler must check the location of the pen event and return `noErr` if the pen's location is in a custom instant mousing region, which will force the pen to behave as a mouse. Your handler should return `eventNotHandledErr` outside the custom mousing regions, to allow the user to commence writing in most locations.

Note: Cocoa applications can implement the `NSResponder` method `shouldBeTreatedAsInkEvent` for these purposes. This method returns `YES` if `theEvent` should be treated as an Ink event, `NO` if `theEvent` should be treated as a mouse event. This provides the ability to distinguish when a pen-down should start an Ink input event versus when a pen-down should be treated as a mouse down event. This allows for both a write-anywhere model for pen-based input and arbitrary application-defined instant-mousing regions. Note the inverse semantics from the Carbon event.

Ink Point Events

Ink Services sends an Ink point event (`kEventInkPoint`) whenever it detects a pen event at the start of or within a phrase (that is, whenever the user is actually entering Ink and not mousing). If the Ink point event is not handled by your application, Ink Services continues to handle the event through the normal recognition path. If the Ink point event is handled by your application, Ink Services drops the current mouse event from further recognition handling. This allows your application to treat certain areas of a window as special, non-inking areas, and provides the option for your application to terminate the Ink input session. It also allows your application to draw its own Ink, while letting Ink Services manage the inking-versus-mousing decision, and even carry out normal recognition services (if your event handler returns `eventNotHandledErr`).

If your application chooses to handle Ink point events, be aware that you can receive mouse events that lie outside your application's windows. For example, if your application draws its own Ink, it could continue to track the Ink points past the visible bounds of the window, making the out-of-bounds Ink visible if the user scrolls to look at it. Or your application could provide a separate Ink-background window for Ink input for the out-of-bounds Ink, similar to the Ink writing guides provided by Ink Services.

Ink Gesture Events

Ink Services dispatches an Ink gesture event (`kEventInkGesture`) only if the Ink is recognized as a gesture with a high degree of confidence. All gestures except the Join gesture can be interpreted unambiguously as a gesture. So for most gestures, whether your application handles a gesture event or not, the event chain is terminated at that point—either your application handles the gesture or it returns `eventNotHandledErr` and Ink Services handles the gesture. However, if the Join gesture (which must be targeted) is not handled by your application, Ink Services performs text recognition on the Ink and posts it as the letter "v."

If an application does not handle the gesture, then Ink Services posts the command event (`HICommand`) associated with the editing action specified by the gesture. If a command event isn't defined for the gesture, as in the case of the Escape, Delete, Tab, Horizontal Space, and Return, then Ink Services posts the keyboard equivalent for the gesture. For these gestures, this is simply the key event associated with the gesture (Escape, Delete, Tab, Space, and Return key presses).

Recall that the Join gesture, which is a tentative gesture, must be handled by the application, otherwise Ink Services treats it as the letter "v." (There is no command event or keyboard equivalent fallback.) As such, the Join gesture is only available in an Ink-aware application. See "Gestures" (page 12) for more information.

Ink Text Events

The Ink text event (`kEventInkText`) is sent when Ink Services recognizes Ink input as text. In Roman languages the Ink text event typically corresponds to an individual word. It contains the original raw Ink and the recognized text and typically includes a list of alternate interpretations that your application can show should you want to provide an easy-to-use correction model. (See "Implementing a Correction Model" (page 33) for details.)

The parameters associated with an Ink text event are `kEventParamInkTextRef` and `kEventParamInkTextKeyboardShortcut`. The parameter `kEventParamInkTextRef` is a reference to an opaque Ink text object (`InkTextRef`). An **Ink text object** contains data that describes the recognized text. You can't access the object directly, but you can use a variety of Ink Services functions to obtain data from the object.

The parameter `kEventParamInkTextKeyboardShortcut` is a Boolean value that indicates whether the Ink text is likely a keyboard equivalent. The value is `true` if the Command or Control key is pressed and the top-choice alternate text is a single character. Checking for this parameter provides an easy way for you to determine if the text associated with an `InkTextRef` is likely to be a keyboard equivalent instead of text. Otherwise, to determine whether the Ink text is a keyboard equivalent, you would need to extract the `kEventParamInkTextRef` parameter, retrieve the `CFStringRef` for the text, determine the length of the string, and then check for modifier keys. In most cases, you don't need to handle an Ink text event that is a keyboard equivalent, and can immediately return `eventNotHandledErr`.

If the Ink text event is not a keyboard equivalent and you do not handle the event, Ink Services posts a `kEventTextInputUnicodeForKeyEvent` event. If that goes unhandled, Ink Services posts a sequence of raw `keyDown` events corresponding to the top-choice recognition result.

