
Action Messages

Data Management: Event Handling



2002-11-12



Apple Inc.
© 2002 Apple Computer, 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, 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

Introduction to Action Messages 5

Organization of This Document 5

Targets and Actions 7

About Action Cells 9

Document Revision History 11

Introduction to Action Messages

Action messages (or methods) and target objects are part of the mechanism by which NSControls respond to user actions and enable users to communicate their intentions to an application.

Organization of This Document

[“About Action Cells”](#) (page 9) gives basic information on action cells. [“Targets and Actions”](#) (page 7) describes the interactions between action messages and their target objects.

Targets and Actions

Action messages (or methods) and target objects are part of the mechanism by which NSControls respond to user actions and enable users to communicate their intentions to an application. A target is an object that NSControl uses as the receiver of action messages. The target's class defines an action message to enable its instances to respond to these messages, which are sent as users click or otherwise manipulate the NSControl. NSControl's `sendAction:to:` asks the NSApplication object `NSApp` to send an action message to the NSControl's target object.

An action method takes only one argument: the sender. The sender may be either the NSControl that sends the action message or, on occasion, another object that the target should treat as the sender. When it receives an action message, a target can return messages to the sender requesting additional information about its status.

You can also set the target to `nil` and allow it to be determined at run time. When the target is `nil`, the NSApplication object must look for an appropriate receiver. It conducts its search in a prescribed order, by following the responder chain until it finds an object that can respond to the message:

- It begins with the first responder in the key window and follows `nextResponder` links up the responder chain to the NSWindow's content view.
- It tries the NSWindow object and then the NSWindow's delegate.
- If the main window is different from the key window, it then starts over with the first responder in the main window and works its way up the main window's responder chain to the NSWindow object and its delegate.
- Next, the NSApplication object tries to respond itself. If it can't respond, it tries its own delegate. NSApp and its delegate are the receivers of last resort.

NSControl provides methods for setting and using the target object and the action method. However, these methods require that an NSControl's cell (or cells) be NSActionCells or custom cells that hold action and target as instance variables and can respond to the NSControl methods.

About Action Cells

An action cell defines an active area inside a control (an instance of `NSControl` or one of its subclasses). As an `NSControl`'s active area, an action cell does three things: it usually performs display of text or an icon; it provides the `NSControl` with a target and an action; and it handles mouse (cursor) tracking by properly highlighting its area and sending action messages to its target based on cursor movement.

`NSActionCell` implements the target object and action method as defined by its superclass, `NSCell`. As a user manipulates an `NSControl`, `NSActionCell`'s `trackMouse:inRect:ofView:untilMouseUp:` method (inherited from `NSCell`) updates its appearance and sends the action message to the target object with the `NSControl` object as the only argument. See [“Targets and Actions”](#) (page 7).

Usually, the responsibility for an `NSControl`'s appearance and behavior is completely given over to a corresponding `NSActionCell`. (`NSMatrix`, and its subclass `NSForm`, are `NSControls` that don't follow this rule.)

A single `NSControl` may have more than one `NSActionCell`. To help identify it in this case, every `NSActionCell` has an integer tag. Note, however, that no checking is done by the `NSActionCell` object itself to ensure that the tag is unique. See `NSMatrix` for an example of a subclass of `NSControl` that contains multiple `NSActionCells`.

Many of the methods that define the contents and look of an `NSActionCell`, such as `setFont:` and `setBordered:`, are reimplementations of methods inherited from `NSCell`. They're overridden to ensure the `NSActionCell` is redisplayed when “visual” attributes change.

Document Revision History

This table describes the changes to *Action Messages*.

Date	Notes
2002-11-12	Revision history was added to existing topic. It will be used to record changes to the content of the topic.

