Token Field Programming Guide for Cocoa

Cocoa > User Experience

2007-12-11

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

Simultaneously published in the United States and Canada.

Contents

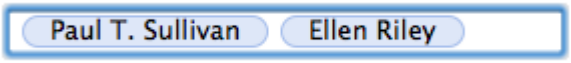
Figures, Tables, and Listings

Introduction to Token Field Programming Guide for Cocoa

This document discusses the role of token fields in a user interface, explains how they work, describes how to configure them, and shows how integrate them into your application.

Token Fields at a Glance

A token field is a text field with tokens as content. A token represents a string or other object.



- **Construction**: Single-cell control
- **Classes and inheritance**:
  - Control: `NSTokenField` : `NSTextField` : `NSControl` : `NSView`: `NSResponder` : `NSObject`
  - Cell: `NSTokenFieldCell` : `NSTextFieldCell` : `NSActionCell` : `NSCell` : `NSObject`
- **Design patterns**: Target-action, delegation, key-value observing
- **Object attributes**: Tokenizing character set, completion delay, token style
- **Important methods**: `objectValue` (NSControl), `tokenField:completionsForSubstring:indexOfToken:indexOfSelectedItem:` (delegate), `tokenField:representedObjectForEditingString:` (delegate), `tokenField:displayStringForRepresentedObject:`d (delegate), `tokenField:menuForRepresentedObject:` (delegate)
- **Important bindings**: value
- **Usage guidelines**: Under “Text Controls” in “Controls" (*Apple Human Interface Guidelines*)

Organization of This Document

This document consists of the following chapters:

- “About Token Fields” (page 9) describes what token fields are designed to do in a user interface and offers some guidelines for their usage.
- “How Token Fields Work” (page 13) discusses the central concepts and mechanisms of token fields.
- “Configuring Token Fields” (page 17) explains how to configure token fields and connect them to other objects in an application.

- "Displaying the Completion List" (page 19) describes how to display a list of possible completions for the currently entered substring.
- "Returning Represented Objects" (page 21) shows how to return an array of represented objects for the tokens in a token field.
- "Getting and Setting Token-Field Values" (page 23) shows how to extract and set the contents of a token field.
- "Implementing Menus for Tokens" (page 25) discusses how to associate menus with tokens in a token field.

About Token Fields

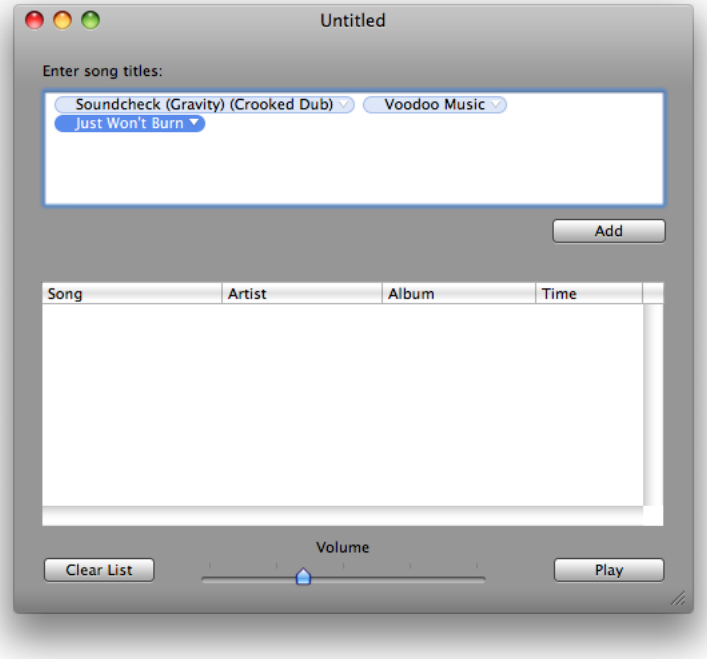
A token field is a text field with tokens as its content. A token represents a string or other object, A token's distinctive form makes it easy for users to recognize and manipulate it. Sometimes users select the string to be tokenized from a list of possible entries presented to them, such as the list of email addresses in one's Address Book application. The following sections describe the look and behavior of tokens in token fields and offer guidelines for the proper use of this control.