Mouse Event Coalescing

Mouse events, whether generated by a mouse and mouse driver or by a graphics tablet and tablet driver, have the potential of being posted at a faster rate than the system can handle. For example, if an application redraws a window for each `mouseDragged` event generated when a user drags a window, the window could move slowly or lag behind the pointer. To avoid slowing down the application, the Carbon Event Manager coalesces mouse events instead of placing all the mouse events in a queue. **Mouse event coalescing** is a process that merges `mouseMoved` and `mouseDragged` events by checking to see if one of these events exists in the event queue, and if it does, replacing the previously queued event with the more recently-generated event. Note that `mouseUp` and `mouseDown` events are never coalesced, as they are semantically meaningful.

Since Ink is accumulated predominantly through `mouseDragged` events (that have tablet data associated with them), event coalescing can reduce the fidelity of the data used to draw Ink input and to perform handwriting recognition. Reduced fidelity can produce Ink that appears faceted instead of smooth, and can reduce recognition accuracy.

To avoid fidelity problems, Ink Services temporarily disables event coalescing when a stylus enters proximity of the graphics tablet. Thus Ink data is guaranteed to be smooth, for both rendering and recognition purposes. When the stylus leaves the proximity of the tablet, or when the stylus is pressed down in an instant-mousing region, event coalescing is enabled again. Thus coalescing is active, as usual, during window drags and normal mouse activity.

If your application calls the function `InkSetApplicationWritingMode` to disable Ink Services management of pen events so that your application can accumulate Ink data on its own, your application may need to manage event coalescing. Otherwise, Ink rendering and recognition may suffer from fidelity problems. You can use the Carbon Event Manager function `SetMouseCoalescingEnabled` to manage event coalescing in your application. Make sure that you disable event coalescing only when absolutely necessary. See *Carbon Event Manager Reference* for more information on event coalescing.

Ink Services Tasks

This section shows how you can use the Ink Services API to accomplish the following tasks:

- "Obtaining Parameters from Ink Text and Gesture Events" (page 25), shows how your application can obtain the data associated with Ink input.
- "Handling Phrase Termination" (page 26), discusses how to override automatic phrase termination and request phrase termination at the appropriate time for your application.
- "Supporting Text Editing With Ink Gestures" (page 28), discusses the gestures you must handle and those you can let Ink Services handle for you.
- "Implementing a Correction Model" (page 33), shows how your application can access a list of alternate interpretations for Ink input and present those alternates in a contextual menu to your users.
- "Implementing Deferred Recognition" (page 35), describes how to set the recognition state, collect tablet data, accumulate an Ink phrase, and request recognition at the appropriate time.

Obtaining Parameters from Ink Text and Gesture Events

If you want to perform any of the other tasks described in this chapter, you will need to obtain one or more event parameters from Ink-related Carbon events. This section shows you how to obtain the parameters associated with Ink text and gesture events. Before you read this section, you should be familiar with the events and event parameters discussed in "Ink-Related Carbon Events" (page 21).

Ink Services generates Carbon events of class `kEventClassInk`. When Ink Services recognizes a phrase as text, it generates the event `kEventInkText`. You can extract the associated parameters—`kEventParamInkTextRef` and `kEventParamInkTextKeyboardShortcut`—by calling the Carbon Event Manager function `GetEventParameter`, as shown in Listing 2-1. Checking for the `kEventParamInkTextKeyboardShortcut` parameter provides an easy way for you to determine if the `InkTextRef` is likely to be a keyboard equivalent for a command instead of text. If this parameter is `false`, the you can use the function `InkTextCreateCFString` to obtain the recognized text associated with the Ink text object (`InkTextRef`). When you pass `0` to this function, you obtain the most likely interpretation.

Listing 2-1 Extracting parameters for the Ink text event

```
OSStatus status = noErr;
InkTextRef myInkTextRef;
Boolean  myKeyboardShorcut;

status = GetEventParameter (myEvent,
                kEventParamInkTextRef,
                typePtr,
                NULL,
                sizeof (Ptr),
                NULL,
                &myInkTextRef);
```

```
status = GetEventParameter (myEvent,
                kEventParamInkTextKeyboardShortcut,
                typeBoolean,
                NULL,
                sizeof (Boolean),
                NULL,
                &myKeyboardShortcut);

if (myKeyboardShortcut == false)
        return (eventNotHandledErr);
InkTextCreateCFString (myInkTextRef, 0);
// Your code to insert the text into the application document.
```

When Ink Services recognizes an Ink phrase as a gesture, Ink Services generates the event `kEventInkGesture`. You can extract the associated parameters—`kEventParamInkGestureKind`, `kEventParamInkGestureBounds`, and `kEventParamInkGestureHotspot`—using the code shown in Listing 2-2.

