What'stheDifferenceBetweenaWindowandaPanel?;¬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.

28012_TableRule.eps ¬

TheAnatomyofaButton;¬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 ªnormalº (off) and ªalternateº (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 ªCompound Objectsº in this chapter ;SettingAttributesConcepts.rtfd;CompoundObjects;¬.) 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.

28012_TableRule.eps ¬

CompoundObjects;¬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 ªInside the NSTableView Objectº in this chapter.
  ;SettingAttributesConcepts.rtfd;InsidetheNSTableViewObject;¬

_NSScrollView.eps ¬_NSBrowser.eps ¬_NSPopUpButton.eps ¬

28012_TableRule.eps ¬

ChangingthePrototypeCell;¬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.

_ChangingProCell.eps ¬

28012_TableRule.eps ¬

InsidetheNSTableViewObject;¬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*. ;/NextLibrary/Frameworks/AppKit.framework/Resources/English.lproj/Documentation/Reference/Classes/NSTableView.rtfd;;¬

_NSTableView.eps ¬

28012_TableRule.eps ¬

SomeEffectsofAutomaticResizing;¬Some Effects of Automatic Resizing

The window below has two identical scroll view objects. Different autosizing ªspringsº 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.

_AutoResize7.eps ¬

_AutoResize2.eps ¬ _AutoResize3.eps ¬ _AutoResize4.eps ¬ _AutoResize5.eps ¬ _AutoResize6.eps ¬ _AutoResize7.eps ¬

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.

_AutoResizeExample.eps ¬