Chapter 5

KEYBOARD USER INTERFACE

CHAPTER 5 KEYBOARD USER INTERFACE

Goal

To understand the available keyboard user interface features of an application and how to customize its behavior.

Prerequisites

Familiarity with end-user keyboard navigation of graphical interfaces.

Objectives

At the end of this section, you will be able to:

» Control how the user navigates your graphical interface with the keyboard

» Assign mnemonics, short cuts and default buttons

Reading

NSView class reference in the Application kit.

Keyboard user interface and navigation

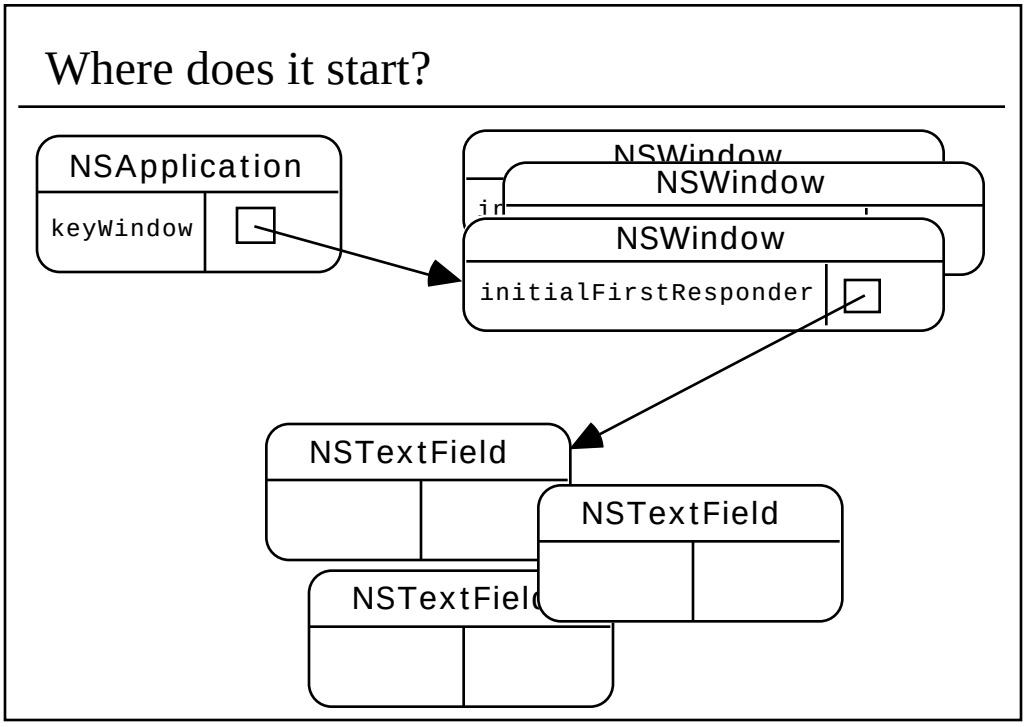
KEY	EFFECT
Tab, Shift-Tab	Navigate between views
Arrows	Navigate within a view
Space	Select and activate control
Return	Press default button
Escape	Dismiss panel
Mnemonics	Navigate between views, activate control

Keyboard user interface and navigation

Because many applications are mainly text based a user spends most their time with their hands on the keyboard. Applications can be most productive if they are easily and quickly driven from the keyboard almost exclusively—minimizing mouse activity. There are a number of keyboard conventions for navigating, selecting, activating and cancelling.