Listing 2-2 Extracting parameters for the Ink gesture event

```
OSStatus status = noErr;
UInt32 myGestureKind;
HIRect myGestureBounds;
HIPoint myGestureHotspot;

status = GetEventParameter (myEvent,
                kEventParamInkGestureKind,
                typeUInt32,
                NULL,
                sizeof (UInt32),
                NULL,
                &myGestureKind);

status = GetEventParameter (myEvent,
                kEventParamInkGestureBounds,
                typeHIRect,
                NULL,
                sizeof (HIRect),
                NULL,
                &myGestureBounds);

status = GetEventParameter (myEvent,
                kEventParamInkGestureHotspot,
                typeHIPoint,
                NULL,
                sizeof (HIPoint),
                NULL,
                &myGestureHotspot);
```

Handling Phrase Termination

The default behavior is for Ink Services to terminate a phrase when one of the following events occur:

- The user removes the stylus from the proximity of the tablet
- A specified period of time elapses in which the stylus is not pressed to the tablet (The user can control the period of time in the Ink preferences pane.)
- The user writes sufficiently far away from the previous Ink—either horizontally, or on a new line

You can use the function `InkSetPhraseTerminationMode` if your application does not want the default behavior or wants complete control over when Ink phrases are terminated. If you turn off automatic phrase termination, you must make sure you manage phrase termination appropriately for your application.

If you want to control phrase termination in your application, you must perform the following tasks:

- Call the function `InkSetPhraseTerminationMode` to turn off automatic phrase termination. You can do so using the following line of code:

  ```
  InkSetPhraseTerminationMode (kInkSourceUser, kInkTerminationNone);
  ```

  The first parameter to this function specifies the source of the Ink data stream. This example shows how to control termination of an Ink phrase that originates from direct user input (`kInkSourceUser`) rather than from the application (`kInkSourceApplication`).

- Write code that monitors Ink input and checks for the termination conditions you define.

  The termination conditions you define determine how you should best carry out this step. For example, if you provide users with a "terminate phrase" button in the user interface, then your application should check for the command issued by the button press.

  In such a case or if your phrase termination criteria depends on such data as where the user is writing, the proximity of the pen to the tablet, whether a modifier key is pressed, or the amount of pressure applied to the pen, you can install a Carbon event handler for the event `kEventInkPoint`, then monitor the location and other relevant parameters returned in this event.

  You may find it useful to call the function `InkIsPhraseInProgress`. The function returns `true` when there is an Ink phrase that can be terminated.

- When your application determines your termination conditions have been met, call the function `InkTerminateCurrentPhrase`.

Listing 2-3 shows an example of a handler for a hypothetical application that provides users with a "terminate phrase" button. The application must first install the handler (which, in this case, handles the event `kEventInkPoint`) and call the function `InkSetPhraseTermination` with the parameter `kInkTerminationNone`. A detailed explanation for each numbered line of code appears following the listing.

Listing 2-3 Code that handles phrase termination

```
{

    GetEventParameter (myInkPointEventRef, kEventParamEventRef,
                     typeEventRef, NULL, sizeof (EventRef), NULL,
                     &myMouseEventRef);                                    // 1

    if (GetEventKind (myMouseEventRef) == kEventMouseDown)                 // 2
    {
        if (MyTestForButtonHit (myMouseEventRef) == true)                  // 3
        {
            if (InkIsPhraseInProgress() == true)                           // 4
```

```
                {
                    InkTerminateCurrentPhrase(kInkSourceUser);                    // 5
                    return (noErr);
                }
            }
        }
        return (eventNotHandledErr);                                            // 6

}
```

Here's what the code does:

1. Extracts the mouse event reference from the Ink point event.
2. Checks to see if the event is a mouse down event.
3. Checks to see if the mouse down event is in the termination button provided by the application. The `MyTestForButtonHit` function is an application-defined function that determines if the mouse event is within the bounds of the termination button.
4. Checks to see if Ink Services has an Ink phrase in progress; otherwise there is nothing to terminate. Note that the function `InkIsPhraseInProgress` should be called only to check whether a phrase is in progress for an Ink data stream that originate from the user, and not for one that originates from your application. See *Ink Services Reference* for more information.
5. Terminates the phrase and returns `noErr` to indicate the event has been handled. You must pass the constant `kInkSourceUser` to specify that the function `InkTerminateCurrentPhrase` should be applied to the Ink data stream that originates from direct user input.
6. Returns `eventNotHandledErr` if any of the previous `if` statements are false. If there is an Ink phrase in progress, it is not terminated.

Supporting Text Editing With Ink Gestures

Before you begin to write any code that supports text editing using Ink gestures, you should be thoroughly familiar with the terms "targeted gesture," "untargeted gesture," and "tentative gesture," because you handle each of these gestures differently. These terms are described in "Gestures" (page 12). You should also be familiar with the Ink-gesture event and event parameters described in "Ink Gesture Events" (page 22), as gesture information is available to your application through Carbon events.