A Token Field in Operation

The basic purpose of a token field is to present entered strings as tokens when the user presses a tokenizing character, such as a comma. The tokens in the field are easier to recognize and manipulate than, say, a comma-separated list of strings. Manipulation of a token includes the ability to cut-and-paste them and to drag them between fields. Through delegation, `NSTokenField` extends this basic behavior.

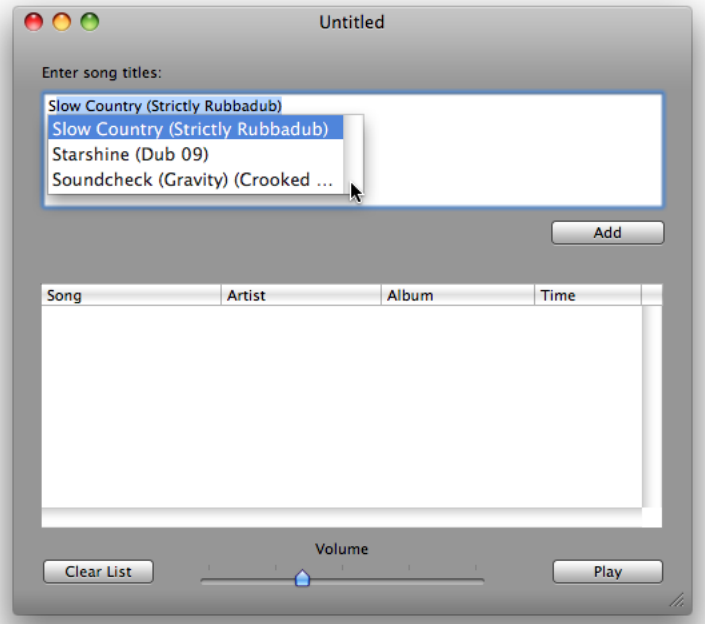
Before the user types in a token field, it looks exactly like a text field. The user types some text and then types a character from **tokenizing character set**; default, the tokenizing characters are a comma or the newline character (entered by pressing Return), which also may cause the action message to be sent. (Note that the newline character is always implied and is not actually specified in the character set.) Upon receiving a tokenizing character, the token field converts the entered string into a token (see Figure 1-1 for examples). Usually a token takes the form of a blue rounded rectangle with the string as title. But there is also a plain-text token style and possibly future styles.

Figure 1-1 Tokens in a token field



If an application implements the appropriate delegation method, after the entry of the first one or two (or more) characters, the token field displays a **completion list**. When the list appears is determined by a **completion delay**, a period that you can configure. Figure 1-2 shows what a typical completion list looks like.

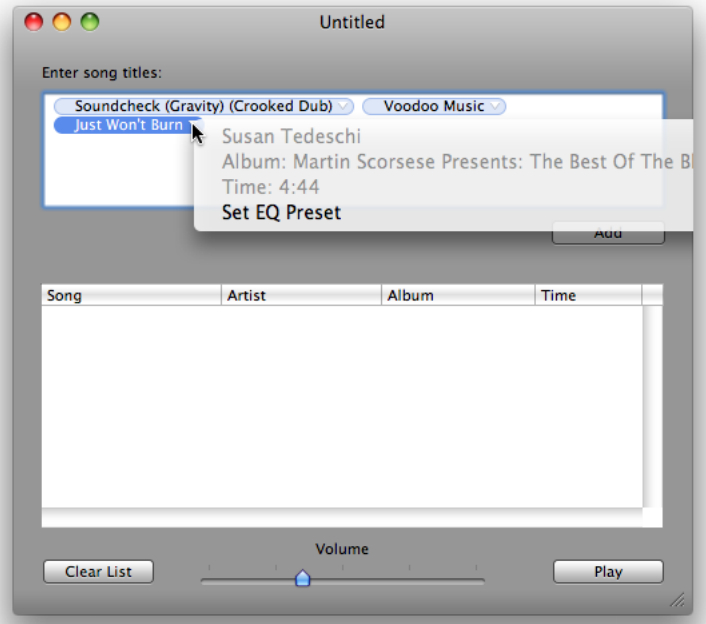
Figure 1-2 A token field's completion list



