3 Setting an Object's Attributes

Examining an object's attributes

Customizing windows and panels

Setting button attributes

Associating images or sounds with buttons

Managing images and sounds

Customizing titles, text fields, and scroll views

Setting textual attributes

Setting text and background colors

Setting mnemonics

Setting box (group) attributes

Customizing browsers

Setting attributes of menu items and pop-up lists

Setting matrix attributes

Setting up table views

Automatically resizing objects

Using tags

But four young Oysters hurried up,
    All eager for the treat;
Their coats were brushed, their faces washed,
    Their shoes were clean and neat —
And this was odd, because, you know,
    They hadn't any feet.

Lewis Carroll, *Through the Looking Glass*

Is it a world to hide virtues in?

Shakespeare, *Twelfth Night*

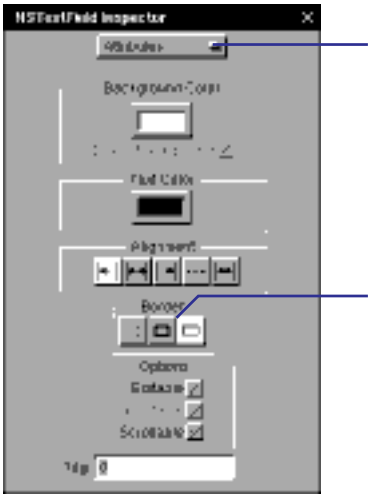
Examining an object's attributes

1 **Choose Inspector from the Tools menu.**

2 **Select an object in the interface.**

You can examine the attributes of any object, whether that object is a graphical object such as a button or panel, or a non-UI object in the Instances display.

The Inspector panel displays the attributes of the currently selected object.

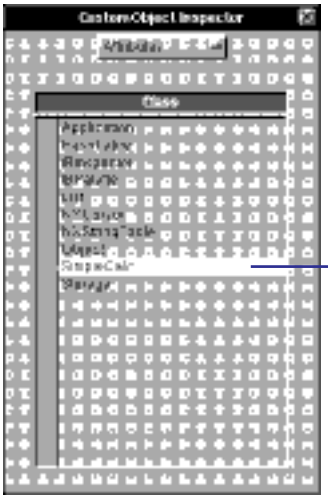


Tip: You can also bring up the Attributes display of the Inspector panel by pressing Command-1.

Once the Inspector panel is visible on the screen, it stays there until you close it. As you select different objects, their attributes are displayed (or dimensions or connections or help links—whatever Inspector display is current).

You can also select objects in the Instances display and examine their attributes. Some of these objects (like First Responder) have no attributes. Others, like an instance of a custom class, have only one attribute.

Select an object. In this case the object is an instance of a custom class.



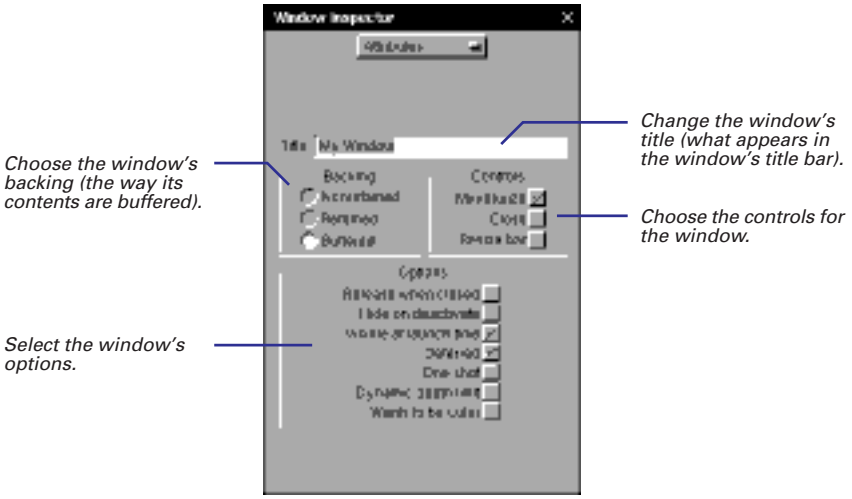
For custom instances, the Attributes display is the same as the Custom Class display.

The class of the selected custom instance is displayed. If you want to, you can change its class in this display by selecting another class name.

Customizing windows and panels

- **Set the window title.**
- **Determine how the Window Server buffers window contents.**
- **Choose the window's controls.**
- **Set the window's options.**

A single Attributes display serves for both windows and panels.

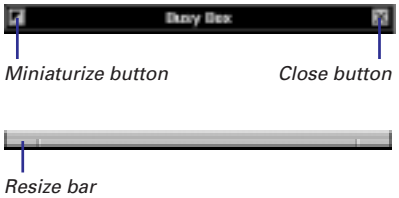


Window Backing

The backing determines how to redraw part of a window when that part is re-exposed after being covered by another window.

- **Nonretained**: The application is responsible for all drawing on the screen because there is no buffer. If the application does nothing, the re-exposed part is replaced by the background color. Nonretained windows are appropriate for transitory images that you don't need to save.
- **Retained**: The Window Server copies the covered part's pixels to a buffer. When the obscured part of the window is later revealed, the Window Server redraws only that part, not the rest of the window. A retained window is the appropriate choice for most situations.
- **Buffered**: The Window Server first draws in the buffer and then copies the buffer to the screen. When an obscured part is revealed, the Window Server refreshes the entire window using the buffer. A buffered window is appropriate when you don't want users watching complicated images being rendered on-screen. It is also the best choice for animation or for redrawing lines of rapidly typed text.

Window Controls



Window Options

Option	Description
Release when closed	The window object is sent a **release** message when it is closed.
Hide on deactivate	The window should disappear when the application is deactivated.
VIsible at launch time	The window should appear when the nib file is loaded.
Deferred	A window device for this object is deferred until it's placed on-screen.
One shot	The window device is released when it is removed from the screen.
Dynamic depth limit	The window's depth limit can change to match the depth of the screen.
Wants to be color	The window is displayed on a color screen (2-monitor systems only).

What's the Difference Between a Window and a Panel?

A panel is a window that serves an auxiliary function within an application. Because it's intended for a supporting role, a panel typically has these features:

- A panel can be the key window, but never the main window.
- When the application is deactivated, the panel moves off-screen (it's removed from the screen list). When the application is reactivated, the panel appears again.
- When a panel is closed, it moves off-screen; it isn't destroyed.
- When instantiated programmatically, panels have a grey background by default, while programmatically created windows have a white background.

Also, a panel usually has fewer controls: only a close button; rarely a resize bar; and sometimes no controls at all.