To support text editing with Ink gestures, your application must write and install an event handler for the Carbon event kind `kEventInkGesture`. The parameters associated with an Ink gesture event are:

- `kEventParamInkGestureKind`—The gesture kind can be any of the constants listed in Table 2-1 (page 29). These constants are defined as the `InkGestureKind` enumeration in the Ink Services API. The Horizontal Space and Return gestures are each represented by two constants because the user can write these gestures facing the opposite direction of what's shown in the Ink pane of System Preferences. The left and right distinction for the Horizontal Space gesture indicates the side on which the long, horizontal tail is drawn. The left and right distinction for the Return gesture indicates the direction the small angle-bracket points.

- `kEventParamInkGestureBounds`—The gesture bounds is the rectangle to which some gestures should apply. You can use this information when your application handles targeted gestures.
- `kEventParamInkGestureHotspot`—The gesture hot spot is the area to which some gestures should apply. Only targeted gestures can have a hot spot. You can use this information when your application handles targeted gestures.

Table 2-1 Gesture constants

Gesture	Gesture kind constant
Undo	`kInkGestureUndo`
Clear	`kInkGestureClear`
Select All	`kInkGestureSelectAll`
Escape	`kInkGestureEscape`
Cut	`kInkGestureCut`
Copy	`kInkGestureCopy`
Paste	`kInkGesturePaste`
Horizontal Space	`kInkGestureLeftSpace`
Horizontal Space	`kInkGestureRightSpace`
Tab	`kInkGestureTab`
Return	`kInkGestureLeftReturn`
Return	`kInkGestureRightReturn`
Delete	`kInkGestureDelete`
Join	`kInkGestureJoin`

After your event handler obtains the gesture kind from the event parameter `kEventParamInkGestureKind` (see Listing 2-2 (page 26)), it should handle gestures as follows:

- For gestures that are always untargeted (Undo, Select All, Escape, and Delete), return `eventNotHandledErr`. Because these gestures are always untargeted, you can let Ink Services handle them for you. Ink Services converts the unhandled gesture to the command event (`HICommand`) associated with the editing action specified by the gesture. If a command event isn't defined for the gesture, as in the case of the Escape and Delete, or if the command goes unhandled, then Ink Services posts the keyboard equivalent for the gesture. Untargeted gestures apply to the current selection or insertion point.

  See "Handling Untargeted Gestures" (page 32) for more details.

- For the Join gesture (which is a tentative gesture), check to see if the top-left and top-right points of the bounding box defined by the gesture are sufficiently close to the end and beginning of two words (or other editable objects), with a space between the words. If so, then your application should elide the space between the words and return `noErr`. See "Handling Targeted Gestures" (page 30) for more details.

  If not, then return `eventNotHandledErr` to signal to Ink Services to process the raw Ink as text and post the recognition results, which should be the letter "v."

  You must make sure that you return `eventNotHandledErr` when you determine the Ink is not a Join gesture, otherwise the user will never be able to enter the letter "v" as a standalone character.

- For all other gestures, check to see if the gesture's hot spot (or bounding box, as appropriate) is over a suitable target, such as a word, image, or other object. If so, then apply the gesture to that target. Note that any non-empty selection range is a suitable target, and should be treated as a single editable object for applying targeted gestures. (So if a gesture hot spot is anywhere in the selection, the gesture should be applied to the entire selection.) See "Handling Targeted Gestures" (page 30) for more details.

  If the gesture is not over a suitable target, then it should be applied to the current selection, if it is non-empty, else it should be applied to the insertion point, like an untargeted gesture.

  If the gesture is being applied to the selection or the insertion point, whether due to a hot spot in the selection or one that is not over any suitable target, your application may choose to return `eventNotHandledErr` to let Ink Services apply the editing action automatically.

Handling Targeted Gestures

You can use gesture and text relationships to determine the extent of the text modified by the gesture. Most targeted gestures have a defined hot spot that your application can use to determine the area to apply the editing action.

Table 2-2 lists targeted gestures, whether the gesture has a hot spot, and the editing actions you should perform when you handle the gesture.

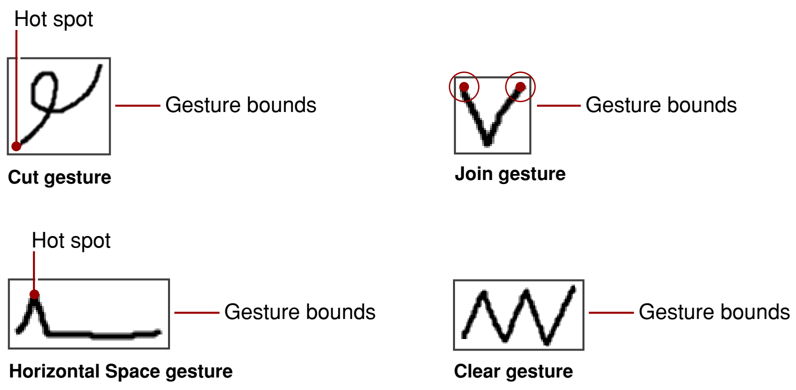
Table 2-2 Targeted gestures, hot spots, and editing actions

Constant	Has a hot spot	Perform the following action ...
`kInkGestureClear`	No, use the gesture bounds	delete the text if the gesture bounds overlaps by 50% or more.
`kInkGestureCut`	Yes, the starting point of the gesture	cut the text (a single word in Roman languages) the hot spot overlaps.
`kInkGestureCopy`	Yes, the starting point of the gesture	copy the text (a single word in Roman languages) the hot spot overlaps.
`kInkGesturePaste`	Yes, the starting point of the gesture	paste the Clipboard contents into the location specified by the hot spot. Paste over a word if the hot spot is on a word.