As users continue typing, the token field narrows the completion list to the matching strings. Users can either type the entire desired string, or they can use the mouse to select the desired string from the list by clicking it. After selection, users type a tokenizing character to convert the string into a token.

Through delegation, tokens may have **represented objects** associated with them; for example, a token with a title of "blue" could have an `NSColor` object associated with it. Tokens can also have menus attached to them, as illustrated in Figure 1-3. These menus can present additional information about the token and can present items that trigger actions on the object represented by the token.

Figure 1-3 A menu attached to a token



For additional guidance on using token fields, see the section on text controls in “Controls" in *Apple Human Interface Guidelines*.

When To Use Token Fields

You use token fields for several reasons. The primary reason is to make what the user enters in the field easy to recognize and convenient to move around, select, and otherwise manipulate.

But you may also to restrict what users enter in a token field to something from a finite list of possible entires. And you may want to associate an underlying represented objects with this string entries. These could be objects representing things such as email addresses, songs from an iTunes playlist, employee records, and so on. To have these features, you must implement the appropriate delegation methods.

To further extend the usefulness of a token, you can give it a menu whose items send messages to the represented object or return it upon request to a target object. When you copy-paste or drag the token between user-interface elements of the same or different applications—assuming you implement the required delegation methods—you are also moving the represented object.

You can also use token fields when all you are interested in is the string value of a token. For example, you might opt for the plain-text token style when you use a token field to enforce the correct spelling of a textual item. Note that there can be only one token per token field that is configured for the plain-text token style.

How Token Fields Work

A token field works on the premise of a finite collection of objects as potential content. These objects can be `NSString` objects or objects of any other type. Objects that are not strings require a display string.

Tokens and Represented Objects

In a sense a token is a labeled represented object even if that object is simply the string used for the label. A represented object is an object that is arbitrarily associated with a cell or a menu item. A token field—more precisely, the `NSTokenFieldCell` component of a token field—inherits the feature of represented objects from `NSCell`. But the implementation extends the notion of represented object to make it apply to all tokens in the field.

As an example, consider a token field in which users enter the names of people in their Address Book. The token field is implemented so that each token field has a represented object of type `ABPerson` (a class in the Address Book framework).

You are not required to assign a represented object to each token in the token field. In this case, the represented object of a token is the string it displays.

For further information on represented objects, see “Represented Objects" in *Control and Cell Programming Topics for Cocoa* and the section on controls and cells in “The Core Application Architecture" in *Cocoa Fundamentals Guide*.