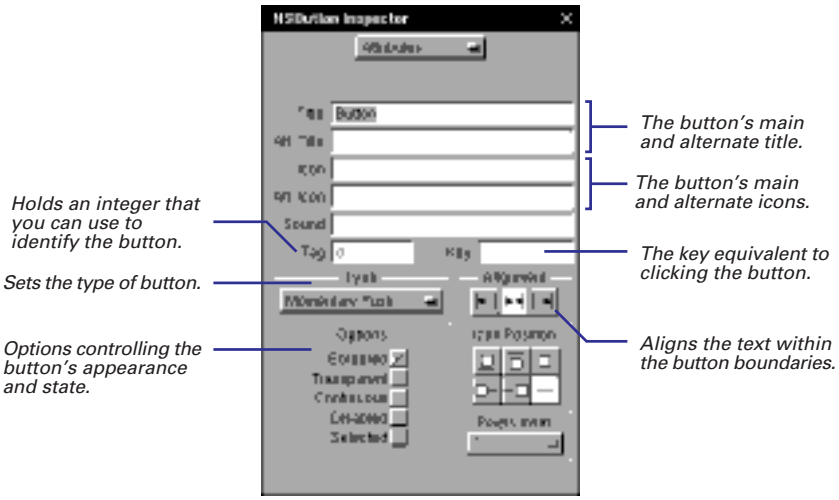
You can make some panels exhibit special behavior for specialized roles:

- A panel can be precluded from becoming the key window until the user makes a selection in it.
- Some panels (e.g., palettes) can float above windows and other panels.
- You can have a panel receive mouse and keyboard events while an attention panel is on-screen. Actions within the panel can thus affect the attention panel.

Setting button attributes

- **Enter the button's main and alternate titles.**
- **Select the button type.**
- **Specify any key equivalent.**
- **Specify button options.**

The Attributes display for buttons enables you to set a button's type, title, icon, alternate title and icon, and various other characteristics. The object labeled Button on the Views palette is only one style of button (albeit the most common style). The palette also holds radio buttons and switch buttons. Using the Attributes display for buttons, you can customize any palette button, making it something that is uniquely suitable for a particular circumstance.



The Icon Position and Pixels Inset controls as well as the Sound and Icon fields are described in detail in the next task, "Associating images or sounds with buttons."

For more information on the Tag field, see "Using tags" in this chapter.

You might think of storing specially configured buttons on a dynamic palette. See Chapter 5, "Using Dynamic Palettes," for complete information.

The Anatomy of a Button

A button is essentially a two-state NSControl object. When a user clicks a button, an action message is sent to a target object. It is two-state because it is either on or off, and when it is on, it typically sends its action message. For a button, the states are also known as "normal" (off) and "alternate" (on).

Like most objects on the Views palette in Interface Builder, a button is actually a compound object: an NSButton object and an NSButtonCell object. (See "Compound Objects" in this chapter.) Most of NSButton's methods match identically declared methods in NSButtonCell. Aside from dispatching the action message, NSButton's unique role is to set the font of the key equivalent, and to manage the highlighting or depiction of the NSButton's current state.

Button Titles and Icons

The Title field's value is what appears in most buttons; you can set the title by double-clicking inside the button. The Icon field identifies an image stored in the nib file (Images display) that appears within the button. The alternate title (Alt. Title) and the alternate icon (Alt. Icon) appear when the user clicks a button of type Momentary Change or Toggle.

Button Key

The Key field identifies a keyboard alternative to clicking the button. Possible values are: \e (Escape), \r (Return), and any normal letter or number.

Button Type

Type	Button Behavior When Clicked
Momentary Push	Button is highlighted, appears to be pressed.
Momentary Change	Alternate button title and icon appear (while mouse button is pressed).
Momentary Light	Button is highlighted, but no illusion of being pressed.
Push On/Push Off	First click highlights button with illusion of being pushed in; second click returns it to normal.
On/Off	First click highlights button. Second click returns it to normal.
Toggle	First click displays alternate title and button. Second click returns to normal.

Button Options

Option	Description
Bordered	A line is drawn around the button's border.
Transparent	The button has no border, text, icon, or background color.
Continuous	The button sends its action message continuously when pressed.
Disabled	Prevents activation of the button; title is in gray.
Selected	The button, when initialized, is to be selected (applies to switch and radio buttons).

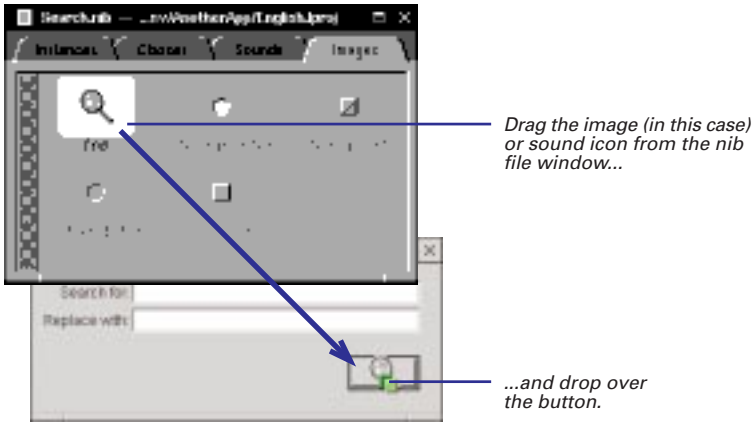
Associating images or sounds with buttons

- **Drag the icon representing an image or sound from the nib file window or the Workspace and drop it over a button.**

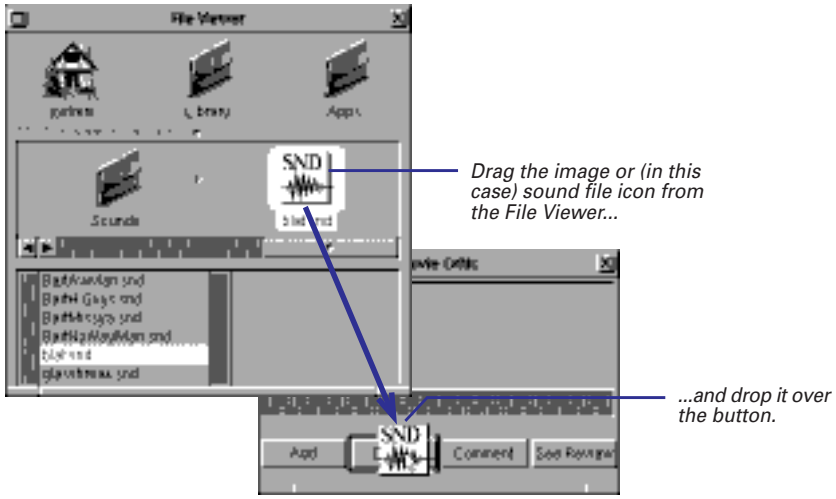
  Or

- **Enter the file name of the image or sound in the appropriate field of the button's Attributes inspector.**

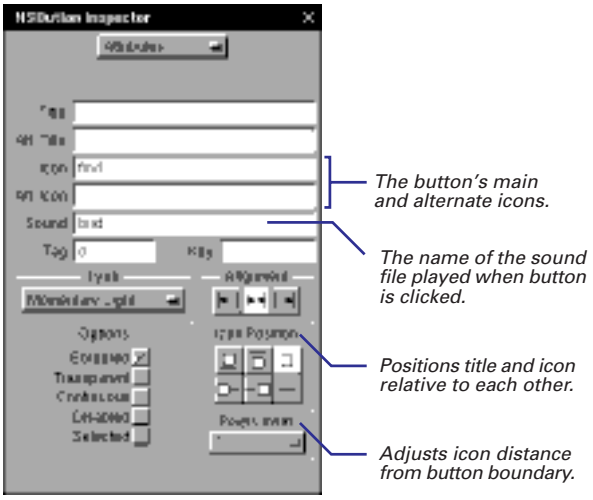
When you click a button that has a sound associated with it, it plays the sound. Images appear in buttons with or without text.