Constant	Has a hot spot	Perform the following action ...
`kInkGestureLeftSpacekInkGesture-RightSpace`	Yes, the topmost point of the gesture	insert a single space character into the location specified by the hot spot.
`kInkGestureTab`	Yes, the starting point of the gesture	insert a single tab character into the location specified by the hot spot.
`kInkGestureLeftReturn`	Yes, the leftmost point of the gesture	insert a return (new line) character into the location specified by the hot spot.
`kInkGestureRightReturn`	Yes, the rightmost point of the gesture	insert a return (new line) character into the location specified by the hot spot.
`kInkGestureJoin`	No, you must extract two points from the gesture bounds	delete the space between the words specified by the top-left and top-right points of gesture bounds.

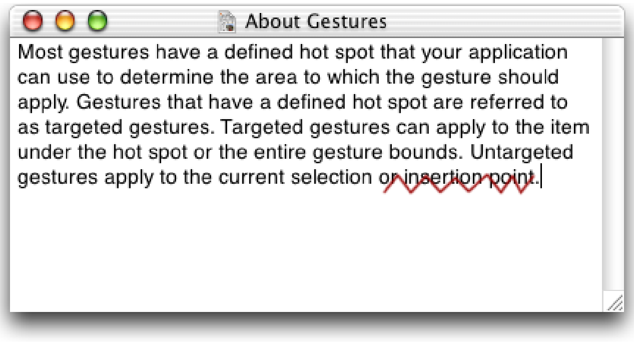
Figure 2-1 shows gesture bounds for four editing gestures—Cut, Horizontal Space, Clear, and Join. The Cut and Horizontal Space gestures have a hot spot but the Clear gesture does not. Rather, the bounds of the Clear gesture define its targeting area. The Join gesture doesn't have a hot spot per se; instead it has two points that specify the two words that should be joined. These two points are contained in the gesture bounds, and your application must extract the points from the bounds.

Figure 2-1 Gesture bounds and hot spots



To handle the Clear gesture, which does not have a hot spot, your application should apply the editing action to all of the text the gesture overlaps (to a sufficient degree—say, more than 50% of the onscreen area of each word—on a word-by-word basis). For example, the Clear gesture in Figure 2-2 is written over the words "or insertion point" so your application should delete those words. To make that determination you would first obtain the gesture bounds from the event `kInkGestureEvent`, then determine the words the gesture is written over, and delete those words.

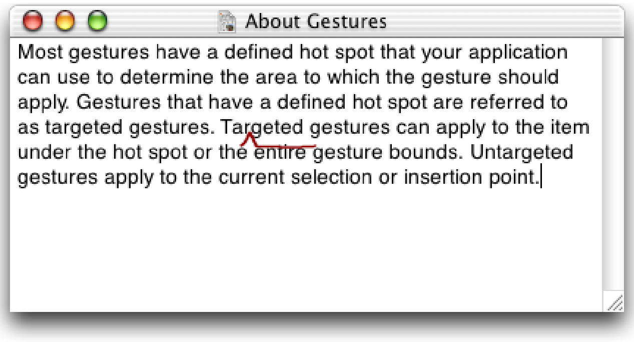
Figure 2-2 The Clear gesture



Most gestures have a defined hot spot that your application can use to determine the area to which the gesture should apply. Gestures that have a defined hot spot are referred to as targeted gestures. Targeted gestures can apply to the item under the hot spot or the entire gesture bounds. Untargeted gestures apply to the current selection or insertion point.

For targeted gestures that have a hot spot, only the hot spot is relevant to the editing action your application takes; the gesture bounds aren't. For example, to handle the Horizontal Space gesture your application must obtain the hot spot from the event `kInkGestureEvent`.The Horizontal Space gesture in Figure 2-3 is positioned between the letters "r" and "g." To process this gesture, your application would determine the characters on either side of the hot spot, then insert a space in that position.

Figure 2-3 The Horizontal Space gesture



Most gestures have a defined hot spot that your application can use to determine the area to which the gesture should apply. Gestures that have a defined hot spot are referred to as targeted gestures. Targeted gestures can apply to the item under the hot spot or the entire gesture bounds. Untargeted gestures apply to the current selection or insertion point.

If the gesture hot spot is drawn outside your application's windows or in a location where the gesture would not apply, the gesture should be treated as an untargeted gesture. That is, you should either return `kEventNotHandledErr` and let Ink Services handle the gesture for you, or you should apply the editing action to the current text selection or insertion point (if there is no selection).

