

---

# NSInputServiceProvider Protocol Reference

[Cocoa > Events & Other Input](#)



2007-04-02



Apple Inc.  
© 2007 Apple Inc.  
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.  
1 Infinite Loop  
Cupertino, CA 95014  
408-996-1010

Apple, the Apple logo, Cocoa, Mac, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries.

Simultaneously published in the United States and Canada.

**Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

**IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY**

**DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.**

**THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.**

**Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.**

# Contents

---

## **NSInputServiceProvider Protocol Reference 5**

---

Overview 5

Tasks 5

    Getting Input Service Provider Information 5

    Handling Events 5

Instance Methods 6

    activeConversationChanged:toNewConversation: 6

    activeConversationWillChange:fromOldConversation: 7

    canBeDisabled 7

    doCommandBySelector:client: 7

    inputClientBecomeActive: 8

    inputClientDisabled: 8

    inputClientEnabled: 9

    inputClientResignActive: 9

    insertText:client: 9

    markedTextAbandoned: 10

    markedTextSelectionChanged:client: 10

    terminate: 10

    wantsToDelayTextChangeNotifications 11

    wantsToHandleMouseEvents 11

    wantsToInterpretAllKeystrokes 12

---

## **Document Revision History 13**

---

## **Index 15**

---



# NSInputServiceProvider Protocol Reference

---

<b>Adopted by</b>	NSInputServer
<b>Framework</b>	/System/Library/Frameworks/AppKit.framework
<b>Availability</b>	Available in Mac OS X v10.0 and later.
<b>Companion guide</b>	Text Input Management
<b>Declared in</b>	NSInputServer.h

## Overview

The NSInputServiceProvider protocol embodies most of the functionality of NSInputServer.

There are two ways you might use this protocol:

- You can subclass NSInputServer and create an instance of your subclass. Your subclass must override most or all of the NSInputServiceProvider protocol methods.
- You can create an NSInputServer object and designate a delegate. The delegate must implement the NSInputServiceProvider protocol.

All messages in this protocol are sent by the client text view except `insertText:client:` (page 9) and `doCommandBySelector:client:` (page 7), which are sent by "NSInputManager".

## Tasks

### Getting Input Service Provider Information

- `canBeDisabled` (page 7)  
Returns YES if the receiver can be disabled when the sender is not a text view, NO

### Handling Events

- `wantsToDelayTextChangeNotifications` (page 11)  
A YES return value tells the client that only a call to its `insertText:client:` (page 9) method constitutes a modification to its text storage.

- [wantsToHandleMouseEvents](#) (page 11)  
Returns YES if the client should forward all mouse events within the text view to the input server.
- [wantsToInterpretAllKeystrokes](#) (page 12)  
Returns YES if the server wants all keystrokes to be sent to it as characters.
- [doCommandBySelector:client:](#) (page 7)  
Handle the command identified by *aSelector*.
- [insertText:client:](#) (page 9)  
Interpret the characters in *aString*, which is actually always an NSString.
- [activeConversationChanged:toNewConversation:](#) (page 6)  
Keyboard focus just switched from another text view to this one.
- [activeConversationWillChange:fromOldConversation:](#) (page 7)  
Keyboard focus is about to move away from this text view.
- [inputClientBecomeActive:](#) (page 8)  
The client, *sender*, has become active.
- [inputClientEnabled:](#) (page 9)  
A text view in the client, *sender*, has become the key-receiving first responder.
- [inputClientDisabled:](#) (page 8)  
A text view in the client, *sender*, has ceased to be the key-receiving first responder.
- [inputClientResignActive:](#) (page 9)  
The client, *sender*, is about to become inactive.
- [markedTextAbandoned:](#) (page 10)  
Abandon any marked text state that may be in process.
- [markedTextSelectionChanged:client:](#) (page 10)
- [terminate:](#) (page 10)  
The client application is quitting.

## Instance Methods

### **activeConversationChanged:toNewConversation:**

Keyboard focus just switched from another text view to this one.

```
- (void)activeConversationChanged:(id)sender
    toNewConversation:(NSInteger)newConversation
```

#### **Discussion**

This is called only when switching within the same application. *sender* can be cast to NSTextInput.

#### **Availability**

Available in Mac OS X v10.0 and later.

