

2: WINDOWS

The Java platform provides several types of **windows**, each for a different type of interaction. To help you choose appropriate windows types for your application, this chapter:

- Introduces objects and properties, which are displayed in windows
- Provides an overview of window types
- Explains how to choose the correct window type
- Describes various window types in detail
- Describes how to title windows and set their state
- Provides guidelines about using multiple document interfaces

This chapter supplements Chapters 7 and 8 of *Java Look and Feel Design Guidelines*, 2d ed.

In this chapter, the **dialog box** window type is subdivided into **action windows** and **property windows**, both described here.

For information about using menus in windows, see Chapter 3.

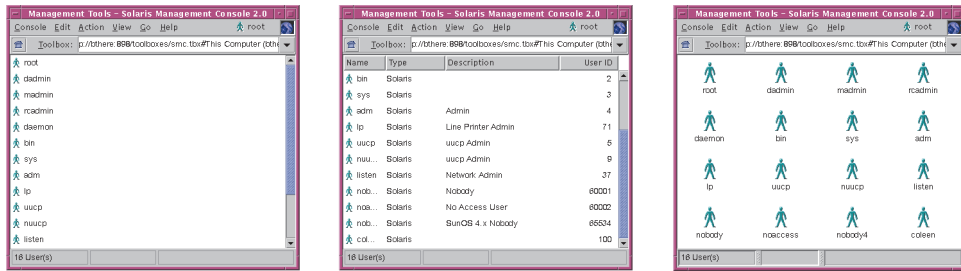
Windows, Objects, and Properties

Windows can display user interface objects. An **object** is a logical entity that an application displays and a user manipulates—for example, a document or paragraph in a word-processing application. User interface objects do not necessarily correspond to Java programming language objects in an application's code. User interface objects represent data or other parts of a user's tasks.

User interface objects have characteristics called **properties**. For example, a paragraph might have a property that determines whether it is indented. Users can view or set the values of properties.

Applications can display a single object in more than one **view**. For example, at a user's request, an application might display the same objects as list items, table entries, or **icons**, as shown in Figure 1.

FIGURE 1 Different Views of the Same Objects



Overview of Window Types

The Java platform provides the following

basic window types:

- Plain windows
- Utility windows
- Primary windows
- Secondary windows

Figure 2 shows these window types and their subtypes.

FIGURE 2 Window Types

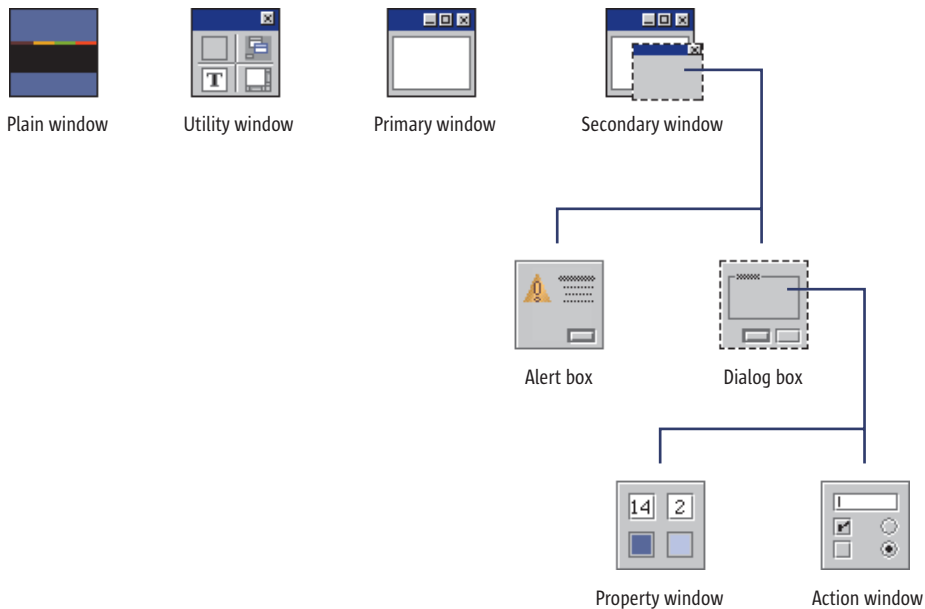


Table 1 lists each window type and describes its intended use.