Handling Untargeted Gestures

Your application does not need to handle untargeted gestures; it can simply return `kEventNotHandledErr` and let Ink Services handle the gesture for you. If for some reason you decide to handle untargeted gestures, you should perform the actions listed in Table 2-3. Note that the editing action for an untargeted gesture can depend on whether or not there is a selection.

Table 2-3 Actions specified by untargeted Ink gestures

Constant	If there is a selection, specifies to ...	If there is no selection, specifies to ...
`kInkGestureUndo`	undo the last action.	undo the last action.
`kInkGestureClear`	clear the current selection.	do nothing.
`kInkGestureSelectAll`	select all items in the window that has user focus.	select all items in the window that has user focus.
`kInkGestureEscape`	act as if the Escape key is pressed.	act as if the Escape key is pressed.
`kInkGestureCut`	cut the current selection.	do nothing.
`kInkGestureCopy`	copy the current selection.	do nothing.
`kInkGesturePaste`	paste the Clipboard contents over the current selection.	paste the Clipboard contents into the insertion point.
`kInkGestureLeftSpacekInkGesture-RightSpace`	replace the current selection with a single space character.	insert a single space character at the insertion point.
`kInkGestureTab`	replace the current selection with a single tab character.	insert a single tab character.
`kInkGesture-LeftReturnkInkGestureRightReturn`	replace the current selection with a return (new line) character.	insert a return (new line) character.
`kInkGestureDelete`	delete the current selection.	delete the item immediately preceding the insertion point.

Implementing a Correction Model

Ink Services communicates events through the Carbon Event Manager. Once Ink Services interprets an Ink phrase, it sends your application the Carbon event kind `kEventInkText` whose associated parameter is a reference to an opaque Ink text object (`InkTextRef`). The Ink text object contains the original Ink entered by the user and a list of interpretations for the Ink in ranked order. Ink Services creates a list of up to five possible interpretations. The most likely interpretation is the first item in the list, with less likely interpretations appearing in rank order after this item. (See "Ink Window" (page 15) for details on how Ink orders the interpretations.)

Your application can implement an easy-to-use correction model by performing the following tasks:

- Create a contextual menu.

  Although the menu doesn't have to be a contextual one, alternate interpretations are typically presented to the user in this way. See "Ink Window" (page 15) for an example of what a contextual menu looks like when used for Ink.

You can create a menu any way you like. For example, you can:

- ❏ Create the menu in Interface Builder and then unarchive the menu using the Interface Builder Services API. For more information, see *Unarchiving Interface Objects With Interface Builder Services*.
- ❏ Use the Menu Manager API. For more information, see *Menu Manger Reference*.

You can obtain either document from the developer documentation website, accessed through:

http://developer.apple.com/

- Insert the list of interpretations into the menu.

  You can call the Ink Services function `InkTextInsertAlternatesInMenu`, supplying the `InkTextRef` and a valid menu reference as parameters, as shown in the following code:

```
alternateItemsCount = InkTextInsertAlternatesInMenu (
                myInkRef, // obtained through your Carbon event handler
                myAlternatesMenu, // a valid menu reference
                0); // location in menu to insert list
```

- Show the contextual menu to the user and obtain the user's selection, if any.

  Your application should display the menu by calling the Menu Manager function `ContextualMenuSelect` (or other appropriate function, such as `PopUpMenuSelect`).

  You can get the user's selection by checking the return value of `outUserSelectionType`, as shown in the following code:

```
status = ContextualMenuSelect (myAlternatesMenu,
                myMouseLocation, // the location to display the menu
                false, // reserved for future use
                kCMHelpItemRemoveHelp, // don't provide a help item
                NULL. // a help string, which is not relevant here
                NULL, // no need for system to examine the selection
                &selectionType, //on output, indicates selection type
                &menuID, // on output, the menu ID of chosen item
                &menuItemIndex); // on output, menu item of chosen item
```

- If the user chooses an item, you should obtain the string associated with the menu item and then call your function to replace the word in the document with the string returned by this function.

  You can obtain the string associated with the user's choice using one of the following two methods:

  - ❏ Call the Ink Services function `InkTextCreateCFString`, supplying `0` as the `iIndex` parameters.

    This will always return the current top-choice interpretation, which will have been automatically altered by the user's selection. Note, however, that if the user selects the original first interpretation, the text may not have changed. Also, if you have placed other, non-Ink items in the menu, the user may have selected one of those instead of having made a selection from amongst the Ink alternates. So if you populate the menu with items unrelated to Ink, you may need to use the second method.

  - ❏ Call the Menu Manager function `CopyMenuItemTextAsCFString` supplying the values that were returned in the parameters `outMenuID` and `outMenuItem` when you called the Menu Manager function `ContextualMenuSelect` to display the menu.

