Search Fields

Cocoa > User Experience

2008-02-08

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

Simultaneously published in the United States and Canada.

Contents

Figures and Listings

Introduction to Search Fields

A search field provides a standard user interface for searching and is a feature in applications like Mail, Safari, and Address Book.

You use the Cocoa classes `NSSearchFieldCell` and `NSSearchField` to implement search fields. Other than sending the action message, however, `NSSearchFieldCell` and `NSSearchField` provide no direct support for textual searches.

Organization of This Document

This programming topic contains the following articles:

- "Understanding Search Fields" (page 9) describes the features of search fields and the classes used to implement them.
- "Adding a Search Field to Your Application" (page 11) describes how to add a search field to your application, using either Interface Builder or a programming language.
- "Configuring a Search Menu" (page 13) describes setting up the search field's pop-up icon menu to show recent search strings and search categories.
- "Implementing the Target" (page 17) describes how to implement the search field's target's action method.
- "Customizing Your Search Field's Appearance" (page 19) describes how to change the appearance of a search field programatically.

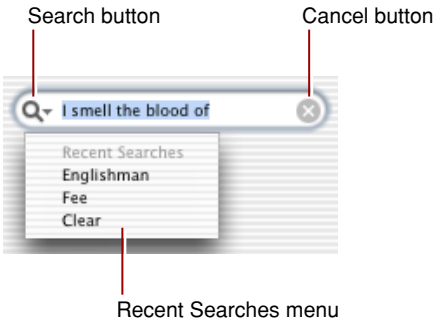
See Also

- *Apple Human Interface Guidelines* provides guidelines on when to use particular interface items and how to position them.
- *Search Kit Reference* describes a powerful and streamlined C language framework for indexing and searching text in most human languages.

Understanding Search Fields

A **search field** is a rounded text field that displays text that the user can select or edit, and that sends its action message to its target when the user presses the Return key. It presents a standard user interface for searches, including a search button, a cancel button, and a pop-up icon menu for listing recent search strings and custom search categories. The search button includes a menu and the option to send the results while the user is typing or when the user presses the Return key. If there is no text in the search field, the cancel button is hidden. Figure 1 shows the major components of a search field.

Figure 1 A search field



A search field is implemented by two classes: **NSSearchFieldCell**, the cell that does most of the work, and **NSSearchField**, the control that contains that cell.

There are, broadly speaking, two ways to configure and use a search field—programmatically, or with Cocoa bindings.

If you configure a search field programmatically, you should set the target and action of the control or its cell to the receiver that is interested in the search request. Also, remember that `NSSearchFieldCell` and `NSSearchField` classes are direct subclasses of `NSTextFieldCell` and `NSTextField`, respectively, so you can use all the methods inherited from these classes.

Adding a Search Field to Your Application

You can create a search field programatically, but the easiest way to add a search field to your application is to use Interface Builder. Simply drag a search field from the Cocoa-Text palette to your window. and add it to a window.

Use the Attributes pane of the Show Info panel to set search-field-specific attributes:

- **Placeholder**, which specifies text that appears in the search field until the user enters text. You can also programmatically send the `setPlaceholderString:` message to the search field's cell.
- **Max Recents**, which specifies the maximum number of recent searches to show in the recents menu. This value can be from `0` to `254`, and defaults to `10`. When the limit is exceeded, the oldest search string on the menu is dropped. Setting the value to a negative value uses the default value; a value greater than `254` sets the maximum to `254`. The recents list is trimmed if there are more entries than the new maximum. You can also programmatically send the `setMaximumRecents:` message to the search field's cell.
- **Auto Save Name**, which if set, the recent search list is saved to an application preference using the name provided, and restored the next time the recents list is needed for the popup menu. You can also programmatically send the `setRecentsAutosaveName:` message. Setting the autosave name to `nil` does not clear out any saved lists. Setting the autosave name to a valid string discards any current recents and loads the recents from the user defaults.
- **Sends Whole Search String**, which specifies whether the search field sends the action message when the user presses the Return key or if it sends the message upon each keystroke (incremental search). You can also programmatically send the `setSendsWholeSearchString:` message to the search field's cell. By default, the cell's action is invoked during typing after a short delay.