TABLE 1 Window Types and Intended Use

Window Type	Intended Use
Plain window	Typically, displays a splash screen, which appears briefly in the time between when an application starts and when the application’s main window appears.
Utility window	Displays a set of tools (for example, the drawing tools in a graphics program), or enables other user interaction that can affect a primary window.
Primary window	Represents an object or a set of objects. A primary window can have any number of dependent, or secondary, windows. For more information, see “Primary Windows” on page 10.
Secondary window	<p>An alert box or a dialog box:</p> <p>Alert box—Enables brief interaction with a user—for example, to display error messages or warn of potential problems. For more information, see “Alerting Users After an Object’s State Changes” on page 30.</p> <p>Dialog box—A property window or an action window:</p> <ul style="list-style-type: none">• Property window—Enables a user to display or set the properties of one or more objects, typically objects in the parent window (which opened the property window). For more information, see “Property Windows” on page 12.• Action window—Prompts a user for information needed to perform an action (such as opening a file). The user requested the action from the parent window. Action windows are not for displaying or setting properties of objects. For more information, see “Action Windows” on page 22.


Window Types for Objects, Properties, and Actions


A window’s intended use determines its correct window type. Choosing the correct window type is especially important when displaying objects or properties.

Only two window types are intended for displaying objects and their properties:

- Primary windows
- Property windows

You can use an action window to let users perform actions on an object. In addition, you can enable users to perform actions on objects by providing drop-down menus or equivalent **controls**.

 To represent an object or a set of objects, use a primary window. To represent an object's properties, use a property window. Use these window types only for these purposes.

 When providing a window for performing actions on an object, use an action window. However, do not use an action window to display or set the properties of an object. Use a property window instead.

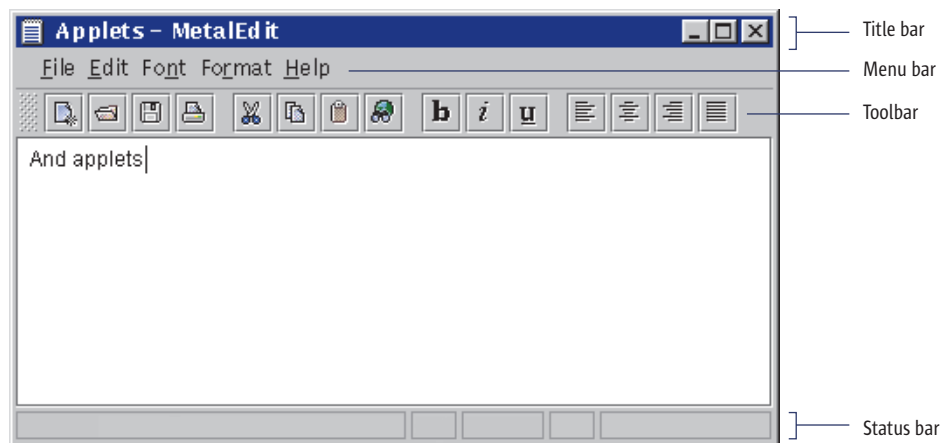
Primary Windows

A primary window is the main window in which a user interacts with a document or data. An application can have one or more primary windows, each of which a user can manipulate independently.

A primary window represents an object (such as an email message) or a set of objects (such as all the messages in a mail window). For information about representing the properties of objects, see “Property Windows” on page 12.

Primary windows contain a **title bar** and, optionally, a **menu bar**, **toolbar**, and **status bar**, as shown in Figure 3.

FIGURE 3 Elements of a Primary Window



Title Bars in Primary Windows

The title bar of a primary window displays text that includes the name of the object, or set of objects, that the window represents. Figure 4 shows a typical title bar for a primary window.

FIGURE 4 Title Bar of a Primary Window

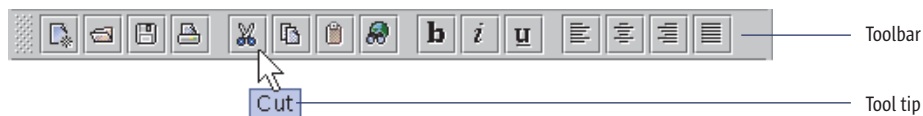


For more information about window titles, see Chapter 7 of *Java Look and Feel Design Guidelines*, 2d ed. In addition, see “Window Titles for Identically Named Objects and Views” on page 26 of this book.

☞ In primary windows, begin the window title text with the name of the object or set of objects that the window represents, followed by a space, a hyphen, another space, and the application name.

Toolbars in Primary Windows Primary windows can contain a toolbar, as shown in Figure 5.

FIGURE 5 Toolbar of a Primary Window



A toolbar can contain any combination of the following controls:

- **Command buttons**—for example, a button for printing or searching
- Controls for choosing modes of interaction—for example, buttons for choosing painting tools in a graphics application
- Controls that dynamically change and display an object’s property values—for example, in a word-processing program, a button that both italicizes the current text and shows that the current text is italicized

☞ If users can access an action from a toolbar, provide keyboard access to that action as well. Alternatively, provide an equivalent action that is keyboard accessible. For example, if you provide a toolbar control for printing text, you might also provide a menu item for printing text. Toolbar controls can differ from their corresponding menu items—but only in that the toolbar control uses default values, whereas the menu item opens an action window.