    This method will always return the user's selection, whether it is from amongst the Ink alternates or not, though the text interpretation still may or may not have changed, and it is your application's responsibility to determine the proper outcome of the user's selection.

- The next time you need to show the menu, you must rebuild the menu before you insert the list of interpretations.

  Delete items you no longer want in the menu and then reinsert the appropriate menu items, making sure that you map the Ink Services alternates correctly to your menu. Text alternates supplied by Ink Services can be identified by the command ID `'inka'`. You can find out how many text alternates items are in the menu using the following code:

  ```
  alternateItemsCount = CountMenuItemsWithCommandID (myAlternatesMenu,
                                      'inka');
  ```

  If there are Ink Services items in the menu, you must delete the previous interpretations before you can add the current interpretations. If the Ink Services menu item had been placed at the beginning of the menu, you can do so with the following code:

  ```
  if (alternateItemsCount > 0)
          DeleteMenuItems (myAlternatesMenu, 1, alternateItemsCount);
  ```

You would also have to remove the single separator item identified by `'inks'` and the Ink data item identified by `'inkd'`. Or you could simply dispose of the old `menuRef` and create and populate a new one.

When a user selects an item from the list of alternates, Ink Services reorders the internal alternates list within the source Ink text object (`InkTextRef`). Thus the user's choice persists in the system data structures without requiring your application to call any additional functions. If it is important for your application to maintain the original order of alternates, then you must use your own internal data structures to keep track of the original list.

Implementing Deferred Recognition

Deferred recognition is the ability to convert pen strokes to text at some time other than when the strokes are first written by the user. If you want to implement deferred recognition, your application must be responsible for collecting Ink input and deciding when recognition occurs. Your application can draw its own Ink by either disabling Ink Services or by requesting that Ink Services doesn't process events it would otherwise have handled. Your application can then access all relevant data directly from standard mouse events.

To implement deferred recognition, your application must perform the following tasks:

- Inform Ink Services not to handle Ink events.

  You can accomplish with the following call:

  ```
  InkSetApplicationRecognitionMode (kInkWriteNowhereInApp);
  ```

- Install a Carbon event handler to gather tablet data.

  Your handler should handle `mouseDown`, `mouseUp`, and `mouseDragged` events. You should obtain the mouse location, tablet data, and key modifiers from each mouse event your application receives.

  You obtain the mouse location by extracting the event parameter `kEventParamMouseLocation`. You get the tablet data record by extracting the event parameter `kEventParamTabletPointRec`. You obtain the key modifiers (if any) in use by extracting the event parameter `kEventParamKeyModifiers`. The following code shows how to obtain the relevant data from the event parameters:

  ```
  OSStatus = noErr;
  ```

```
HIPoint location;
TabletPointRec tabletPt;
UInt32 modifiers;

// Get the full-resolution, floating point mouse location
status = GetEventParameter (myInEvent,
                    kEventParamMouseLocation,
                    typeHIPoint,
                    NULL, // On output, the actual type
                    sizeof (location),
                    NULL, // On output, the actual size
                    &location);
require_noerr (status, TabletEvent_err); /* macro to handle error */

// Get the tablet data record
status = GetEventParameter (myInEvent,
                    kEventParamTabletPointRec,
                    typeTabletPointRec,
                    NULL,
                    sizeof (TabletPointRec),
                    NULL,
                    (void*) &tabletPt);
require_noerr ( status, TabletEvent_err);

// Get any keyboard modifiers in use
status = GetEventParameter (myInEvent,
                    kEventParamKeyModifiers,
                    typeUInt32,
                    NULL,
                    sizeof (modifiers),
                    NULL,
                    &modifiers);
require_noerr (status, TabletEvent_err);
```

If it is useful for your application, you can also obtain tablet proximity data. Proximity data defines events which occur when the stylus is near, but not touching, the tablet.

Note: You could gather the same data by handling the event `kEventInkPoint`, and while handling mouse events directly may incur slightly less overhead than handling Ink point events, gathering your own Ink in this fashion requires your application to manage all of the pen-vs.-mouse issues (has the pen moved far enough, soon enough, is the pen placed in an instant mouser, and so forth).

Your application would also have to manage event coalescing and phrase termination itself. Managing these requires additional work for you and is likely to produce a different user experience in your application. Thus we recommend against managing real-time user handwriting input using mouse events, and only show this example to indicate how raw data can be handled for applications where that is necessary and appropriate.

- Store tablet data.

  As your application collects tablet-generated data, it must store the data (`HIPoint`, `TabletPointRec`, and any key modifiers) in any way that makes sense for your application. However, it is advisable not to use an array. The number of points that can be generated by a single stroke is variable, and may be quite large. Using an array would require you to allocate large arrays and to check for an array-bounds overrun.

- Add the stroke to the current Ink phrase.