When you drag an image or sound from the File Viewer, it automatically gets added to the Images or Sounds section of the nib file.



Several fields and controls in the Inspector's Attributes display for buttons relate to images and sounds.



Before you type in the file name, you should insert the resource into the nib file or the project. Usually, you want to add the resource to the project. See "Managing images and sounds" for more information.

Note that the name of an image or sound in this display is the file name (**find.tiff** and **Poit.snd**, for example) minus the extension. Instead of dragging and dropping image and sound icons, you can type their file names (minus the extension) in the appropriate field.

Icon Position and Pixels Inset

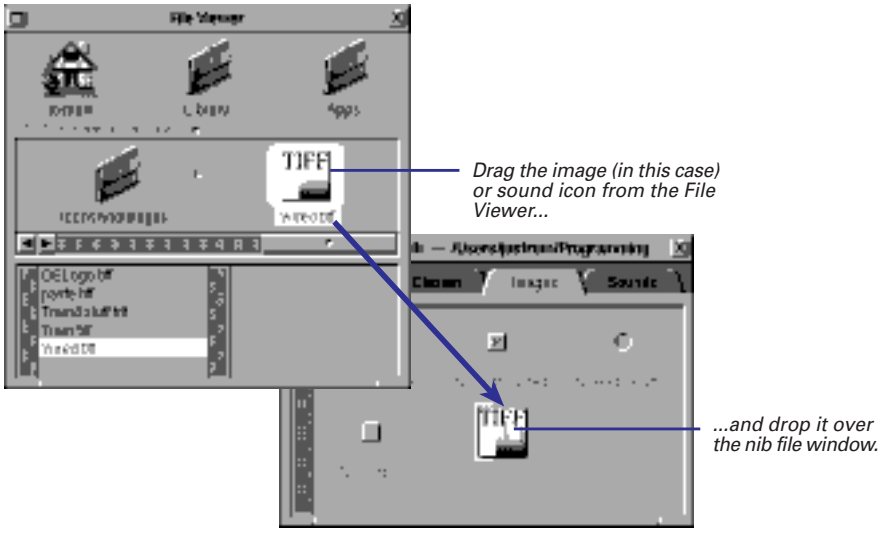
The six buttons in the Icon Position group position the button title and icon relative to each other. Thus, you can have the title above, below, to the left, or to the right of the icon, or show only one or the other. The Pixels Inset pop-up list gives several pixel distances for adjusting the spacing between the icon and the nearest edge of the button.

Tip: If you want to import images into your interface for decorative purposes, use the image view object on the DataViews palette. You simply drop the image on the image view. You'll probably want to deselect the bordered option and the Editable option in the image view's Attributes inspector.

Managing images and sounds

- **To add an image or sound, drag its file icon from the File Viewer and drop it over the nib file window.**
- **Examine the image or sound in the Inspector's Attributes display.**

You can add images and sounds to a nib file. The image or sound is added to the appropriate display no matter what display is currently showing.



As shown in "Associating images or sounds with buttons," you can add images and sounds to a nib file as a side effect of associating them with a button.

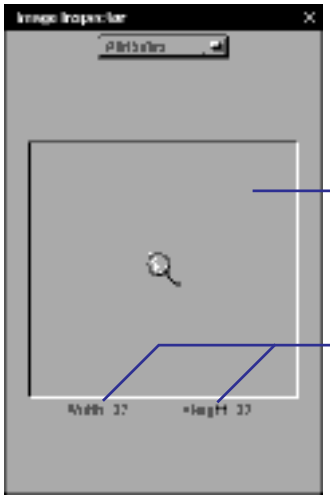
Although the association of images and sounds with buttons is an important reason for putting them into a nib file, there are other reasons. When you composite an image or play a sound in your code, the search path (if your code supplies no path) starts with the application's executable (already loaded resources), the main bundle, and the main bundle's **.lproj** directories. Then the standard directories are searched:

- the appropriate subdirectory of the user's **~/Library** directory
- the appropriate directory in **/LocalLibrary**
- the appropriate directory in **/NextLibrary**

If you do not want to risk an image or sound not being in one of these standard directories, then you should store it in a nib file or in the project.

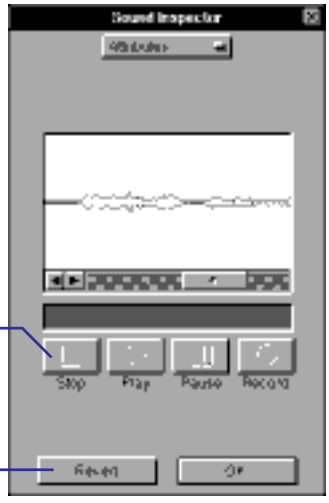
Tip: For most situations, the recommended course of action is to add images and sounds to your project. If you add them only to a nib file, they won't be available to an application until the nib file is loaded.

Images and sounds have their own Attributes displays. For images, this is mostly useful for images that are too large to show in the nib file window.



If your system has a microphone or some other input source connected, you can record new sounds. Click OK to save new sounds.

Select a sound and then select the Attributes display to see the sound's waveform. You can cut, copy, and paste the waveform.



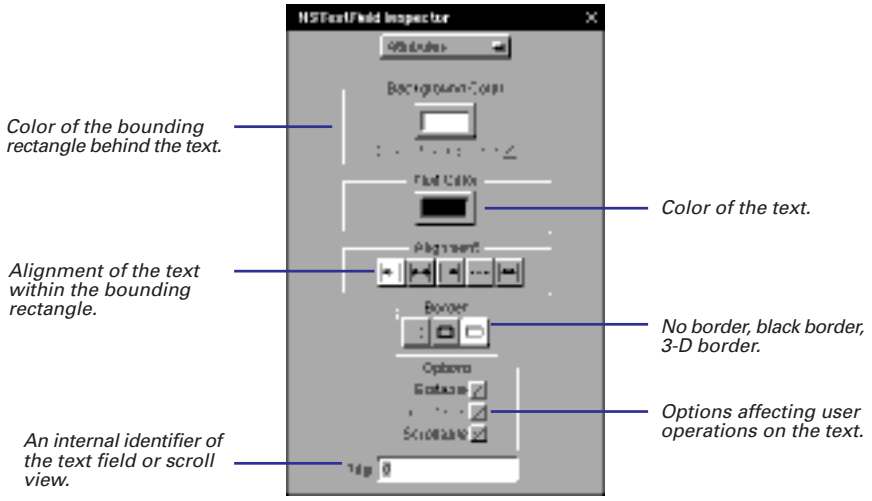
Controls to record and play sound. Above the controls is a horizontal sound meter.

Click to get back the original sound.

Customizing titles, text fields, and scroll views

▶ **Set background and text color, text alignment, border style, tag, and options affecting access to text.**

The objects that exist principally to display text—text fields, titles, and scroll views—have controls for initializing those objects with various characteristics. To see what certain effects look like, drag a text field onto a window and click the buttons on this display. The Title object is just a specially configured text field: non-selectable with transparent backgrounds and no borders.