☞ When providing a keyboard-accessible alternative to a toolbar control, ensure that the alternative has at least the capabilities of the toolbar control.

☞ Omit all toolbar controls from the **keyboard traversal order** so that expert users can navigate more easily. If a user presses the Tab key to move keyboard focus in a window, do not move focus to toolbar controls.

☞ Provide a tool tip for each toolbar control, especially if the control has no label. (For more information about tool tips, see “Tool Tips” on page 62.)

Status Bars in Primary Windows The bottom of primary windows can include a status bar, as shown in Figure 6.

FIGURE 6 Status Bar at the Bottom of a Primary Window



You can use the status bar to display status messages and read-only information about the object that the window represents.

☞ In a window’s status bar, ensure that each message fits the available space when the window is at its default size.

☞ To avoid displaying obsolete information in a window’s status bar, clear each status message when the next user action occurs in that window.

Property Windows A property window is a dialog box in which users can display or change values of one or more object properties.

Property Window Characteristics A property window has four main behavioral characteristics—one from each of the following pairs:

- **Modal** or **modeless**
- **Single-use** or **multiple-use**
- **Dedicated** or **non-dedicated**
- **Inspecting** or **non-inspecting**

This section discusses only the following characteristics:

- Dedicated
- Non-dedicated
- Inspecting
- Non-inspecting

For a discussion of the remaining characteristics, see Chapter 8 of *Java Look and Feel Design Guidelines*, 2d ed.

Table 2 describes each main behavioral characteristic that can apply to property windows.

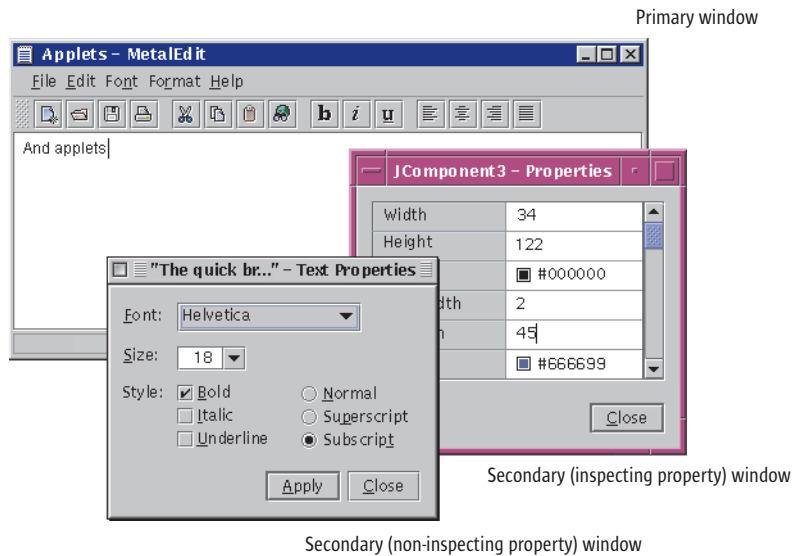
TABLE 2 Property Window Characteristics

Characteristic	Description
Modal	Prevents a user from interacting with other windows in the current application.
Modeless	Does not prevent a user from interacting with other windows in the current application.
Single-use	Intended for users who are likely to perform only one operation with the dialog box before dismissing it.
Multiple-use	Intended for users who are likely to perform more than one operation with the dialog box before dismissing it.
Dedicated	Affects only objects already selected when the property window opened.
Non-dedicated	Affects currently selected objects, even if the current selection changes.
Inspecting	Displays a continuously updated view of the property values for the currently selected object, even if the values change.
Non-inspecting	Displays a static view, or snapshot, of the selected object's property values.

Only a few combinations of the characteristics in Table 2 are recommended, so choosing the correct property window characteristics is simpler than it might seem. This section describes how to make the correct choices. Later sections describe each property window characteristic in detail.

Figure 7 shows a primary window, an inspecting property window, and a non-inspecting property window.