The Object Value of Token Fields

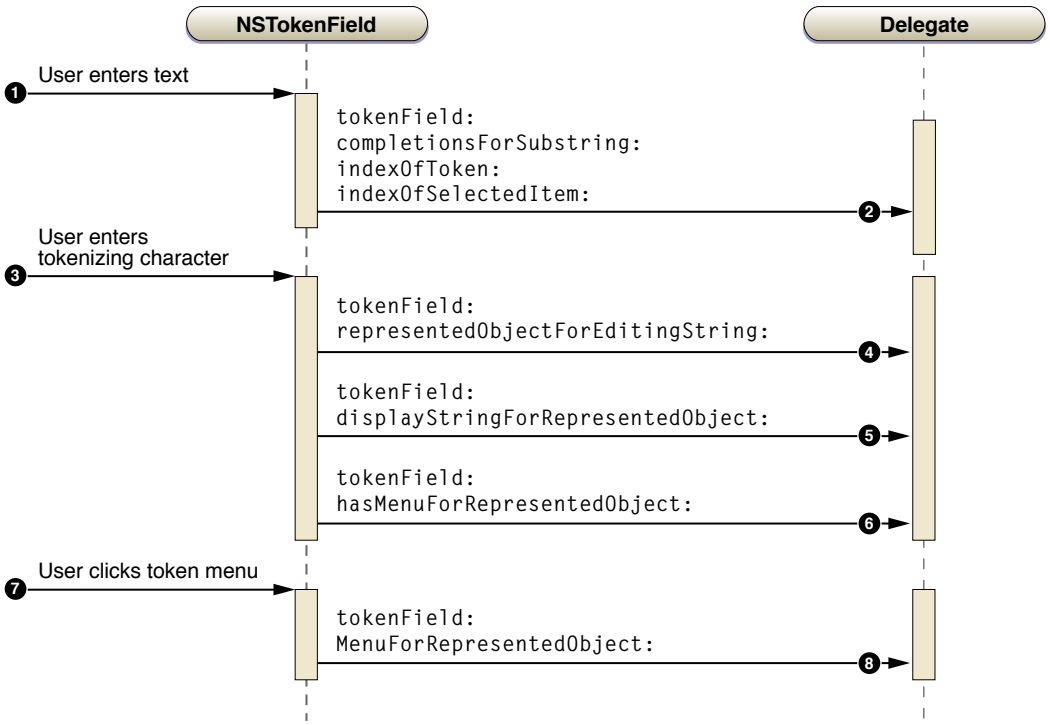
When you want to retrieve the contents of a token field, you send it an `objectValue` message. This message returns an array of the field’s represented objects, whether those objects are strings or something else. Conversely, you can set the contents of a token field by sending it a `setObjectValue:` message, passing in an array of the represented objects you wish the field to have. If these objects are not strings, the token field queries its delegate for the strings to display for the represented objects.

Because a token field is a direct descendent of `NSTextField`, it is a control that sends an action message to its target when the user presses the Return key (or if the insertion point leaves the field, if the action is configured as “Send on End Editing”). Pressing the Return key either tokenizes the most recently entered string or causes the action message to be sent. Your implementation of the action method is an ideal place to ask the token field (`sender`) for its object value.

Basic Interaction With the Delegate

To acquire the capabilities of completion lists, represented objects, and token menus for token fields, you must implement a number of delegation methods. A token field sends a series of messages to its delegate as illustrated in Figure 2-1.

Figure 2-1 Messages to the token field delegate



1. The user enters text in the token field.
2. The delegate receives the `tokenField:completionsForSubstring:indexOfToken:indexOfSelectedItem:` message and returns a list of possible completions for the passed-in substring.

   The delegate continues to receive this message as the user continues typing in the token field; each time it returns a progressively narrowed list of possible completions.
3. The user selects a string from the completion list and types the tokenizing character.

   The user could enter a string that is not in the list of possible completions and that is also tokenized.
4. The delegate receives the `tokenField:representedObjectForEditingString:` message and returns a represented object that corresponds to the passed-in editing string.

   If the delegate doesn't implement this message or returns `nil`, the entered string is the represented object.

5. If the delegate implements the `tokenField:representedObjectForEditingString:` method to return a represented object for a entered string, it next receives the `tokenField:displayStringForRepresentedObject:` message. The delegate implements this method to return a display string for the given represented object. (This display string may be different from the string entered from the completion list.)

6. The token queries the delegate with `tokenField:hasMenuForRepresentedObject:` to find out if the token has a menu. If there is a menu, it adds a triangular discovery button when it draws the token.

7. The user clicks a token's menu-discovery button.

8. The delegate receives the `tokenField:menuForRepresentedObject:` message and returns an `NSMenu` object (containing the desired menu items) to the token field, which displays the menu.