You can connect the a search field to a menu template. The details of the menu's contents are described in "Configuring a Search Menu" (page 13).

You need to configure the targets and actions of the search field. This is described in "Implementing the Target" (page 17).

You can also specify that the search field use a controller to perform searches. In the Bindings pane, disclose the value binding and select the controller to bind to as well as the controller's key. In the Connections pane, make sure the correct target and action pair is selected. Additional information about using controllers can be found in *Cocoa Bindings Programming Topics*.

Configuring a Search Menu

A search field has a menu that users can access by clicking the small triangle next to the search button. The items in this pop-up icon menu have two purposes: To list recently entered search strings, and to list search categories for limiting the scope of the search. You can also specify *when* the search field sends its action message.

Generating Action Messages

A search field has two modes for generating action messages. The first mode is similar to a text field; a message is sent when a user clicks the search button or presses Return when the insertion point is in the search field. The second mode, which is the default, is for incremental searches; the search field sends an action message at each keystroke. Typically you set the mode in Interface Builder (see "Adding a Search Field to Your Application" (page 11)). To set the message-generation mode programmatically, send `setSendsWholeSearchString:` to the search field cell with an argument of `NO` for incremental searches and `YES` for regular searches.

Configuring a Menu Template

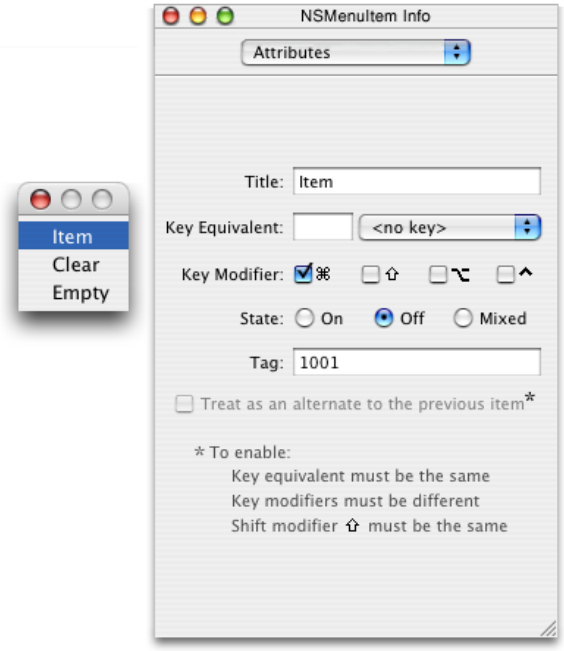
If you specify a menu in Interface Builder or programatically, the search button has a menu indicator and the search menu is displayed when the user clicks the search button. Do not modify the menu while it is in use by the search field. This is because a copy is not made.

To configure this pop-up menu, create a menu template (which is an `NSMenu` object) and populate this menu template with the items (`NSMenuItem` objects) you want to appear. Items in the menu can be given special meaning by using predefined tags described in the Constants section of `NSSearchFieldCell`.

You can specify the menu template used when generating the pop-up menu in the search field. This is easily done with Interface Builder. Or, if you have programmatically constructed the menu template, invoke `setSearchMenuTemplate:` to install it in the search field.

Figure 1 shows a simple menu template with a recents item, a clear item, and a no recents item.

Figure 1 Sample menu template with info pane



The menu item tag is specified in the Tag field. In Figure 1, the menu item is of the type `NSSearchFieldRecentsMenuItemTag`. Although not pictured, the tag for the Clear menu item is `NSSearchFieldClearRecentsMenuItemTag` and for the Empty menu item it's `NSSearchFieldNoRecentsMenuItemTag`. You can also add a menu item with the `NSSearchFieldRecentsTitleMenuItemTag` tag for an item that does not appear if there are no recent strings to display, or conversely a menu item with the `NSSearchFieldNoRecentsMenuItemTag` tag for an item that only appears if there are no recent strings to display.

Important: Currently, Interface Builder does not allow you to enter the symbolic name for these menu item tags. Instead you must enter the corresponding numeric value in the Tag field. For `NSSearchFieldRecentsTitleMenuItemTag` enter `1000`, for `NSSearchFieldRecentsMenuItemTag` enter `1001`, for `NSSearchFieldClearRecentsMenuItemTag` enter `1002`, and for `NSSearchFieldNoRecentsMenuItemTag` enter `1003`. This limitation does not apply to specifying the tag programatically.