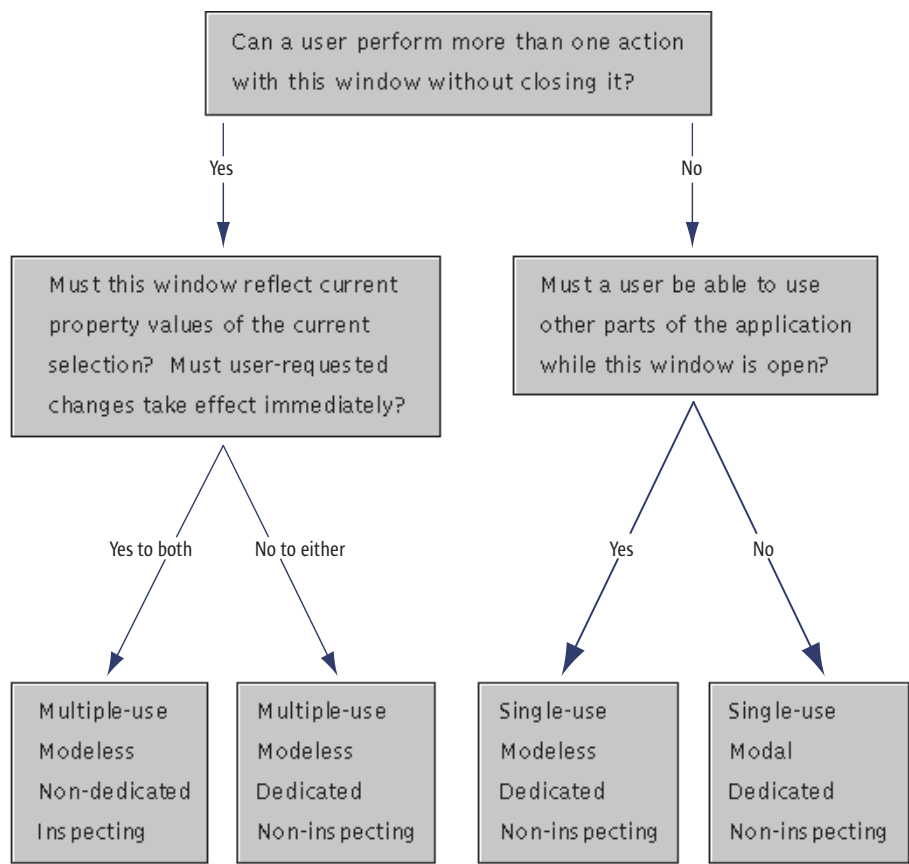
FIGURE 7 Property Windows and a Primary Window



For information about positioning a property window in relation to its parent window, see “Positioning Secondary Windows” on page 29.

Choosing the Correct Property Window Characteristics Before choosing characteristics for your application’s property windows, consider how users should interact with each window. A property window’s intended use determines its correct window characteristics. Figure 8 helps you choose the correct characteristics for property windows.

FIGURE 8 Steps for Choosing Property Window Characteristics



Of the sets of property window characteristics in Figure 8, only two sets are typically used in applications. Table 3 provides examples of property windows whose characteristics match those of the typical sets.

TABLE 3 Examples of Typical Property Windows

Property Window Characteristics	Example
Multiple-use, modeless, non-dedicated, inspecting	The small windows for choosing colors or layers in graphics applications such as Adobe® Photoshop software.
Single-use, modeless, dedicated, non-inspecting	The Preferences dialog box of a typical application.

Dedicated and Non-Dedicated Property Windows A dedicated property window affects only objects already selected when the property window opened. Changing the selection while a dedicated property window is open does not change which objects the property window affects.

In contrast, a non-dedicated property window affects only objects currently selected—even if the selection changes while the property window is open. In other words, a non-dedicated property window affects whichever objects are currently selected when a user clicks the window’s OK button or Apply button. In a non-dedicated property window, a user can change which objects the window affects. To do so, the user can select different objects while the window is open.

Inspecting and Non-Inspecting Property Windows Property windows are inspecting or non-inspecting, depending on:

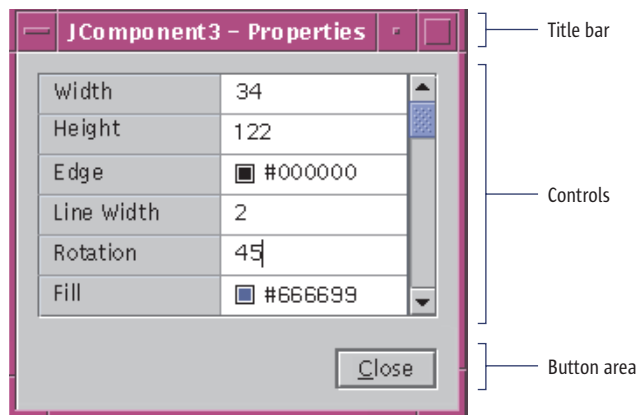
- How current their displayed information is
- When a user’s changes take effect

Inspecting Property Windows An inspecting property window is a dialog box that both:

- Displays a continuously updated view of the property values for the selected object, and
- Enables a user to change the displayed property values (and the selected object) immediately

Figure 9 shows an inspecting property window.