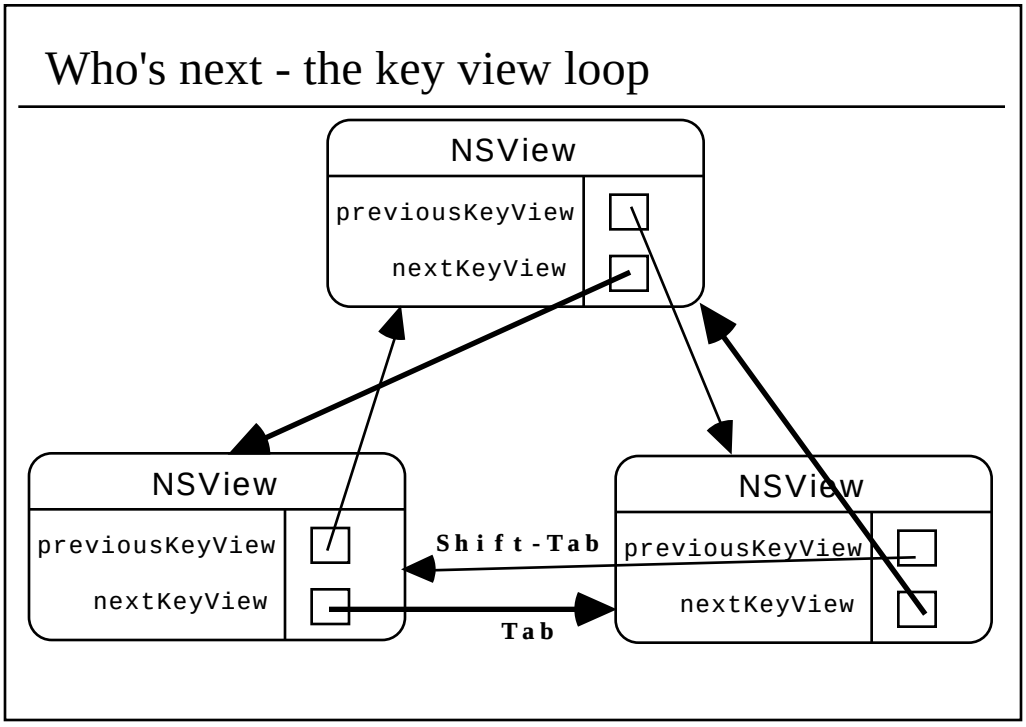
The participating Application Kit classes already implement this behavior. But as the user interface designer for your application, you have some creative and technical choices to make.

You need to pick the mnemonics and short cuts for your specific control labels. You have to decide which default button makes sense for each dialog. More challenging, you have to determine a logical order for visiting the controls on your window which may incorporate a complex form. You may even create a new custom control and make sure it plays by the rules. How does it all work?



Where does it start?

When your application presents a window for the first time, some user interface object is in focus, ready to act on a keystroke. This is the **initialFirstResponder** of the window. You can connect this to an object of your choice in Interface Builder. The default selection is based on our literary penchant for the upper left. The first object spacially located there that is capable of being first responder is selected.

