Combo Box Programming Topics

User Experience: Controls

2002-11-12

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

Simultaneously published in the United States and Canada.

Contents

Introduction to Combo Boxes

A Combo box is a control that gives the user two ways to enter a value: entering it directly in a text field, or choosing it from a pop-up list of pre-selected values.

Developers who want to incorporate a combo box into their user interface should read this document.

Organization of This Document

This topic describes how to use a combo box; "How Combo Boxes Work" (page 7) gives basic information on combo boxes. "Providing Data for a Combo Box" (page 9) describes how to provide data for the combo box's pop-up list. "Setting the Combo Box's Value" (page 13) describes how to set and retrieve the combo box's value. "Managing the Combo Box's List" (page 11) describes how to use the combo box's pop-up list. "Using Automatic Completion in Combo Boxes" (page 15) describes how the combo box can try to complete what the user enters into the text field.

Because NSComboBox is a subclass of NSTextField, see *Text Fields* for more information.

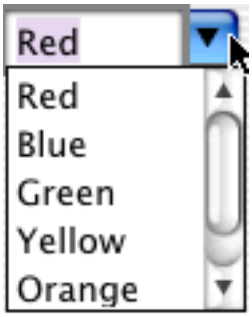
How Combo Boxes Work

A combo box is a control that gives the user two ways to enter a value: entering it directly in a text field, or choosing it from a pop-up list of pre-selected values. Use this control whenever you want the user to enter information that can be selected from a finite list of options. Note that while you can construct your NSComboBox so that users are restricted to only selecting items from the combo box's pop-up list, this isn't the combo box's normal behavior.

While the pop-up list is visible, typing into the text field causes an incremental search to be performed on the list. If there's a match, the selection in the pop-up list changes to reflect the match.

The NSComboBox normally looks like this:

When you click the downward-pointing arrow at the right-hand side of the text field the pop-up list appears, like this:



If there isn't sufficient room for the pop-up list to be displayed below the text field, it's instead displayed above the text field. Selecting an item from the list, clicking anywhere outside the control, or activating another window dismisses the pop-up list.

Note that NSComboBox is a also a subclass of NSTextField, and thus inherits all of NSTextField's methods. NSComboBox relies heavily upon its cell class, NSComboBoxCell. NSComboBoxCell is a NSTextFieldCell subclass, which combines a text field cell with a button cell.

A combo box is implemented by two classes: NSComboBoxCell, the cell which does most of the work, and NSComboBox, the control that contains that cell. Every method in NSComboBoxCell has a cover in NSComboBox. (A cover is a method of the same name that calls the original method.)

Providing Data for a Combo Box

The NSComboBox control can be set up to populate the pop-up list either from an internal item list or from an object that you provide, called its data source. Specify which to use with `setUsesDataSource:`. By default, a combo box uses the internal list.

If you specify that a combo box uses an external data source and then try to invoke a method that uses the internal list—such as `addItemWithObjectValue:`—the method throws an exception.

Working With an External Data Source

An external data source declares the methods that the combo box uses to access its data. Use one if an internal list isn't efficient for your data. An external data source can store its items in any way, but it must be able to identify them by an integer index.

To specify that combo box uses an external data source, first use `setUsesDataSource:` with `YES` as the argument, then use `setDataSource:` with your data source object as the argument. If you use `setDataSource:` before `setUsesDataSource:`, `setDataSource:` throws an exception.

The data source must define these methods. The method `setDataSource:` logs a warning if its argument doesn't implement them.

- `numberOfItemsInComboBox:` returns how many items to display.
- `comboBox:objectValueForItemAtIndex:` returns the object that corresponds to the specified index.

The data source can optionally define these methods. The method `setDataSource` doesn't check for them and the combo box invokes them only if they're available.

- `comboBox:indexOfItemWithStringValue:` returns the index for the item that matches the specified string. If this method is available, the combo box performs incremental searches when the user types into the text field with the pop-up list displayed.
- `comboBox:completedString:` returns a string that begins with the specified string. If autocompletion is enabled, the combo box tries to complete what the user enters into the text field with an item from the pop-up list. If this method isn't available and autocompletion is enabled, the combo box goes through each item one-by-one to find a completion.

And here are some NSComboBox methods your data source may need if it loads data in the background:

- `noteNumberOfItemsChanged` informs the combo box that the number of items in the data source has changed.
- `reloadData` marks the combo box as needing redisplay, so it reloads the data for the visible pop-up items and draws the new values.

The combo box treats objects provided by its data source as values to be displayed in the combo box's pop-up list. If these objects aren't of common value classes—such as strings, numbers, and so on—you'll need to create a custom NSFormatter to display them. See *Data Formatting Programming Guide for Cocoa* for more information.