There are several other methods that a token field sends to its delegate, including `tokenField:styleForRepresentedObject:`, which allows the substitution of plain-text tokens for the encapsulating kind.

Configuring Token Fields

The following sections describe how to set the attributes of token fields , establish common bindings between token fields and controller objects, and to make delegate and target-action connections. You can accomplish most of these tasks in Interface Builder.

Token Field Attributes

You can set the token field attributes listed in Table 3-1 in Interface Builder.

Table 3-1 NSTokenField attributes in Interface Builder

Attribute	Description
Token Style	Choose one of Default, Plain, or Rounded. The default token style currently is Rounded—the blue rounded rectangle. Tokens in the plain style are simple text without any background; with the plain text style only one token is allowed per token field. You can change the token style on a case-by-case basis by implementing the delegation method `tokenField: styleForRepresentedObject:`.
Comp. Delay	Specify the completion delay for the token field: the period (in seconds) after the user begins typing before the token field displays the completion list. The default value is zero, which means "display the list immediately".

The tokenizing character set is an attribute not displayed by Interface Builder. When users enter one of the characters from the current tokenizing character set, it tells the token field to convert the preceding string to a token. The default tokenizing character is the comma; carriage return (or newline character) is not in the set, but is implied in all cases. You can change the tokenizing character set using `setTokenizingCharacterSet:`.

The Interface Builder inspector also displays attributes of the superclasses of `NSTokenField`, namely `NSTextField`, `NSControl`, and `NSView`. None of the settings for these attributes has a particular consequence for token fields.

Token Field Connections

For a token field to acquire the capabilities of completion lists, non-string represented objects, and token menus, a delegate must respond to the messages sent by the control. Be sure to connect the delegate outlet to an object in the application that implements the appropriate delegation methods. Also make a target-action connection between the token field and an action method implemented by a target object in the application. You can make these connections in Interface Builder or programmatically.

Displaying the Completion List

When the user begins typing in a token field, the control sends (after the specified completion delay) a `tokenField:completionsForSubstring:indexOfToken:indexOfSelectedItem:` message to its delegate. The delegate evaluates the passed-in substring for the current token and returns an array of strings that are the possible completions of the substring.

The code in Listing 4-1 is in an application that makes use of the Scripting Bridge technology (introduced in Mac OS X 10.5) to query the iTunes application for the tracks in the user's music library. The application stores these tracks (`iTunesTrack` objects) in an instance variable named `trackNames`. The delegate in this method gets the names of all tracks and then uses the `NSArray` method `filteredArrayUsingPredicate:` to narrow this array of track names to those whose initial characters match the passed-in substring.

Listing 4-1 Returning a completion list

```
- (NSArray *)tokenField:(NSTokenField *)tokenFieldArg completionsForSubstring:(NSString
 *)substring indexOfToken:(NSInteger)tokenIndex indexOfSelectedItem:(NSInteger
*)selectedIndex {

    NSArray *trackNames = [tracks valueForKey:@"name"];
    NSArray *matchingTracks = [trackNames filteredArrayUsingPredicate:
               [NSPredicate predicateWithFormat:@"SELF beginswith[cd] %@", substring]];
    return matchingTracks;
}
```

The `selectedIndex` parameter, which is not used in this example, allows the delegate to return a default selection in the completion list.

Returning Represented Objects

When the user enters a string and presses a tokenizing character, the token field sends the `tokenField:representedObjectForEditingString:` message to its delegate. This message asks the delegate to return a represented object for the entered token string (the `editingString` parameter in Listing 5-1). In this example, the delegate finds and returns the `iTunesTrack` object with the name matching `editingString`.