An NSScrollView object is a compound object consisting of one or two NSScroller objects and an NSClipView object, which has as its document view (subview) an NSText object in Interface Builder. The document view is what is scrolled. The NSScrollView object has a slightly different Attributes display: no text alignment buttons and a different set of options.

A tag is an internal identifier of an object that you can use in your code. See "Using tags" in this chapter for more information.

For more information on the NSTextField, NSScrollView, NSScroller, NSText, and NSClipView classes, see the *Application Kit Reference*.

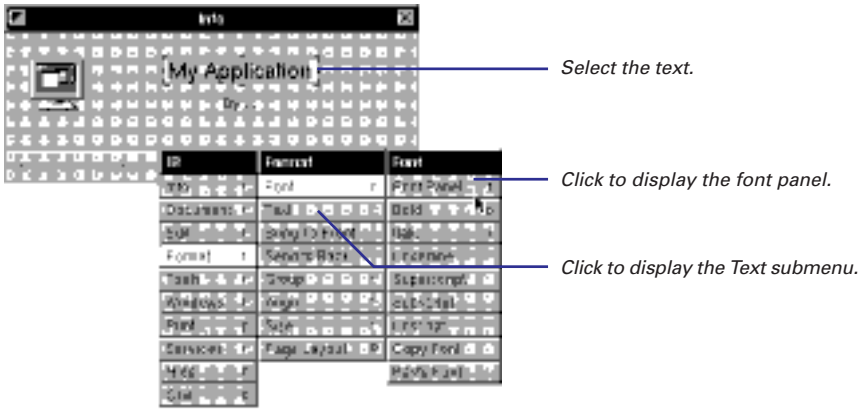
Text Field and Scroll View Options

Option	Description
Editable	Allows the user to edit text.
Selectable	Allows the user to select text.
Scrollable (NSTextField)	Text scrolls to the left if necessary.
Multiple fonts allowed (NSScrollView)	Text is in RTF format.
Graphics allowed (NSScrollView)	Text is in RTFD format (graphics can be inserted).

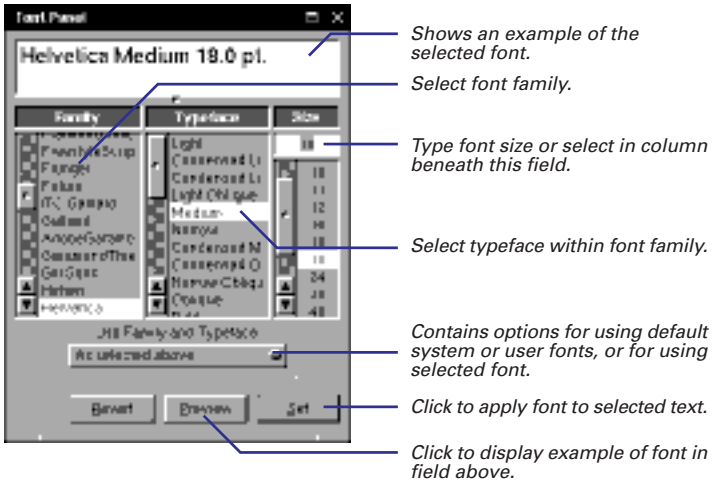
Setting textual attributes

- **Set the font characteristics of selected text using the Font Panel.**
- **Set the alignment of selected text within its boundaries using commands on the Text menu.**

Almost all palette objects—from buttons to browsers—can display text. You can set the font and alignment attributes of this text.



The Text submenu of the Format menu also has commands that affect selected text; it offers options for aligning text and for displaying, copying, and pasting the ruler in a Text object. With rulers you can set tabs and indentation. Note that rulers can only appear in NSText objects (for instance, inside a scroll view).



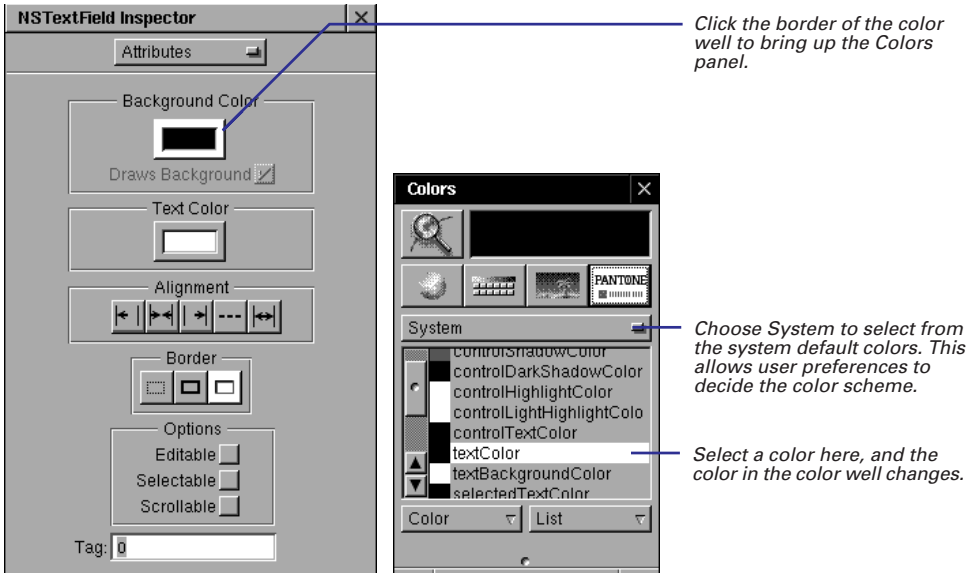
Setting text and background colors

1 **Select an NSTextField, NSScrollView, or NSMatrix object.**

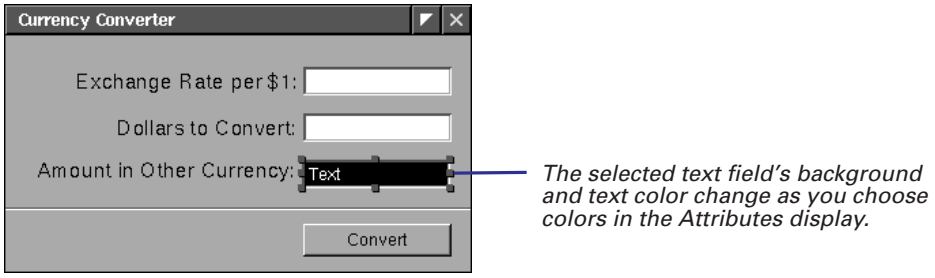
2 **In the Inspector panel, select the Background or Text color well.**

3 **In the Colors panel, choose a color.**

You can set the color of any of the text objects (NSTextField, NSScrollView, and NSMatrix). You can specify both a color for the text and a color for the background.



As with setting fonts, you can either choose to enforce a specific color or allow the user to select the color. For example, if you choose Black for the Text Color from the NeXT list in the PANTONE view, the text will be black for all users. However, if you choose textColor from the System list, the text will be the color the user selects in the system default settings (which is black unless the user changes it).



As a shortcut, drag a color from the Colors panel directly to the object to color its text. If you hold down the Shift key when you drag, the background is colored instead.

Setting mnemonics

▶ **Alternate-double-click a letter to choose that as the mnemonic.**

You can set a mnemonic in any object that displays a title, such as a form, a text field, or a button. Users can select that object by holding down the Alternate key and pressing the mnemonic you've chosen for the object. In this way, you give your users the option of navigating through the interface through the keyboard instead of the mouse.