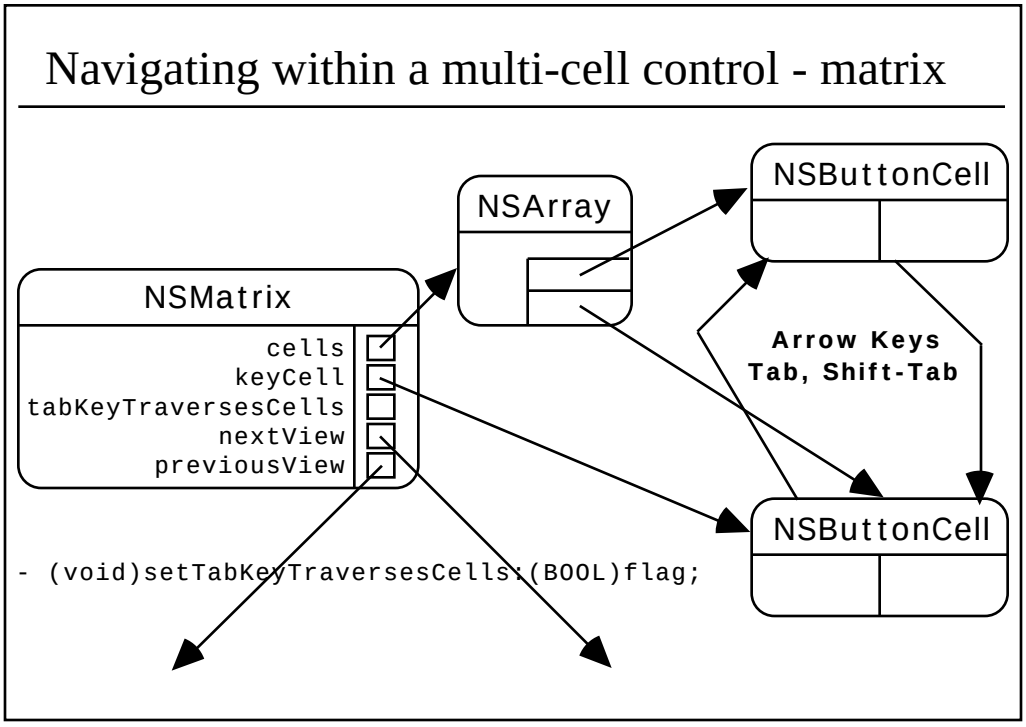


Who's next - the key view loop

All NSViews have the means for being linked together into a doubly-linked list with **nextKeyView** and previousKeyView pointers. The list usually wraps back to the beginning to form a loop. Known as the key view loop, it defines the order in which views are visited by successive Tab and Shift-Tab keystrokes.

By default, all the views in your window are hooked together into a key view loop, again, using traditional spacial rules of left to right, top to bottom. But complex user interfaces often group controls together or modify these assumptions for aesthetic or technical reasons. In these cases, you are likely to want a custom key view loop. Some user interfaces group controls into boxes or forms that work best multiple disjoint loops. In all cases, key view loops can be connected and disconnected graphically using Interface Builder.

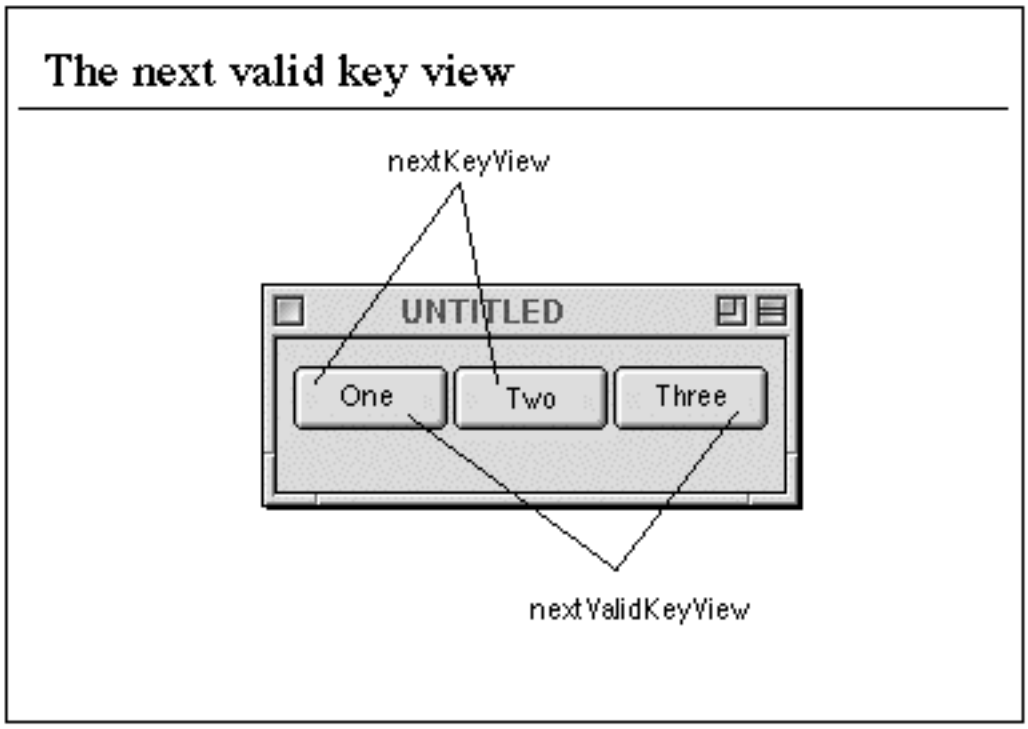
The default key view loop is built by a window only if **initialFirstResponder** is nil. Once you connect **initialFirstResponder** in Interface Builder, or programmatically at run time it is assumed that you have already configured a custom key view loop. You have to choose between the default or setting up the entire loop manually.



