

---

# IMKServerInput Protocol Reference

User Experience



2009-05-06



Apple Inc.  
© 2009 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

---

## **IMKServerInput Protocol Reference 5**

---

Overview	5
Tasks	5
Supporting Key Binding	5
Unpacking Text Data	5
Receiving Events Directly from the Text Services Manager	6
Committing a Composition	6
Getting Input Strings and Candidates	6
Instance Methods	6
candidates:	6
commitComposition:	7
composedString:	7
didCommandBySelector:client:	8
handleEvent:client:	8
inputText:client:	9
inputText:key:modifiers:client:	9
originalString:	10
Constants	10
Info Dictionary Keys	10

---

## **Document Revision History 11**

---

## **Index 13**

---



# IMKServerInput Protocol Reference

(informal protocol)

---

<b>Framework</b>	System/Library/Frameworks/InputMethodKit.framework
<b>Declared in</b>	IMKInputController.h

## Overview

`IMKServerInput` is an informal protocol that defines methods for receiving text events. This is intentionally not a formal protocol because there are three ways to receive events. An input method chooses one of the following approaches and implements the appropriate methods:

- **Key binding.** In this approach the system tries to map each key-down event to an action method that the input method has implemented. If successful (action method found), the system calls `didCommandBySelector:client:`. If unsuccessful (action method not found), the system calls `inputText:client:`. For this approach you need to implement `inputText:client:` (page 9) and `didCommandBySelector:client:` (page 8).
- **Text data only.** In this approach, you opt to receive all key events without the key binding, and then unpack the relevant text data. Key events are broken down into the Unicodes, the key code that generated them, and modifier flags. This data is then sent to the `inputText:key:modifiers:client:` (page 9) method, which you need to implement.
- **Handle all events.** In this approach, you receive events directly from the Text Services Manager as `NSEvent` objects. You must implement `handleEvent:client:` (page 8) method.

## Tasks

### Supporting Key Binding

- `inputText:client:` (page 9) *required method*  
Handles key down events that do not map to an action method. (required)
- `didCommandBySelector:client:` (page 8) *required method*  
Processes a command generated by user action such as typing certain keys or pressing the mouse button. (required)

### Unpacking Text Data

- `inputText:key:modifiers:client:` (page 9) *required method*  
Receives Unicode, the key code that generated it, and any modifier flags. (required)

## Receiving Events Directly from the Text Services Manager

- `handleEvent:client:` (page 8) *required method*  
Handles key down and mouse events. (required)

## Committing a Composition

- `commitComposition:` (page 7) *required method*  
Informs the controller that the composition should be committed. (required)

## Getting Input Strings and Candidates

- `composedString:` (page 7) *required method*  
Return the current composed string. (required)
- `originalString:` (page 10) *required method*  
Return the a string that consists of the precomposed unicode characters. (required)
- `candidates:` (page 6) *required method*  
Returns an array of candidates. (required)

## Instance Methods

### **candidates:**

Returns an array of candidates. (required)

- `(NSArray*)candidates:(id)sender`

#### **Parameters**

*sender*

The client object requesting the candidates.

#### **Return Value**

An array of candidates. The returned array should be an autoreleased object.

#### **Discussion**

An input method should look up its currently composed string and return a list of candidate strings that that string might map to.

#### **Availability**

Available in Mac OS X v10.5 and later.

#### **Declared In**

IMKInputController.h

**commitComposition:**

Informs the controller that the composition should be committed. (required)

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

**Parameters**

*sender*

The client object requesting the input method to commit the composition.

**Discussion**

If an input method implements this method, it is called when the client wants to end the composition session immediately. A typical response would be to call the `insertText` method of the client and then clean up any per-session buffers and variables. After receiving this message an input method should consider the given composition session finished.

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

NumberInput\_IMKit\_Sample

**Declared In**

IMKInputController.h

**composedString:**

Return the current composed string. (required)

```
- (id)composedString:(id)sender
```

**Parameters**

*sender*

The client object requesting the string.

**Return Value**

The current composed string, which can be an `NSString` or `NSAttributedString` object. The returned object should be an autoreleased object.

**Discussion**

A composed string refers to the buffer that an input method typically maintains to mirror the text contained in the active inline area. It is called the composed string to reflect the fact that the input method composed the string by converting the characters input by the user. In addition, using the term composed string makes it easier to differentiate between an input method buffer and the text in the active inline area that the user sees.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

IMKInputController.h

## didCommandBySelector:client:

Processes a command generated by user action such as typing certain keys or pressing the mouse button. (required)

```
- (BOOL)didCommandBySelector:(SEL)aSelector client:(id)sender
```

### Parameters

*aSelector*

The action associated with the key down event. The selector can be an action specified in the input method dictionary of keys and actions (that is, an action specific to the input method) or one of the `NSResponder` action methods such as `insertNewLine:` or `deleteBackward:`. By definition such action methods do not return a value.