FIGURE 9 Inspecting Property Window



In inspecting property windows, a user does not click an OK button or Apply button to apply changes. The application applies changes automatically.

An inspecting property window displays the values of the selected object. If a user changes the selection, the values in the property window also change immediately to reflect the newly selected object. An inspecting property window continuously updates its view of an object's property values, even if those values change outside a user's control. Most inspecting property windows are modeless.

If your application has many types of objects, avoid creating a separate inspecting property window for each type. Instead, create a single inspecting property window whose contents change depending on the properties of the selected object.

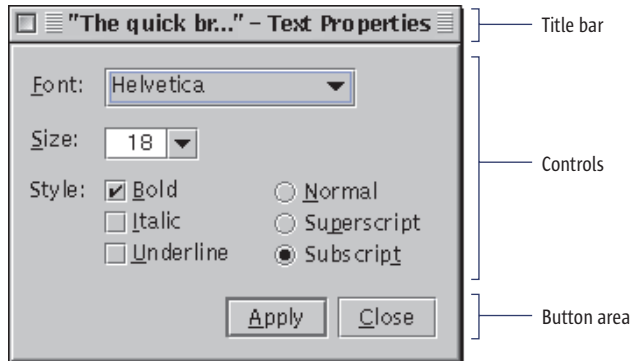
If users need to update two or more interdependent property values, do not provide an inspecting property window. Instead, provide a non-inspecting property window, thereby ensuring that changes to interdependent properties occur at the same time.

Here is an example of why inspecting property windows are inappropriate for updating interdependent property values. An application has a Customer object for which users can enter an address that includes a city and country—such as Paris, France. As a user types the address, the Customer object automatically verifies that the specified city (`Paris`) is in the specified country (`France`).

In an inspecting property window, if a user tries to change the city name from `Paris` to `Tokyo`, the Customer object rejects the change because the user has not changed the country name from `France` to `Japan`. If the user then tries to change the country name, the change is again rejected, because the city name has not been changed from `Paris` to `Tokyo`.

Non-Inspecting Property Windows A non-inspecting property window is a property window that displays a static view, or snapshot, of the selected object's property values—accurate as of the time that the property window opened. Figure 10 shows a non-inspecting property window.

FIGURE 10 Non-Inspecting Property Window



Non-inspecting property windows are particularly useful when a user needs to update several interdependent values at the same time.

If a user changes property values in a non-inspecting property window, those changes take effect only if the user clicks the window's OK button or Apply button. Changes that take place beyond the user's control are not reflected in the window until it opens again.

NOTE - If your application's objects can change outside a user's control, alert the user to any such changes. For more information, see "Alerting Users After an Object's State Changes" on page 30.

Behavior and Layout of Property Windows For property windows, the correct behavior and command-button layout depend on the window's characteristics. This section provides guidelines for the behavior and layout of property windows.



Place no menu, toolbar, or status bar in property windows.



To enable users to open a property window, place an item labeled Properties on the *Object* menu, if there is one. (*Object* stands for the type of the object whose properties the window displays—for example, Document.) If your application has no *Object* menu, place the Properties item on the Edit menu. Label the item *Object* Properties if the Edit menu also contains items for other property windows.

Title Text in Property Windows Property windows include title text, displayed in the title bar, as shown in Figure 11.


FIGURE 11 Title Text in the Title Bar of a Property Window





Regardless of a property window's characteristics, the title text consists of the following items, in order:

1. The name of the object that the window represents
2. A hyphen, preceded by one space and followed by another
3. The name of the command that opened the window

In Figure 11, the window represents an object named SuperRivet. The command that opened the window is Alloy Properties.

 In property windows, format the title text as *Object Name - Command*, as shown in Figure 11. *Object Name* stands for the name of the currently displayed object. Precede the hyphen by one space and follow it by one space. *Command* stands for the name of the command that opened the property window.

 If a user might not know which application created a particular property window, include the application's name in that window's title text. Format the title text like this: *Object Name - Command - Application Name*. (Precede each hyphen by one space and follow it by one space.)

 In the title text of inspecting property windows, update the current object's name each time you update the window's contents.

Command Buttons in Non-Inspecting Property Windows Table 4 describes the command buttons you can place in non-inspecting property windows.

TABLE 4 Command Buttons for Non-Inspecting Property Windows

Button	Description
Apply	Updates the properties of the associated object.
Reset	Discards any changes made in the window since the last “apply” action. The Reset command then refills the window's fields with the values from the associated object.
OK	Updates the properties of the associated object and then closes the window.