#### **See Also**

- [activeConversationWillChange:fromOldConversation:](#) (page 7)
- `conversationIdentifier` (NSTextInput)

**Declared In**

NSInputServer.h

**activeConversationWillChange:fromOldConversation:**

Keyboard focus is about to move away from this text view.

```
- (void)activeConversationWillChange:(id)sender
    fromOldConversation:(NSInteger)oldConversation
```

**Discussion**

This is called only when switching within the same application. *sender* can be cast to `NSTextInput`.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- [activeConversationChanged:toNewConversation:](#) (page 6)
- `conversationIdentifier` (`NSTextInput`)

**Declared In**

NSInputServer.h

**canBeDisabled**

Returns YES if the receiver can be disabled when the sender is not a text view, NO

```
- (BOOL)canBeDisabled
```

**Discussion**

otherwise.

**Availability**

Available in Mac OS X v10.0 and later.

**Declared In**

NSInputServer.h

**doCommandBySelector:client:**

Handle the command identified by *aSelector*.

```
- (void)doCommandBySelector:(SEL)aSelector client:(id)sender
```

**Discussion**

The command can be from the set of `NSResponder` action methods or from the set of selector values in the `DefaultKeyBindings` dictionary referenced in the input server's "Info" file. *sender* can be cast to `NSTextInput`.

If you are subclassing `NSInputServer`, there is no need to override this method in the subclass. All you have to do is implement in the subclass the command methods you want to handle. If you do need to override this method, then you must call `super` for commands not handled.

If your `NSInputServer` uses a delegate, the delegate's implementation of this method must call `[sender doCommandBySelector:aSelector]` for commands it does not handle.

#### Availability

Available in Mac OS X v10.0 and later.

#### See Also

- [doCommandBySelector:](#) (`NSTextInput`)

#### Declared In

`NSInputServer.h`

## inputClientBecomeActive:

The client, *sender*, has become active.

```
- (void)inputClientBecomeActive:(id)sender
```

#### Discussion

This is called when the client application starts up and whenever it becomes active after being inactive. *sender* can be cast to `NSTextInput`.

#### Availability

Available in Mac OS X v10.0 and later.

#### See Also

- [inputClientEnabled:](#) (page 9)  
- [inputClientResignActive:](#) (page 9)

#### Declared In

`NSInputServer.h`

## inputClientDisabled:

A text view in the client, *sender*, has ceased to be the key-receiving first responder.

```
- (void)inputClientDisabled:(id)sender
```

#### Discussion

[inputClientResignActive:](#) (page 9) may also be called just after this is called. *sender* can be cast to `NSTextInput`.

#### Availability

Available in Mac OS X v10.0 and later.

#### See Also

- [inputClientEnabled:](#) (page 9)  
- [inputClientResignActive:](#) (page 9)

#### Declared In

`NSInputServer.h`



## inputClientEnabled:

A text view in the client, *sender*, has become the key-receiving first responder.

```
- (void)inputClientEnabled:(id)sender
```

### Discussion

This is called the first time any text view becomes enabled after client application activation and again whenever focus switches to a text view. [inputClientBecomeActive:](#) (page 8) may have been called just before this is called. *sender* can be cast to `NSTextInput`.

### Availability

Available in Mac OS X v10.0 and later.

### See Also

- [inputClientBecomeActive:](#) (page 8)
- [inputClientDisabled:](#) (page 8)

### Declared In

`NSInputServer.h`

## inputClientResignActive:

The client, *sender*, is about to become inactive.

```
- (void)inputClientResignActive:(id)sender
```

### Discussion

This is called when the client application quits and whenever it is deactivated. *sender* can be cast to `NSTextInput`.

### Availability

Available in Mac OS X v10.0 and later.

### See Also

- [inputClientBecomeActive:](#) (page 8)
- [inputClientDisabled:](#) (page 8)
- [terminate:](#) (page 10)

### Declared In

`NSInputServer.h`

## insertText:client:

Interpret the characters in *aString*, which is actually always an `NSString`.

```
- (void)insertText:(id)aString client:(id)sender
```

### Discussion

Here is where you do the interpreting of keyboard input. If your server's interpretation is disabled or the characters in *aString* are not of interest to the server, you can simply pass *aString* along to the sender's `insertText:` method. *sender* can be cast to `NSTextInput`.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- [insertText:](#) (NSTextInput)

**Declared In**

NSInputServer.h

**markedTextAbandoned:**

Abandon any marked text state that may be in process.