*sender*

The client object sending the key down event.

### Return Value

YES if the command is handled; NO if the command is not handled. If not handled, the event passes to the client.

### Discussion

This method is called when the system binds a key down event to an action method. If you implement this method you should test if it is appropriate to call the action method before actually calling it, because calling the action method implies that you agree to handle the command. Suppose you have implemented a version of `insertNewLine:` that terminates the conversion session and sends the fully converted text to the client. However, if your conversion buffer is empty, you want the application to receive the return key that triggered the call to `insertNewLine:`. In that case, when `didCommandBySelector:client:` is called you should test your buffer before calling your implementation of `insertNewLine:`. If the buffer is empty, return NO to indicate that the return key should be passed on to the application. If the buffer is not empty, call `insertNewLine:` and then return YES as the result of `didCommandBySelector:client:`.

### Availability

Available in Mac OS X v10.5 and later.

### See Also

- [inputText:client:](#) (page 9)

### Declared In

IMKInputController.h

## handleEvent:client:

Handles key down and mouse events. (required)

```
- (BOOL)handleEvent:(NSEvent*)event client:(id)sender
```

### Parameters

*event*

The event to handle.

*sender*

The client object sending the event.

### Return Value

YES if the event is handled; otherwise NO.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

IMKInputController.h

**inputText:client:**

Handles key down events that do not map to an action method. (required)

```
- (BOOL)inputText:(NSString*)string client:(id)sender
```

**Parameters**

*string*

The key down event, which is the text input by the client.

*sender*

The client object sending the key down events.

**Return Value**

YES if the input is accepted; otherwise NO.

**Discussion**

An input method should implement this method when using key binding (that is, it implements [didCommandBySelector:client:](#) (page 8)).

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

IMKInputController.h

**inputText:key:modifiers:client:**

Receives Unicode, the key code that generated it, and any modifier flags. (required)

```
- (BOOL)inputText:(NSString*)string key:(NSInteger)keyCode
  modifiers:(NSUInteger)flags client:(id)sender
```

**Parameters**

*string*

The text input by the client.

*keyCode*

The key code for the associated Unicode.

*flags*

The modifier flags.

*sender*

The client object.

**Return Value**

YES if the input is accepted; otherwise NO.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

IMKInputController.h

**originalString:**

Return the a string that consists of the precomposed unicode characters. (required)

```
- (NSAttributedString*)originalString:(id)sender
```

**Parameters**

*sender*

The client object requesting the original string.

**Return Value**

The original string of precomposed unicode characters. If an input method stores the original input text, it returns that text. The return value is an attributed string so that the input method can restore changes they made to the font, and other attributes, if necessary. The returned object should be an autoreleased object.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

IMKInputController.h

## Constants

### Info Dictionary Keys

Constants for keys used to look up information in the info dictionary.

```
extern const NSString *kIMKCommandMenuItemName;
extern const NSString *kIMKCommandClientName;
```

**Constants**

kIMKCommandMenuItemName

Used to look up the `NSMenuItem` object that is passed to menu item actions.

Available in Mac OS X v10.5 and later.

Declared in `IMKInputController.h`.

kIMKCommandClientName

Used to look up the client object; the client conforms to the `IMKInputText` and `NSObject` protocols.

Available in Mac OS X v10.5 and later.

Declared in `IMKInputController.h`.

# Document Revision History

---

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

Date	Notes
2009-05-06	Clarified constant descriptions.
2007-06-06	New document that describes the informal protocol used to receive text events.

## REVISION HISTORY

### Document Revision History

# Index

---

## C

---

`candidates:` <NSObject> instance method [6](#)  
`commitComposition:` <NSObject> instance method [7](#)  
`composedString:` <NSObject> instance method [7](#)

## D

---

`didCommandBySelector:client:` <NSObject> instance method [8](#)

## H

---

`handleEvent:client:` <NSObject> instance method [8](#)

## I

---

**Info Dictionary Keys** [10](#)  
`inputText:client:` <NSObject> instance method [9](#)  
`inputText:key:modifiers:client:` <NSObject> instance method [9](#)

## K

---

`kIMKCommandClientName` constant [10](#)  
`kIMKCommandMenuItemName` constant [10](#)

## O

---

`originalString:` <NSObject> instance method [10](#)