Hold down the Alternate key and double-click the letter to set the mnemonic. The letter is underlined.

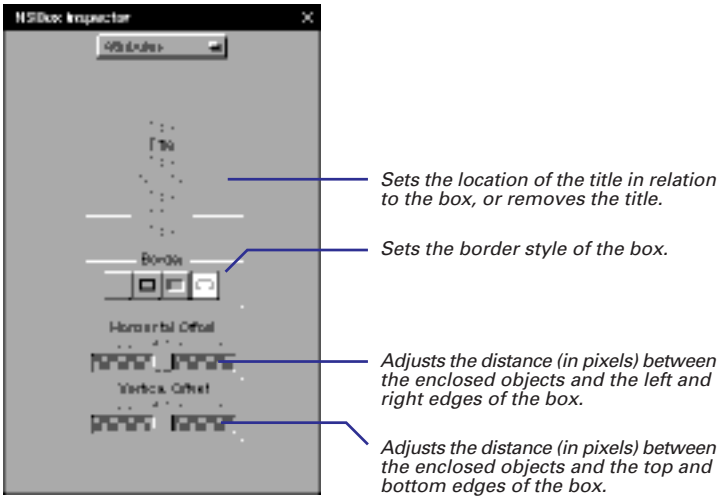
To delete a mnemonic, Alternate-double-click the letter again.

Mnemonics, together with inter-field tabbing, allow users to navigate through objects in a window using only the keyboard. Inter-field tabbing allows users to select objects by tabbing. See Chapter 4 for more information.

Setting box (group) attributes

- **Set title position, border style, and horizontal and vertical offsets.**

When you group a selection of objects in a box, that box (actually the box's content view) becomes the superview of the enclosed objects. In Interface Builder, you can move, copy, paste, and delete the group of objects as one. The box has several attributes that you can set.



See Chapter 2, "Composing the Interface," to learn how to group objects inside of a box.

For more information on box objects, see the NSBox class specification in the *Application Kit Reference*.

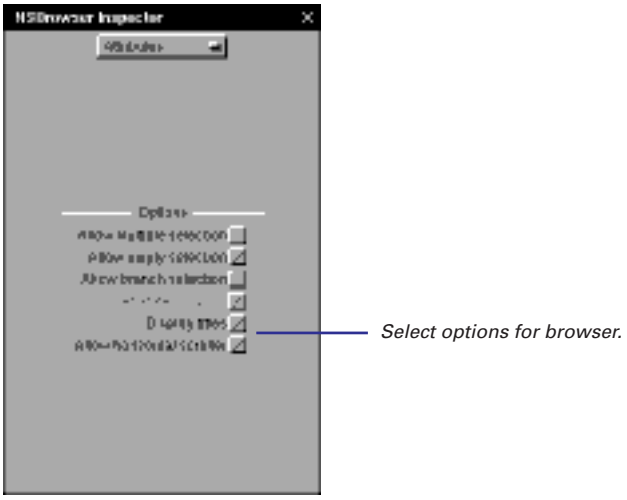
You can drag a box onto your interface and then programmatically replace its content view (blank by default) with another NSView object, or programmatically add subviews to the content view. You can also manipulate this box to make decorative rectangles and lines.

Tip: To make a line in an interface (such a a divider line between sections of a panel), drag a box onto the interface. Then switch off the title and make the box as narrow as possible in the required dimension (vertically or horizontally). Finally, set the offset (vertical or horizontal, whichever is applicable) to zero.

Customizing browsers

▶ **Select the browser options.**

Browsers display lists of data and allow users to select items from the list. They can hold one-dimensional lists or hierarchically organized lists of data such as directory paths. Browsers display these hierarchical levels in columns, which users can navigate using buttons or scrollers. An entry in a column can be either a *leaf node* or a *branch node*. Leaf nodes terminate a path; branch nodes, which have a right-arrow icon, lead into the next level in the hierarchy. A browser's attributes affect its navigation controls, methods of selection, and appearance.



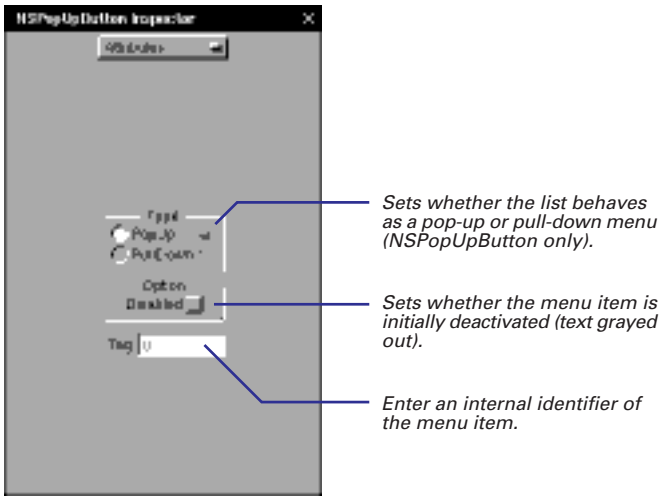
Browser Options

Option	Description
Allow multiple selection	The user can select more than one node at a time.
Allow empty selection	Makes it possible to have no cells selected; otherwise, the first cell in the column is selected by default.
Allow branch selection	The user can select branch nodes (such as directories).
Separate columns	Separates columns with a bezeled bar (if not set, a black line appears).
Display titles	Titles are above columns and column divider is bezeled bar.
Allow horizontal scroller	Allows users to scroll horizontally as well as vertically.

Setting attributes of menu items and pop-up lists

- **Set whether the list is a pop-up or pull-down type (not applicable to menu items).**
- **Set whether the item is initially disabled.**
- **Assign a tag to the item.**

Menus and pop-up or pull-down lists (NSPopUpButton instances) are compound objects containing objects that conform to the NSMenuItem protocol. The Attributes displays for menu items and NSPopUpButtons are almost identical. The following is the display for NSPopUpButtons.



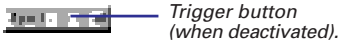
If you choose Disabled, the menu item's text is gray at application launch. When the user clicks the item, no action message is sent. If conditions change to make the items's function relevant, your code must re-enable the item.

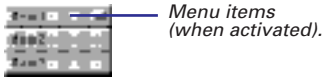
A tag is an internal identifier of an object that you can use in your code. See the task, "Using Tags," in this chapter for more information.

Once you expose a pop-up list's menu items, you can add more menu items to it from the Menus palette. See "Creating menus" in Chapter 2 for details.

Pop-Up Lists and Pull-Down Lists

An NSPopUpButton contains a trigger button and three menu items. Double-click the trigger button to see the menu items; you can initialize their titles or (in the Attributes display) disable them and assign them tags.





A *pop-up list*'s trigger button always displays the item that was last selected. In a *pull-down list* the trigger button's title is fixed. A pull-down list is effective for selecting actions in a very specific context, like the "Operations" pull-down list in Interface Builder's Classes display.

Compound Objects

Most of the objects you can drag from the standard Interface Builder palettes are actually compound objects. They consist of two or more objects that work together in specific ways.

Controls and Cells

A control (an instance of an NSControl subclass) functions as an event translator. It translates a user event like a mouse click into an action message and directs that message to another object in the application (the target).