TABLE 4 Command Buttons for Non-Inspecting Property Windows *(Continued)*

Button	Description
Close	Closes the property window but not the application. If a user has changed the values in the window but has not applied them, the Close button opens an alert box containing the following text: “Your changes have not been saved. To save the changes, click Apply. To discard the changes, click Discard. To cancel your Close request, click Cancel.”
Cancel	Works like the Close button, except that the Cancel button does not display an alert box before discarding changes.
Help	Displays help text in another window while leaving the property window open.

Before deciding which command buttons to place in a non-inspecting property window, estimate how many times a user needs to use the window before closing it.

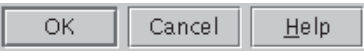
If a user will use a property window only once before closing it, then place an OK and a Cancel button—in that order—at the bottom right of the window, as shown in Figure 12.

FIGURE 12 Required Buttons for a Single-Use Property Window



Optionally, you can add a Help button to the right of the Cancel button, as shown in Figure 13.

FIGURE 13 Required and Optional Buttons for a Single-Use Property Window



If a user will use a property window repeatedly before closing it, place an Apply and a Close button—in that order—at the bottom right of the window, as shown in Figure 14.

FIGURE 14 Required Buttons for a Repeated-Use Property Window




Optionally, you can place a Reset button between the Apply button and the Close button, and place a Help button to the right of the Close button, as shown in Figure 15.


FIGURE 15 Required and Optional Buttons for a Repeated-Use Property Window





The following guidelines apply only to non-inspecting property windows.


 Place either an OK button or an Apply button in non-inspecting property windows.


 Make the OK button or the Apply button the default command button. (For more information about default command buttons, see Chapter 10 of *Java Look and Feel Design Guidelines*, 2d ed.)


 Place the dismissal button to the right of the OK button or Apply button.


 If a non-inspecting property window has an OK button, label its dismissal button Cancel. Otherwise, label the dismissal button Close.

 Ensure that clicking the title bar's close-window control has the same effect as clicking the window's Close or Cancel button.

 Open an alert box if a user clicks the Close button before applying changes entered in the window. In the alert box (which includes a Discard button), display the following text: "Your changes have not been saved. To save the changes, click Apply. To discard the changes, click Discard. To cancel your Close request, click Cancel."

 If a non-inspecting property window has an Apply button, ensure that clicking the Apply button updates the associated object, using the current values from the property window.

 If a non-inspecting property window has an OK button, ensure that clicking the OK button updates the properties values of the associated object and then closes the window.

 If a non-inspecting property window needs a Reset button, place that button between the window's Apply and Close buttons.

☺ Ensure that clicking the Reset button performs the following operations, in order:

1. Discards any changes made in that window since it opened, or since the last “apply” operation
2. Refills the window’s fields with the current values of its associated object

Command Buttons in Inspecting Property Windows In inspecting property windows, place a Close button at the bottom right of the window. Optionally, place a Help button to the right of the Close button.

The following guidelines apply only to inspecting property windows.

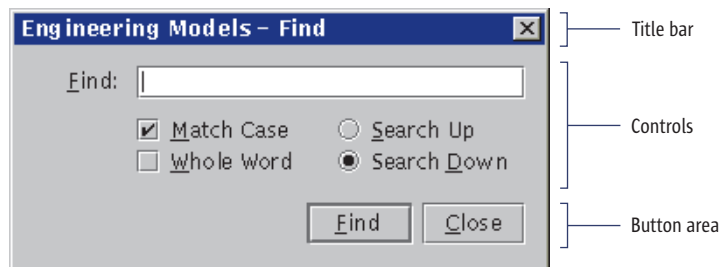
☺ Ensure that clicking the Close button immediately closes the window.

☺ Ensure that clicking the title bar’s close-window control has the same effect as clicking the window’s Close button.

☺ Some controls do not immediately send their updates to the object being inspected. (For example, a text field does not send its updated text until it has lost input focus.) In inspecting property windows, send all pending updates to the window’s object when a user clicks the window’s Close button or close-window control.

Action Windows Action windows are dialog boxes that request information for completing an action. Open an action window if a user activates a menu command or command button that requires additional user input to complete an action. Figure 16 shows an action window.

FIGURE 16 Action Window



As shown in Figure 16, an action window contains:

- Title bar
- Controls
- Button area

An action window has no menu, toolbar, or status bar.


The label of a menu command or command button that opens an action window ends with an **ellipsis (...)**. The ellipsis means that the command requires additional user input.


Like property windows, action windows can be:

- Modal or modeless
- Single-use or multiple-use

For more information about these characteristics, see Chapter 8 of *Java Look and Feel Design Guidelines*, 2d ed.

For information about positioning an action window in relation to its parent window, see “Positioning Secondary Windows” on page 29.

 Ensure that an action window has no menu, toolbar, or status bar.