Navigating within a multi-cell control—matrix

Multi-cell controls like NSMatrix are like nested user interfaces with the need for an internal policy of keyboard navigation. Arrow keys are used to move up and down, right or left and this applies to a matrix of radio or check buttons, menus, NSPopupButton and so on. Depending on your layout, it may or may not make sense to have Tab and Shift-Tab visit the cells within the matrix or jump right to the next real view. This is configurable.

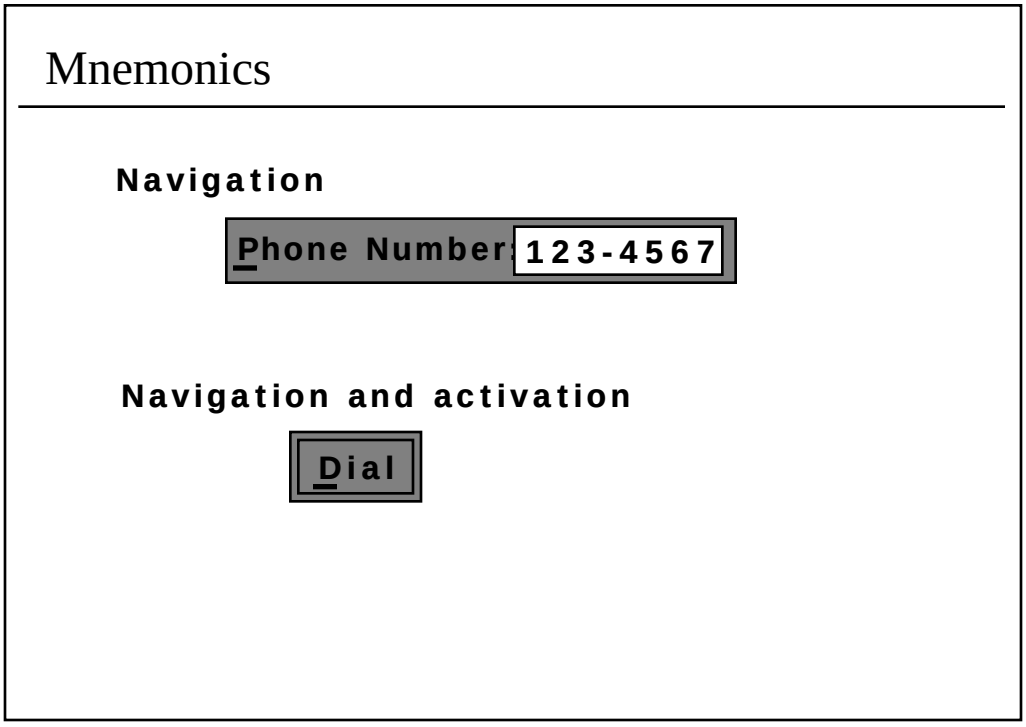
Note that NSMatrix still has previous and next key view outlets that point to the real NSViews on either side. It also has an outlet to point to the key cell. It is up to the NSMatrix to direct the keystrokes through or over its cells.



The next valid key view

To be a key view, a view instance must be able and willing. Not all views will accept key view status, depending on their current state or in some cases never. A disabled button should not be visited. A button can be configured so that it will not be visited at all, regardless.

All views are nonetheless linked together by the next and previous key view outlets. At any point in time, a view can also determine its next valid key view, and they may not be the same.