When a stroke has been completed (as indicated by the first mouse-up event after a stroke has been initiated), you can add it to the current Ink phrase by building an array of `InkPoint` data structures, and filling the structures with the collected data. (The `InkPoint` data structure holds mouse location (`HIPoint`), tablet data (`TabletPointRec`), and key modifier information (`UInt32`).) The data must be in the chronological order in which it was generated.

Then, you must call the function `InkAddStrokeToCurrentPhrase` to add the stroke to the current Ink phrase. When you call the function, you must pass the number of structures in the array, as well as the array of `InkPoint` structures, as shown in the following code:

```
UInt32 myPointCount;
InkPoint * myInkPointArray;

InkAddStrokeToCurrentPhrase (myPointCount, myInkPointArray);
```

It's best to dynamically assemble and destroy the `InkPoint` array when you need to call this function.

- When your application determines that a phrase is complete and ready to be recognized, it can call the function `InkTerminateCurrentPhrase` with the `iSource` parameter set to `kInkSourceApplication`.

  Calling this function terminates the phrase and triggers the Ink recognizer.

- Handle the Ink event (`kEventInkText` or `kEventInkGesture`) that is generated by the recognizer when recognition is complete for the phrase.

A similar approach may be used to import larger quantities of data, gathered in an offline mode, potentially using a proprietary data format, such as for a digital pen that stores ink written on paper pads. Converting such data into `InkPoint` arrays and invoking the functions `InkAddStrokeToCurrentPhrase` and `InkTerminateCurrentPhrase` would allow such devices to use the recognition services provided by Ink Services.

Document Revision History

This table describes the changes to *Using Ink Services in Your Application*.

Date	Notes
2003-07-24	Made changes to the text and source code in the section "Handling Phrase Termination" (page 26) to reflect the use of the `iSource` input parameter to the functions `InkSetPhraseTerminationMode` and `InkTerminateCurrentPhrase`.
	In the section "Implementing a Correction Model" (page 33) changed "Menu items" to "Text alternates" in the sentence "Text alternates supplied by Ink Services can be identified by the command ID `'inka'`." This reflects the addition to the Ink Services API of two command IDs for the separator `'inks'` and Ink data `'inkd'` menu items. See the *Ink Services Reference* for more information.
	Made a change to the section "Implementing Deferred Recognition" (page 35) to reflect the use of the `iSource` input parameter to the function `InkTerminateCurrentPhrase`.
2003-06-19	First release of this document. This is a preliminary version.

Glossary

deferred recognition The process of recognizing an ink phrase that was drawn by the user at an earlier time.

event coalescing See mouse event coalescing.

gesture A handwritten mark that is recognized as having a special meaning, such as, Select All, Cut, and Copy.

Ink Raw data that represents the input drawn by the user with the stylus.

Ink pad The part of the Ink window that provides a simple note pad interface where handwritten input is converted into editable text.

Ink phrase The grouping of ink data created by the recognition system, based on the timing and spacing of the user's handwriting. In Roman languages, an ink phrase is typically a short string of characters with no spaces between them such as an individual character, several characters, a word, or, an entire URL. For most situations an Ink phrase is equivalent to a word.

Ink server The component of Ink technology that manages the recognizer, the language model, and the Ink window.

Ink text Words written in electronic ink.

Ink input method A low-level task which takes the user input and then draws the appropriate data on the screen. In effect, converting physical pen strokes into electronic Ink.

Ink text object An opaque object that contains information about an Ink phrase.

Ink toolbar The toolbar that appears at the top of the Ink window.

Ink window Comprised of the Ink toolbar and the Ink pad, allows the user to control various aspects of Ink and to enter Ink input.

Ink writing guides The lines (alternating solid and broken) that appear when a user is writing directly into an application.

instant mousing area An area in which stylus input is interpreted as mouse input; the system "instantly" interprets the stylus as a mouse in these special places and ink is not generated.

mouse event coalescing A process that merges `mouseMoved` and `mouseDragged` events by checking to see if one of these events exists in the event queue, and if it does, updating the queue with the position and delta information from the more recently-generated event.

pen See stylus.

pen event A mouse event that contains tablet data.

phrase termination Defines when Ink input should be processed by the recognizer.

recognized text Ink words processed by the recognition system.

recognizer The algorithmic component of Ink Services that identifies written text and gestures.

searchable Ink Ink that remains visible to the user as ink, but for which recognition has taken place.

stroke An array of points that define the path of the stylus, starting with a stylus-down event and ending when the stylus is lifted.

stylus The hand held instrument used to enter data into the computer. Also referred to as a pen.

targeted gesture A gesture that has a defined hot spot that an application can use to determine the area to which the gesture should apply.

tentative gesture Ink that the system treats tentatively as a gesture until your application either confirms the Ink is indeed a gesture or informs the system the Ink is not a gesture. The Join gesture is the only tentative gesture.

termination mode The conditions that define the end of an Ink phrase.

untargeted gesture A gesture that does not have a defined hot spot. An application should apply the gesture to the current selection or insertion point.