You can also do all of this programatically. Listing 1 illustrates how you might set up the search menu template.

Listing 1 Setting the menu template for recent search strings

```
- (void) awakeFromNib
{
    NSMenu *cellMenu = [[[NSMenu alloc] initWithTitle:@"Search Menu"]
                autorelease];
    NSMenuItem *item;

    item = [[[NSMenuItem alloc] initWithTitle:@"Clear"
                                action:NULL
                                keyEquivalent:@""] autorelease];
    [item setTag:NSSearchFieldClearRecentsMenuItemTag];
```

```
    [cellMenu insertItem:item atIndex:0];

    item = [NSMenuItem separatorItem];
    [item setTag:NSSearchFieldRecentsTitleMenuItemTag];
    [cellMenu insertItem:item atIndex:1];

    item = [[[NSMenuItem alloc] initWithTitle:@"Recent Searches"
                                action:NULL
                                keyEquivalent:@""] autorelease];
    [item setTag:NSSearchFieldRecentsTitleMenuItemTag];
    [cellMenu insertItem:item atIndex:2];

    item = [[[NSMenuItem alloc] initWithTitle:@"Recents"
                                action:NULL
                                keyEquivalent:@""] autorelease];
    [item setTag:NSSearchFieldRecentsMenuItemTag];
    [cellMenu insertItem:item atIndex:3];

    id searchCell = [searchField cell];
    [searchCell setSearchMenuTemplate:cellMenu];
}
```

A menu is associated with your search field only if you specify one with Interface Builder. If you are creating the search field programatically and do not want a menu, set the menu template to `nil`.

Specifying a Search Category

To set up a search category, you create a menu item with a title and an action selector and add it to the menu template. The action selector identifies the method that is invoked using the target-action paradigm when the user selects the item; in this method you can set a flag or other value so that your search implementation knows how to limit the scope of its search. Typically you would add the menu item to the menu template in Interface Builder. You can also do it programmatically, as illustrated in Listing 2. This is useful if you create or update the menu dynamically, for example if the categories depend on what table columns are visible in a table view.

Listing 2 Setting the menu template with a search category

```
- (void) awakeFromNib
{
    NSMenu *cellMenu = [[[NSMenu alloc] initWithTitle:@"Search Menu"]
                        autorelease];
    NSMenuItem *item;

    item = [[[NSMenuItem alloc] initWithTitle:@"First Name"
                                      action:@selector(setSearchCategoryFrom:)
                                keyEquivalent:@""] autorelease];
    [item setTarget:self];
    [item setTag:1];
    [cellMenu insertItem:item atIndex:0];

    item = [[[NSMenuItem alloc] initWithTitle:@"Last Name"
                                      action:@selector(setSearchCategoryFrom:)
                                keyEquivalent:@""] autorelease];
    [item setTarget:self];
```

```
    [item setTag:2];
    [cellMenu insertItem:item atIndex:1];

    id searchCell = [searchField cell];
    [searchCell setSearchMenuTemplate:cellMenu];
}
```

Tags are optional for search categories, but they allow you to easily discriminate between menu items in the action method. If you choose to specify tags, their values must not be any of the values corresponding to `NSSearchFieldRecentsTitleMenuItemTag`, `NSSearchFieldRecentsMenuItemTag`, `NSSearchFieldClearRecentsMenuItemTag`, or `NSSearchFieldNoRecentsMenuItemTag`.

You also have to implement the action method invoked by the menu item—see "Search Using a Search Category" (page 17).

Implementing the Target

Because a search field is a control object, when a user does something to activate the search field, it sends an action message to a target object, telling it to perform the search. You configure the target and action in Interface Builder as you would any other control, as described in Communicating With Objects >The Target-Action Mechanism. Other than invoking the action method, `NSSearchFieldCell` and `NSSearchField` provide no support for textual searches. You must implement the search behavior yourself.

Simple Search

Typically you implement the action method in a controller object in your application. You need to find out what the search string is and then perform the search using the search term, as illustrated in Listing 1.