Working with an Internal List

NSComboBox provides a complete set of methods that allow you to add, insert, and delete items in the internal item list for combo boxes that don't use a data source:

- To add one or more items to the end of the list, use `addItemWithObjectValue:` or `addItemsWithObjectValues:`
- To insert an item into the middle of the list, use `insertItemWithObjectValue:atIndex:`
- To find the index for a particular object, use `indexOfItemWithObjectValue:`.
- To find the object at a particular index, use `itemObjectValueAtIndex:`.
- To remove items from the list, use `removeAllItems`, `removeItemAtIndex:`, or `removeItemWithObjectValue:`.
- To retrieve an array of all the list's items, use `objectValues`.
- To retrieve the number of items in the list, use `numberOfItems`.

If `usesDataSource` returns `YES` and you use any of the above methods, the method will throw an exception. By default, `usesDataSource` returns `NO`.

Managing the Combo Box's List

There are a number of ways that you can affect a combo box's list's appearance, as well as controlling them programatically.

Setting the List's Appearance

These methods let you control the list's appearance.

- To choose whether the pop-up list has a vertical scroll bar, use `setHasVerticalScroller:`. When there's no scroll bar, the user can still scroll to items that aren't displayed by holding the mouse at the top or bottom of the list. And when there is a scroll bar, the scroll bar is shown even when all items can be displayed in the visible portion of the list.
- To set the number of items displayed in the pop-up list, use `setNumberOfVisibleItems:`. The default is 5.
- To set the amount of space that surrounds each item in the list, use `setIntercellSpacing:` with an argument of type NSSize. The width component is the size in points of the list's left and right margins. The height component is the space in points above and below each list item.
- To set the height of each list item, use `setItemHeight:`.

Manipulating the List's Selection

These methods let you manipulate the list's selection:

- To select a particular item, use `selectItemAtIndex:` or `selectItemWithObjectValue:`.
- To retrieve the selected item, use `indexOfSelectedItem` or `objectValueOfSelectedItem`.
- To deselect an item, use `deselectItemAtIndex:`.

Note that changing the list's selection does not change the content's of the combo box's text field. For more information, see "Setting the Combo Box's Value" (page 13).

Scrolling the List

These methods let you scroll the list. The list doesn't need to be visible to use these methods:

- To scroll the list so that a particular item is as close to the top as possible, use `scrollItemAtIndexToTop`.

- To scroll the list so that a particular item is visible, use `scrollItemAtIndexToVisible`.

Setting the Combo Box's Value

When you set or retrieve a combo box's value with the standard NSControl methods (such as `setStringValue:`, `stringValue`, `setFloatValue:`, and `floatValue`), you're setting or retrieving the value of the combo box's text field, and not the current selection of the list. Programmatically changing the combo box's value does not change what's selected in the combo box's list. Conversely, programmatically changing what's selected in the list does not change the text field's value. If you want the text field value and the list selection to match up, you need to set them individually.

For example, say you want to initialize the combo box's list and text field to the list's third item. This code does that for a combo box that maintains an internal item list:

```
[myComboBox selectItemAtIndex:2]; // First item is at index 0
[myComboBox setObjectValue:[myComboBox objectValueOfSelectedItem]];
```

This code does that for a combo box with an external data source:

```
[myComboBox selectItemAtIndex:2];
[myComboBox setObjectValue:
    [myComboBoxDataSource comboBox:myComboBox
        ObjectValueForItemAtIndex:[myComboBox indexofSelectedItem]]];
```

And this code initializes the combo box's text field to "Red" and selects it from the list if available. Note that this works for combo boxes with either internal or external data sources:

```
[myComboBox setStringValue:@"Red"];
[myComboBox selectItemWithObjectValue:@"Red"];
```

Using Automatic Completion in Combo Boxes

A combo box can perform automatic completion, trying to complete what the user enters into the text field with an item from the pop-up list. If it does, every time the user enters characters at the end of the text field, the combo box calls the NSComboBoxCell method `completedString:`. If `completedString:` returns a string that's longer than the existing string, the combo box replaces the existing string with the returned string, and selects the additional characters. If the user adds characters somewhere besides the end of the string or deletes characters, the combo box does not try to complete it.

The default implementation of `completedString:` first checks whether the combo box uses a data source and whether the data source responds to `comboBox:completedString:` or `comboBoxCell:completedString:`. If so, the combo box cell returns that method's return value. Otherwise, this method goes through the combo box's items one-by-one and returns the first item which starts with the string that the user entered. This comparison is case-sensitive.

To read and set whether a combo box performs completion, use `completes` and `setCompletes:`. By default, it does not.

Document Revision History

This table describes the changes to *Combo Box Programming Topics*.

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.