Controls supply the mechanism but not the content of the target/action paradigm. They need action cells (or instances of NSActionCell subclasses) to hold this information:

- **target** the object receiving the action message
- **action** the method that specifies what the target is to do

At least one of these cells occupies the same area as its control. Because it descends from NSCell, a cell also has content (text or image), which it draws upon request from the control.

This division of responsibility makes for greater efficiency because a control can have multiple cells and send a different action message to a different target for each of those cells. Because cells are lightweight objects, it is more efficient in some contexts to associate one control with many cells.

Matrices

A matrix (an instance of the NSMatrix class) is a control that manages more than one cell. It organizes its cells in rows and columns. The cells must be the same size and usually are of the same class (although a matrix can have instances of different subclasses of NSCell).

Each cell in a matrix can have its own action and target. A matrix also has its own action and target. If a cell doesn't have an action, the matrix's action is sent to its target. If a cell doesn't have a target, the matrix sends the cell's action to its own target.

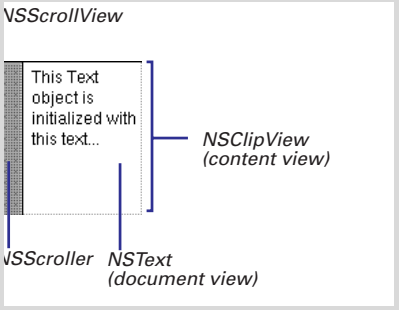
In Interface Builder, you can convert a single-celled control (such as an NSButton, NSSlider or NSTextField) into a matrix by Alternate-dragging a resize handle of that control. The associated cell, whether an NSButtonCell, NSSliderCell, or NSTextFieldCell, is duplicated for each row and column of the matrix.

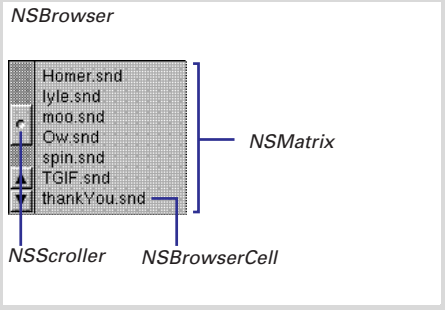
Forms are a special type of matrix (NSForm inherits from NSMatrix). They have special cells (NSFormCell instances) that compose both the form entry fields and the titles of those fields.

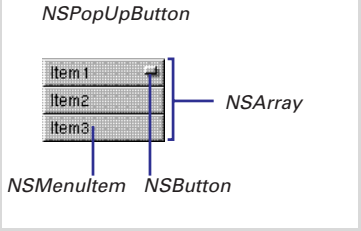
Special Compound Objects

Some objects on Interface Builder's standard palettes are of a more complex composition.

- **Scroll View** This object coordinates the interaction between NSScroller objects and an NSClipView object to scroll a document. It consists of one or two NSScrollers, an NSClipView, and the document view, which is generally NSText.
- **Browser** This object has scroll bars for controls and columns to show hierarchically organized data. Each column is a matrix of NSBrowserCell objects.
- **Pop-Up List** This object has a trigger button and an array of objects that conform to the NSMenuItem protocol.
- **Menu** This object's content area contains an array of objects that conform to the NSMenuItem protocol.
- **Table View** See "Inside the NSTableView Object" in this chapter.



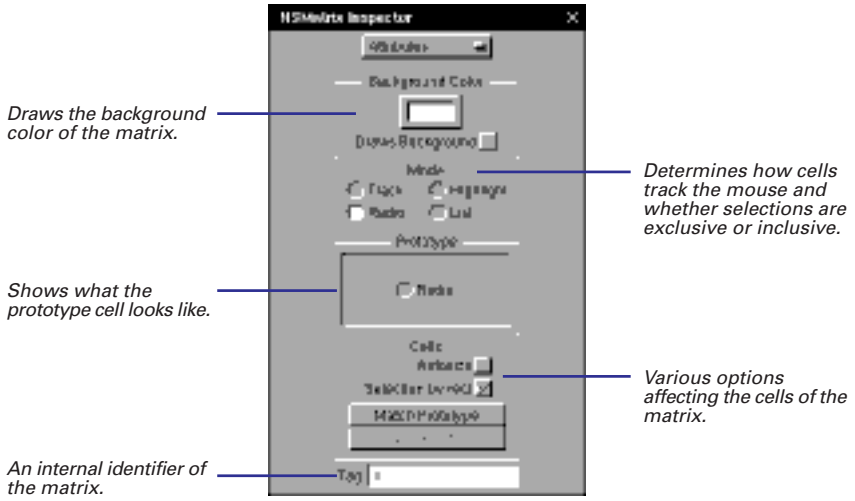




Setting matrix attributes

- **Set the background gray of the matrix.**
- **Set the matrix selection mode.**
- **Set autosizing behavior and other properties of cells.**
- **Inspect the cell prototype and change it, if necessary.**

The Attributes display for matrices allows you to determine how a matrix and its cells look and behave.



Matrix Selection Mode

The selection modes specify how cells behave when a user is dragging a mouse within a matrix. They also determine if the user can select multiple items in the matrix—a column of switch buttons, for example, allows multiple selection.

- **Track**: The cells track the mouse when it is within their bounds but do not highlight themselves. This mode would be suitable for a "graphic equalizer" matrix of sliders. Dragging the mouse causes the sliders to move.
- **Radio**: Only one cell in the matrix can be selected at a time, as is the typical case with a matrix of radio buttons.
- **Highlight**: Each cell is highlighted while it tracks the mouse and is unhighlighted when done tracking. This mode allows multiple selections. A matrix of switch buttons commonly has this mode.
- **List**: Cells are highlighted as the mouse is dragged across them, but they do not track the mouse. In this mode, a matrix supports multiple selection, enabling a user, for instance, to select a range of text in a matrix of text objects.

Cells Options

Option	Description
Autosize	If set, the cells resize when the matrix is resized, keeping the space between cells constant. If not set, the space between cells changes.
Selection by rect	Allows users to select multiple cells by dragging the mouse around them.
Match Prototype	Applies the new prototype to the selected matrix's existing cells.
Tags = Position	Resequences the cell's tags if you've added cells to a previously created matrix. When you create a matrix, cells are assigned consecutive tags starting from zero. For two dimensional matrices, the progression is from left to right (row), then down (column). When you later add new cells, they all have tags of zero.

A tag is an internal identifier of an object that you can use in your code. See "Using tags" in this chapter for more information.

Changing the Prototype Cell

When a matrix creates its cells, it typically makes them by copying a prototype cell stored as an instance variable. (It can also instantiate its cells from their class.)

You can examine and alter this prototype cell's attributes through the Inspector's Prototype display. This display is only available when you select a matrix.

If you change the prototype, you must click the Match Prototype button on the Attributes display of the matrix for the existing cells to reflect the changes.