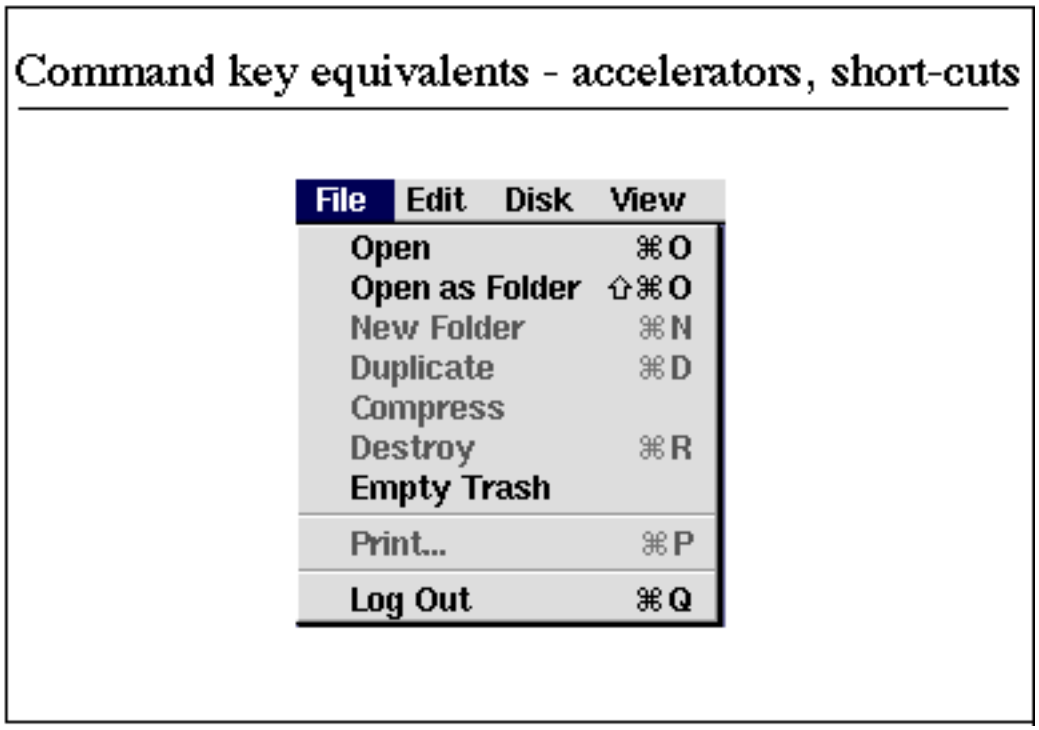


Mnemonics

Mnemonics are single letters that can be used as keyboard navigation aids. They are used to move the focus to a new object and, in the case of controls like buttons, will press or activate the button as well. A title can have a mnemonic and, not being a valid key view, when navigated to will pass control to the next valid view, the text field just next to it for instance. While editing a text field, mnemonics are inactive so that keystrokes will be accepted as typed text input, not navigation commands.