Listing 1 Simple action method for a search field's target

```
- (IBAction)updateFilter:sender
{
    NSString *searchString = [searchField stringValue];

    /*
     Method continues to perform the search and display the results...
     */
}
```

Search Using a Search Category

If you want to support search categories (see "Specifying a Search Category" (page 15)), you have to implement the action method invoked by the category menu items as well as that invoke by the search field itself. The category action method should record in some way the menu item that was selected. It might also set the placeholder string for the search menu, as illustrated in Listing 2.

Listing 2 Example search category menu item action method

```
- (IBAction)setSearchCategoryFrom:(NSMenuItem *)menuItem
{
    searchCategory = [menuItem tag];
    [[searchField cell] setPlaceholderString:[menuItem title]];
}
```

In the search field's action method, you need to find out what the search string is and then perform the search using the search term, taking into account any category that might have been set. A common way to represent the search is as an instance of `NSPredicate`. In Listing 3, `searchCategory` is set in the `setSearchCategoryFrom:` method illustrated in Listing 2.

Listing 3 Example action method for a search field's target

```
- (IBAction)updateFilter:sender
{
    /*
     Create a predicate based on what is the current string in the
     search field and the value of searchCategory.
     */
    NSPredicate *predicate = nil;

    NSString *searchString = [searchField stringValue];

    if ((searchString != nil) && (![searchString isEqualToString:@""]))
    {
        if (searchCategory == 1)
        {
            predicate = [NSPredicate predicateWithFormat:
                                    @"firstName contains[cd] %@", searchString];
        }
        if (searchCategory == 2)
        {
            predicate = [NSPredicate predicateWithFormat:
                                     @"lastName contains[cd] %@", searchString];
        }
    }

    /*
     Method continues to perform the search and display the results...
     */
}
```

The Cancel Button

Clicking the cancel button behaves exactly as if the user selected all the text in the field and deleted it. The action message is sent to the target as usual. The target would typically see that the string in the field is the empty string, and so wouldn't filter anything.

Customizing Your Search Field's Appearance

You can specify the behavior in Interface Builder of search fields, like most Cocoa controls, without writing any source code. This is described in "Adding a Search Field to Your Application" (page 11). If you want to change a search field's appearance, for example, its buttons or field locations, this article describes how.

Customizing Buttons

In general, avoid changing the search button cell unless you want a custom appearance or behavior. Typically, the search button has two behaviors: it either displays a menu (if a menu template is set), or it invokes the search field's action method (when the button is clicked). The cancel button appears only if there is text in the search field.

You change the search button cell by sending `setSearchButtonCell:` to the button's cell, and change the cancel button cell by sending `setCancelButtonCell:` to the button's cell. To remove the search button or the cancel button, set the cell for that button to `nil`.

If you have customized either the search and cancel buttons, you can easily "reset" their attributes to the search field cell's default image, alternate image, target, and action. You do this for the search button by sending `resetSearchButtonCell` to the button's cell, and for the cancel button by sending `resetCancelButtonCell` to the button's cell. All other attributes of the search field cell are not changed.

Customizing Field Locations

The following NSSearchFieldCell methods return the bounds for the buttons and the text field. Override these methods to position the items differently. NSSearchFieldCell assumes that there is a search button, text field, and cancel button, from left to right.

```
- (NSRect) searchButtonRectForBounds:(NSRect)rect;
- (NSRect) cancelButtonRectForBounds:(NSRect)rect;
- (NSRect) searchTextRectForBounds:(NSRect)rect;
```

Document Revision History

This table describes the changes to *Search Fields*.

Date	Notes
2008-02-08	Clarified the behavior of the cancel button.
	In "Implementing the Target" (page 17), added a link to a description of the target-action mechanism.
2005-08-11	Updated links.
	In "Implementing the Target" (page 17), added a link to a description of the target-action mechanism.
2004-06-28	Clarified in "Implementing the Target" (page 17) that `setSendsWholeSearchString:` must be sent to the search field's cell.
2004-03-03	In "Configuring a Search Menu" (page 13), included a work around for specifying menu item tags for search field menus in Interface Builder and also clarified the use of tags in search categories.
2004-03-01	Clarified in "Adding a Search Field to Your Application" (page 11) that the messages are sent to the search field's cell.
2003-10-15	First version of *Search Fields*.