- (void)markedTextAbandoned:(id) *sender*

**Discussion**

This can happen if the user clicks the mouse outside of the marked text area or if the window containing the text view closes. The client can do what it wants with the marked text. `NSTextView` leaves it as inserted text. *sender* can be cast to `NSTextInput`.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- [markedTextSelectionChanged:client:](#) (page 10)

- [markedTextAbandoned:](#) (NSInputManager)

**Declared In**

NSInputServer.h

**markedTextSelectionChanged:client:**

- (void)markedTextSelectionChanged:(NSRange) *newSelection* client:(id) *sender*

**Discussion**

The user selected a portion of the marked text or clicked at the beginning or end of marked text or somewhere in between. *sender* can be cast to `NSTextInput`.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- [markedTextAbandoned:](#) (page 10)

- [markedTextSelectionChanged:client:](#) (NSInputManager)

**Declared In**

NSInputServer.h

**terminate:**

The client application is quitting.

- (void)terminate:(id) *sender*

#### Discussion

This is called after [inputClientResignActive:](#) (page 9). *sender* can be cast to `NSTextInput`.

#### Availability

Available in Mac OS X v10.0 and later.

#### See Also

- [inputClientResignActive:](#) (page 9)

#### Declared In

NSInputServer.h

## wantsToDelayTextChangeNotifications

A YES return value tells the client that only a call to its [insertText:client:](#) (page 9) method constitutes a modification to its text storage.

- (BOOL)wantsToDelayTextChangeNotifications

#### Discussion

A NO return value tells the client that all text given to it, whether marked text or not, should constitute a modification to its text storage. A YES return value tells the client that only unmarked text given to it should constitute a modification to its text storage. The client may for example want to filter all text that is part of a modification but leave marked text unfiltered.

#### Availability

Available in Mac OS X v10.0 and later.

#### See Also

- [wantsToDelayTextChangeNotifications](#) (NSInputManager)

#### Declared In

NSInputServer.h

## wantsToHandleMouseEvents

Returns YES if the client should forward all mouse events within the text view to the input server.

- (BOOL)wantsToHandleMouseEvents

#### Discussion

If the server needs to implement the `NSInputServerMouseTracker` protocol, return YES.

#### Availability

Available in Mac OS X v10.0 and later.

#### See Also

- [wantsToHandleMouseEvents](#) (NSInputManager)

#### Declared In

NSInputServer.h

## wantsToInterpretAllKeystrokes

Returns YES if the server wants all keystrokes to be sent to it as characters.

- (BOOL)wantsToInterpretAllKeystrokes

### Discussion

If this method returns NO, control key combinations and function keys (the arrow keys, PageDown, F5, and so on) are delivered to the input server via the key binding mechanism and [doCommandBySelector:client:](#) (page 7).

The Unicode values for the characters representing keyboard function keys (the arrow keys, PageDown, F5, and so on) names like `NSUpArrowFunctionKey`, and are documented in `NSEvent`. Control-key combinations are the usual ASCII control character codes.

For more information on key bindings, see “About Key Bindings”.

### Availability

Available in Mac OS X v10.0 and later.

### See Also

- [wantsToInterpretAllKeystrokes \(NSInputManager\)](#)

### Declared In

`NSInputServer.h`

# Document Revision History

---

This table describes the changes to *NSInputServiceProvider Protocol Reference*.

Date	Notes
2007-04-02	Made editorial improvements.
2006-05-23	First publication of this content as a separate document.

## REVISION HISTORY

### Document Revision History

# Index

---

## A

---

activeConversationChanged:toNewConversation:  
    **protocol instance method 6**  
activeConversationWillChange:fromOldConversation:  
    **protocol instance method 7**

## C

---

canBeDisabled **protocol instance method 7**

## D

---

doCommandBySelector:client: **protocol instance  
    method 7**

## I

---

inputClientBecomeActive: **protocol instance method  
    8**  
inputClientDisabled: **protocol instance method 8**  
inputClientEnabled: **protocol instance method 9**  
inputClientResignActive: **protocol instance method  
    9**  
insertText:client: **protocol instance method 9**

## M

---

markedTextAbandoned: **protocol instance method 10**  
markedTextSelectionChanged:client: **protocol  
    instance method 10**

## T

---

terminate: **protocol instance method 10**

## W

---

wantsToDelayTextChangeNotifications **protocol  
    instance method 11**  
wantsToHandleMouseEvents **protocol instance method  
    11**  
wantsToInterpretAllKeystrokes **protocol instance  
    method 12**