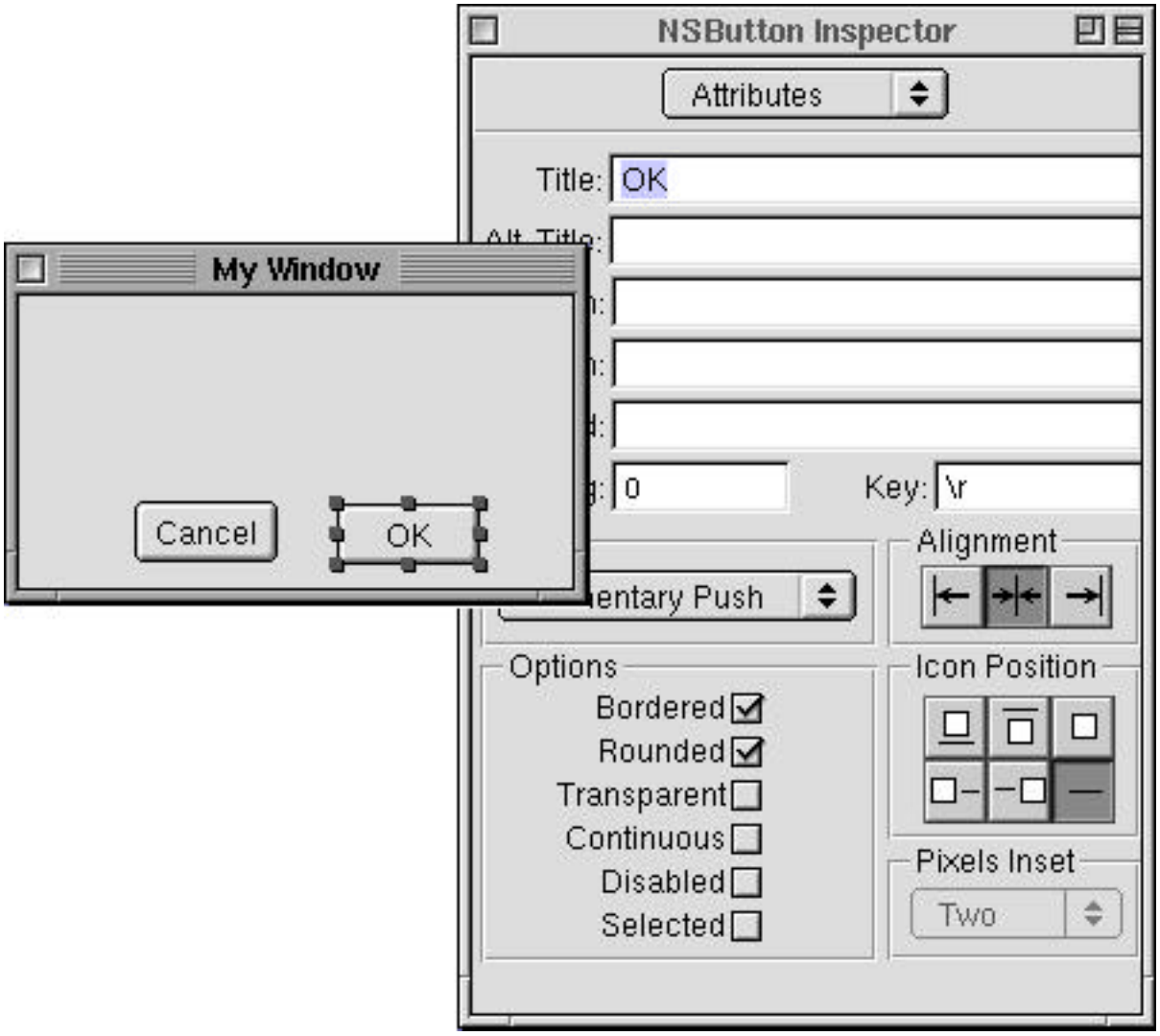
You can assign a mnemonic in Interface Builder by pointing at the letter (not necessarily the first in a word or phrase) and double-clicking the mouse while holding down the Alternate key. Note, you cannot be in the middle of editing the text itself. The control should be selected but not the text that makes up the title or label.

Mnemonics apply to menus and menu items as well but may require a modifier key, such as the Alternate key.



Command-key equivalents—accelerators, short-cuts

These navigation short cuts apply to menu items and buttons and frequently conform to an industry-wide convention. Because they use modifier keys, they are available at all times, even during text data entry into a text field or view.



Providing a default button

Every window has a **defaultButtonCell** outlet. This points to a button that will be activated by any Return or Enter keystroke while the window is key. This applies even while typing in a text field but not when another button is in focus—the first responder.

You can assign the default button cell by entering “\r” in the key equivalent field of the Interface Builder inspector.

Dismissing modal panels with Escape

Cancelling or dismissing an dialog is usually done by convention with the Escape key. It is generally applied to modal panels and has the effect of stopping the modal loop with the **abortModal** message. This is the case for the Application Kit stock modal panels such as NSAlertPanel.

‘\E’ can be set as the keyboard equivalent for any single button on a panel so that Escape works consistently with your custom modal panels as well.

Important ideas from this section

- There are a number of keyboard user interface and navigation conventions that your application should follow.
- The capabilities are generally built into Application Kit classes but you must assign the mnemonics, accelerators, the default buttons and connect your controls into a coherent key view loop.
- All connections and assignments can be performed with Interface Builder and stored in the nib file.