 Place an ellipsis (...) at the end of the label for a menu command or command button that opens an action window.

Title Text in Action Windows The title text of an action window helps users understand the window’s purpose. An action window’s title text includes:

- The name of the object that the action window affects (if you know the name)
- The name of the menu item or command button that opened the action window (omitting any trailing ellipsis)

Figure 17 shows the title text of an action window opened from a Print menu item. The window affects an object named `MySalesForecast`.

FIGURE 17 Title Text in the Title Bar of an Action Window





If an action window enables a user to create an object, the window’s title text cannot include that object’s name because the object does not yet exist. Figure 18 shows the title text of such an action window, opened from a New Rivet menu item.


FIGURE 18 Title Text of an Action Window That Creates a New Object





In Figure 18, the title text shows only the name of the menu item that opened the action window.

 In the title text of an action window, include the name of the menu item or command button that opened the action window. Omit the name's trailing ellipsis, if there is one.

 If an action window affects an existing object, format the title text like this: *Object Name - Menu Item*. (Precede the hyphen by one space and follow it by one space.)

 If an action window creates a new object, make the window's title text the name of the menu item or command button that opened the window.

 If a user might not recognize the source of a particular action window, format the title text like this: *Object Name - Menu Item - Application Name*. (Precede each hyphen by one space and follow it by one space.)

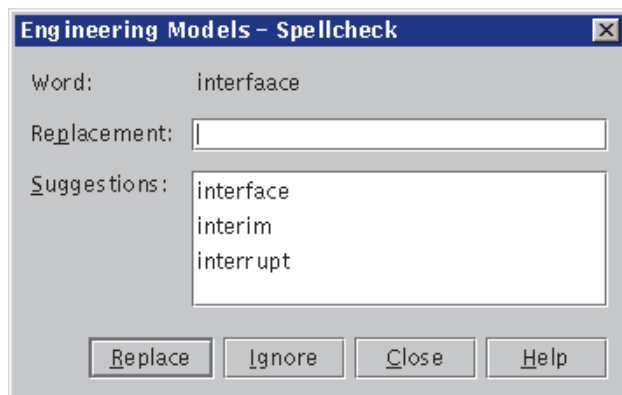
 In a modeless action window, ensure that the title text always reflects the object that the window affects.

Command Buttons in Action Windows Action windows have the following command buttons (in left-to-right order):

- One or more command buttons that perform actions
- One dismissal button
- One Help button (optional)

The command buttons are right-justified in the window's button area, as shown in Figure 19.

FIGURE 19 Command Buttons in an Action Window (Multiple-Use)



An action window's leftmost command button performs an action using values from the window's controls. For example, in Figure 19, the Replace button performs an action using the value that the user enters in the input field labeled Replacement.

The Help button, if there is one, is always rightmost in the button area, as in Figure 19. (For more information about the Help button, see Chapter 8 of *Java Look and Feel Design Guidelines*, 2d ed.)

An action window's rightmost command button (or the one directly to the left of the Help button) closes the window.

If an action window has additional buttons, each one performs a different action using the values from the window's controls. The additional buttons are between the leftmost and rightmost buttons.

The correct labels to use for an action window's command buttons depend on whether the window is for single-use or multiple-use:

- In a single-use action window, you can label the first button OK—although a more specific label, such as Print, is better. The window's dismissal button must be labeled Cancel.
- In a multiple-use action window, you should assign the first button a meaningful label, such as Replace. The window's dismissal button must be labeled Close, as in Figure 19.

In action windows with two or more buttons that can perform an action, each button needs a unique action-specific label. Do not label a button OK in such a window.




Ensure that each action window has (in left-to-right order) one or more command buttons that perform actions, one dismissal button, and, optionally, a Help button.





Action windows can be single-use or multiple-use. In a single-use action window, ensure that all command buttons (except the Help button) perform their action and then close the window. In a multiple-use action window, ensure that all command buttons except the Close button perform their action and leave the window open.



In a single-use action window, label the dismissal button Cancel. In a multiple-use action window, label the dismissal button Close.

 If users can perform only one action from a single-use action window, label the window's command button OK, or preferably, provide a more specific label. However, if the window has more than one button that can cause an action, do not label any button OK. Instead, provide a more specific label for each button.

 In an action window, ensure that clicking the dismissal button immediately discards all data entered in that window.

 In an action window, ensure that clicking the title bar's close-window control has the same effect as clicking the window's dismissal button.