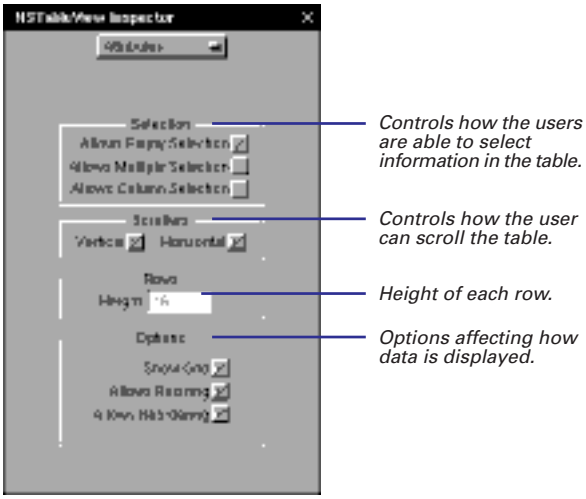
Choose Prototype here to display the prototype cell's attributes.

Setting up table views

- **Set the type of selections the user can make.**
- **Set whether the table view should scroll vertically, horizontally, or both.**
- **Set the height for table rows.**
- **Set if the user can resize columns, reorder rows, and if cells are bordered.**

A table view displays information in a table and allows the user to select and change that information. Table views contain rows and columns of information. When you create a table view in Interface Builder, you create the number of columns you want. Rows are added programmatically.

Tip: To add a column to a table view, select an existing column, then copy and paste.



The Selection options control how the user can select information in the table view. By default, the user can select rows in the table one at a time. Allows Column Selection means the user can select columns of information. Allows Multiple Selection means the user can select more than one row or column at a time.

Table View Options

Option	Description
Show Grid	Lines are drawn around each cell in the table.
Allows Resizing	The user can resize the table columns.
Allows Reordering	The user can rearrange the table rows.

Inside the NSTableView Object

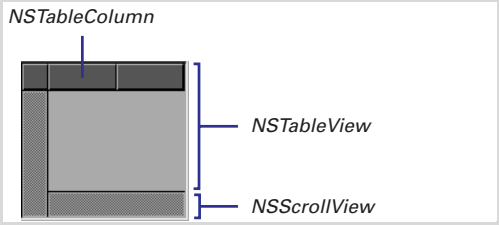
NSTableView used to be available only to people using the Enterprise Objects Framework or, before that, DBKit. Now, NSTableView is part of the Application Kit, so every application can take advantage of its features.

When you drag a table view from the TabulationViews palette to your interface, you're actually getting several objects. The NSTableView is nested inside of an NSScrollView. The NSTableView itself is made up of one NSTableColumn object for each column and an NSTableHeaderView, which displays the column headings.

Each NSTableColumn has an NSCell associated with it that is used to draw all of the cells in that column. The NSCell may have an NSFormatter associated with it that defines how the contents of that cell are formatted. You can associate an NSFormatter with the NSTableColumn's NSCell in Interface Builder by dragging one from the Formatters palette.

Also associated with an NSTableView is an object conforming to the NSTableDataSource protocol. You don't create this object in Interface Builder unless you're creating an application based on the EO Framework. The data source controls the display of data in the NSTableView. You implement methods defined by the protocol to retrieve values from the table, to change values in the table, or to add rows to the table.

For more information about table views, see the NSTableView class specification in the *Application Kit Reference*.



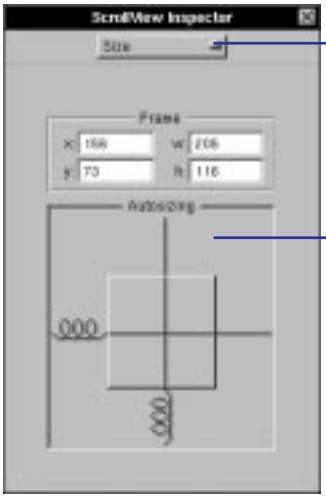
Automatically resizing objects

1 **Select an object.**

2 **Choose the Size display of the Inspector panel.**

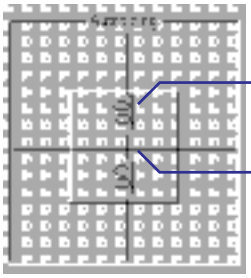
3 **In the Autosizing view of the display, click lines to make them springs or click springs to make them lines.**

When you resize a window, the objects in the window must often adjust their size or the distances between themselves and other objects. The Size display of the Inspector panel lets you tell a selected object how to resize itself. The lines inside and outside the box affect different aspects of resizing behavior.



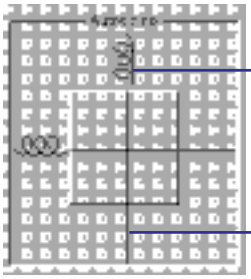
Choose Size here.

Click to toggle between a line and a spring, setting resizing characteristic..



Inside the box
This spring indicates that, when the window or superview is resized vertically, the object resizes itself to maintain its distance from the top and bottom edges of the window or superview.

This sraight line indicates that, when the window or superview is resized horizontally, the object maintains its initial size.

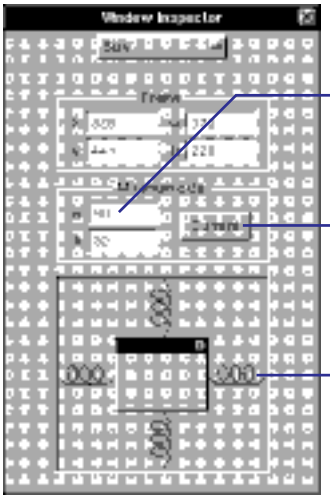


Outside the box
This spring indicates that, when the window or superview is resized vertically, the space between the top edge of the object and the top of the enclosing view or window is adjusted proportionally.

This sraight line indicates that, when the window or superview is resized, the object maintains the initial distance between its bottom edge and the bottom of the enclosing view or window.

For examples of the effects of these "autosizing" characteristics on views within a resized window, see "Some Effects of Automatic Resizing."

If you do not make a view resize itself when its superview or window resizes, some ugly behavior could result. For instance, if the user makes a window small, objects that don't resize themselves could become truncated by the resized window's borders. One recourse to this unwanted outcome is to specify a minimum size for the window.



You might need to make several iterations in Interface Builder—setting resizing characteristics in objects and shrinking the window in test mode—to determine what the ideal minimum size should be.

Interface Builder includes a test mode that simulates the actual operation of the interface. In test mode, you can test the resizing behavior of your windows and views, see how connected objects communicate, play sounds associated with buttons, and do similar operations. See "Testing the Interface" in Chapter 4, "Making and Managing Connections," for more information.

When There Are Conflicts

You can create an impossible resizing relationship, such as specifying as fixed the object's dimensions and its distance from the window's edges. In cases of conflict, an object's fixed dimension takes precedence over its fixed distance from a border. If all dimensions are made resizable, adjustments to the window or superview's changed dimensions are made equally to the object and its distance from a border.

Some Effects of Automatic Resizing

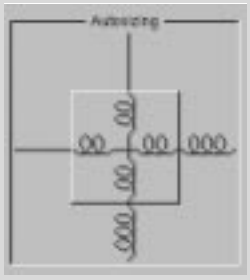
The window below has two identical scroll view objects. Different autosizing "springs" are set in each, and then the window is resized in test mode. The screen shots under After Resizing show you the results.

In the first example, one object resizes vertically while the other doesn't (distances to borders are absolute for both). The result: the object that doesn't resize itself is truncated when the window is vertically shortened.