Listing 5-1 Returning represented objects for tokens

```
- (id)tokenField:(NSTokenField *)tokenField representedObjectForEditingString: (NSString
 *)editingString {
    iTunesTrack *track = [tracks objectWithName:editingString];
    if ([track exists])
        return track;
    return nil;
}
```

If the delegate returns `nil`, no represented objects are associated with the token string. Otherwise, the token field queries its delegate for the display string to use for each token by invoking the `tokenField:displayStringForRepresentedObject:` method. Listing 5-2 shows an implementation of this delegation method.

Listing 5-2 Returning the display string for a represented object

```
- (NSString *)tokenField:(NSTokenField *)tokenFieldArg
displayStringForRepresentedObject:(id)representedObject {     return [representedObject
 name]; }
```

Getting and Setting Token-Field Values

To retrieve the objects represented by the tokens in a token field, send the token field an `objectValue` message. Although this method is declared by `NSControl`, `NSTokenField` implements it to return an array of represented objects. If the token field simply contains a series of strings, `objectValue` returns an array of strings. To set the represented objects of a token field, use the `setObjectValue:` method, passing in an array of represented objects. If these objects aren't strings, `NSTokenField` then queries its delegate for the display strings to use for each token.

A common place to call `objectValue` is in an action method. Listing 6-1 gives an example of such a method.

Listing 6-1 Getting and setting the contents of a token filed

```
- (IBAction)addToPlaylist:(id)sender { // sender is token field
    // add songs to playlist, select first one added
    NSIndexSet *curSongIndex = [NSIndexSet indexSetWithIndex:(NSUInteger)[currentList
count]];
    [currentList addObjectsFromArray:[sender objectValue]];
    [songTable reloadData];
    [songTable selectRowIndexes:curSongIndex byExtendingSelection:NO];
    [sender setObjectValue:nil];

}
```

Note that this method clears the token field by setting its object value to `nil`.

Implementing Menus for Tokens

If you want tokens in a token field to have menus, you must implement the `tokenField:hasMenuForRepresentedObject:` and `tokenField:menuForRepresentedObject:` delegation methods. A token field invokes the former method just before it displays a token to find out if it should draw a discovery triangle. It invokes the latter method when the user clicks the triangle.

Listing 7-1 gives a sample implementation of these methods. Note that it sets the token's represented object as the represented object of the menu item that invokes an action method. The target of the action method fetches the represented object from the menu item to act upon it.

Listing 7-1 Implementing the menu delegation methods

```
- (BOOL)tokenField:(NSTokenField *)tokenField
hasMenuForRepresentedObject:(id)representedObject {
    return YES;
}

- (NSMenu *)tokenField:(NSTokenField *)tokenField
menuForRepresentedObject:(id)representedObject {

    NSMenu *tokenMenu = [[[NSMenu alloc] init] autorelease];

    if (![representedObject exists])
        return nil;

    NSMenuItem *artistItem = [[[NSMenuItem alloc] init] autorelease];
    [artistItem setTitle:[representedObject artist]];
    [tokenMenu addItem:artistItem];

    NSMenuItem *albumItem = [[[NSMenuItem alloc] init] autorelease];
    [albumItem setTitle:[NSString stringWithFormat:@"Album: %@", [representedObject
album]]];
    [tokenMenu addItem:albumItem];

    NSMenuItem *durationItem = [[[NSMenuItem alloc] init] autorelease];
    [durationItem setTitle:[NSString stringWithFormat:@"Time: %@", [representedObject
time]]];
    [tokenMenu addItem:durationItem];

    NSMenuItem *mItem = [[[NSMenuItem alloc] initWithTitle:@"Show Album Art"
action:@selector(showAlbumArt:) keyEquivalent:@""] autorelease];
    [mItem setTarget:self];
    [mItem setRepresentedObject:representedObject];
    [tokenMenu addItem:mItem];

    return tokenMenu;
}
```

Document Revision History

This table describes the changes to *Token Field Programming Guide for Cocoa*.

Date	Notes
2007-12-11	New document that describes how to set up and programatically manage a token field.

Document Revision History