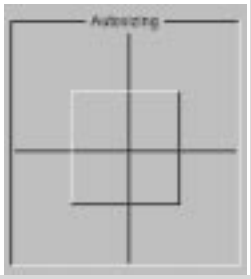
In the second example, both objects resize themselves, but Object B maintains its distance to surrounding objects. This causes Object B to be more severely resized than Object A.

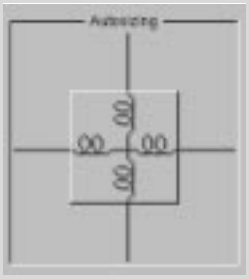
To learn more about the effects of resizing, try some experiments on your own using different combinations of objects and autosizing attributes.

Object A

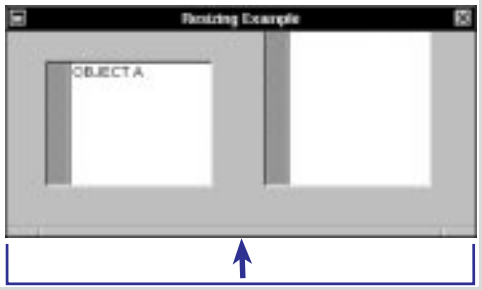


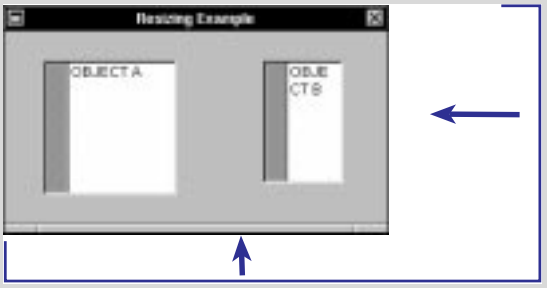
Object B





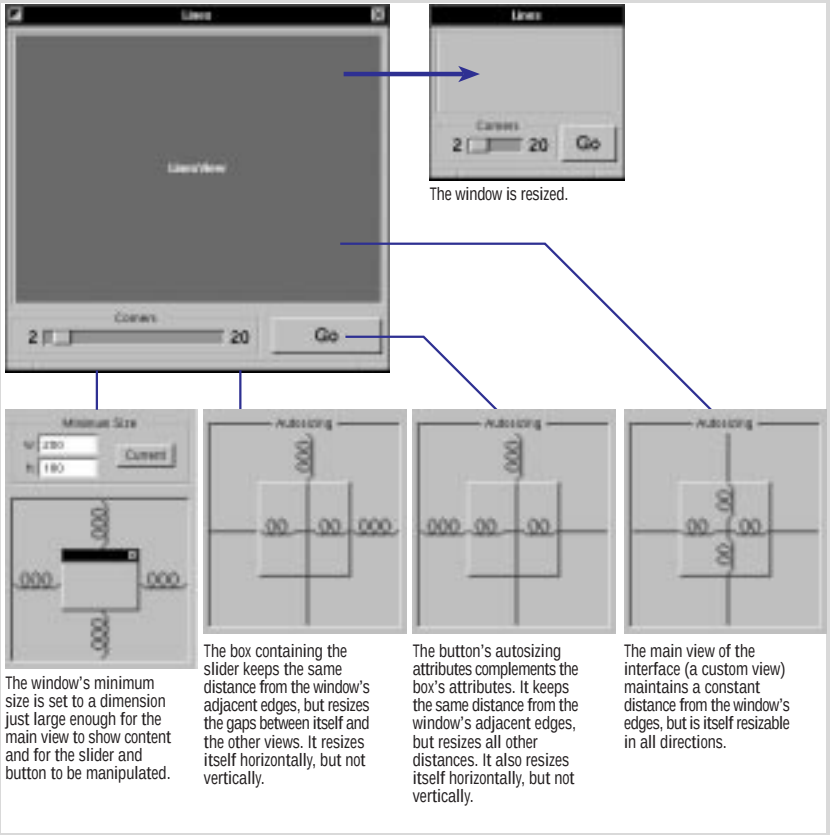
After Resizing





Automatic Resizing: An Example

This example interface incorporates autosizing attributes in such a combination that the window can shrink to a very small size and still be usable.



The window is resized.

The window's minimum size is set to a dimension just large enough for the main view to show content and for the slider and button to be manipulated.

The box containing the slider keeps the same distance from the window's adjacent edges, but resizes the gaps between itself and the other views. It resizes itself horizontally, but not vertically.

The button's autosizing attributes complements the box's attributes. It keeps the same distance from the window's adjacent edges, but resizes all other distances. It also resizes itself horizontally, but not vertically.

The main view of the interface (a custom view) maintains a constant distance from the window's edges, but is itself resizable in all directions.

Using tags

1 **In Interface Builder, specify the tag integers for objects.**

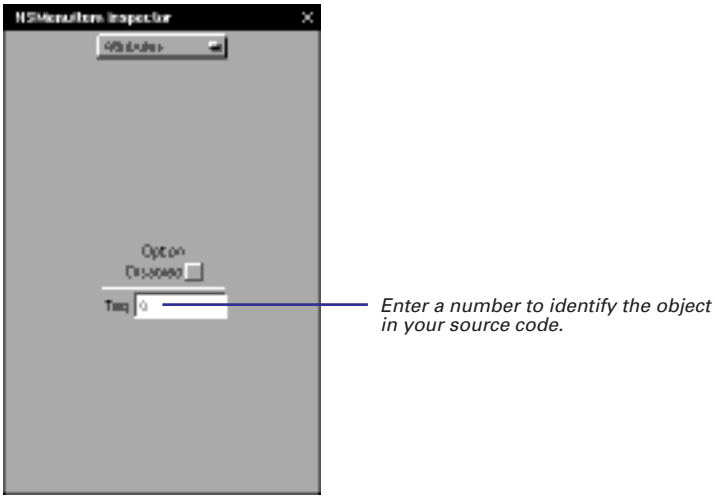
2 **If the integers are not intrinsically meaningful, define constants for them in your source code.**

3 **Send the tag message to a tagged object to get the integer.**

4 **Evaluate the integer and act upon it.**

Tags are integers that you use in your code to identify objects. They offer a convenient alternative to such methods of object identification as fetching an object's title. (What if the object's title changes while the application is running, or the application is localized?) Tags can also carry useful information associated with an object, and thus make it easier to integrate that information into a program. Tags are commonly assigned to the cells contained by matrices.

You can specify tags in the Tag fields of most Attributes displays.



You can also set tags programmatically in most NSView objects by sending those objects the **setTag:** message.

The integers that you assign could have some intrinsic value; for instance, they could be numbers that are multiplication factors for a document-zoom feature, or numbers that correspond to the number of a keypad in a calculator application. If the tag numbers are not intrinsically meaningful (that is, they're arbitrary), it's prudent to define constants to express them.

```
typedef enum {
    LEFT = 1,
    RIGHT,
    BOTTOM,
    TOP,
    HORIZONTAL_CENTERS,
    VERTICAL_CENTERS,
    BASELINES
} AlignmentType;
```

When you need to identify a tagged objects in your code, use the **tag** method.

```
- (void)align:sender
{
    [self alignBy:(AlignmentType)[[sender selectedCell] tag]];
}
```