Window Titles for Identically Named Objects and Views

In applications with multiple windows, each window title should be unique. This section helps you create unique window titles for:

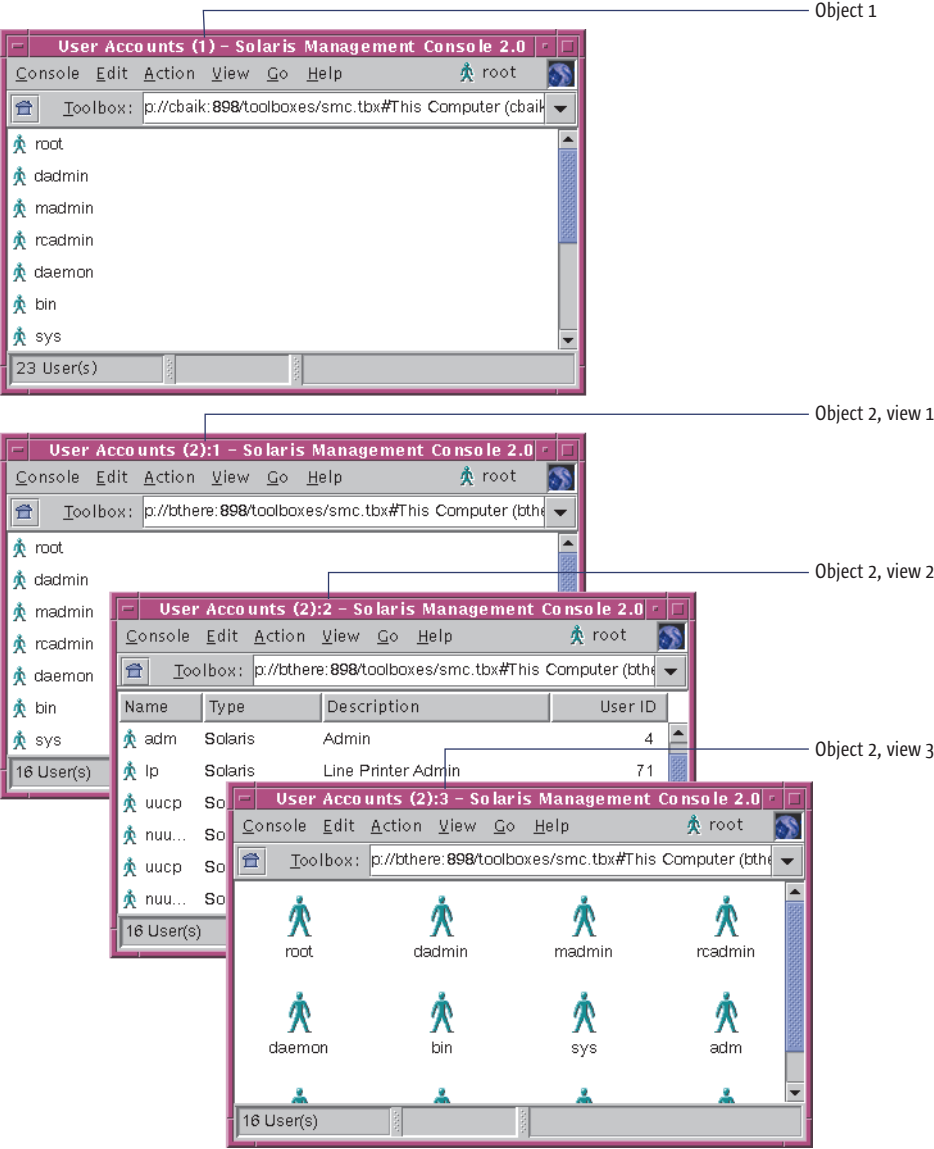
- Multiple windows representing different objects whose names are identical
- Multiple windows representing different views of the same object

The title text of a primary window should be in the following format:

Document or Object Name - Application Name


Figure 20 illustrates the conventions for titling windows that represent identically named objects or multiple views of the same object.

FIGURE 20 Window Titles for Identically Named Objects and for Multiple Views




In primary windows that display identically named objects or views, the window title includes a suffix to the *Object Name*. However, the suffix is not part of the *Object Name*. For example, in Object 2, view 1, of Figure 20, the suffix “(2):1” is not part of “User Accounts”—the *Object Name*.


Window Titles for Identically Named Objects Applications can display more than one window at a time. As a result, different windows can display objects whose names are identical. For example, two windows might each display a different document named `Report`. To avoid confusion and to help users distinguish between such windows, use the following guideline.


 If two or more objects have the same name, make their names unique in window titles by appending a space and the suffix *(n)* to the *Object Name* in each window's title text—where *n* is an increasing integer.


Window Titles for Multiple Views of the Same Object Typically, applications show a particular object in only one window at a time. Sometimes, however, an application needs to show two or more views of the same object in different windows.

For example, at a user's request, an application might show the contents of a folder or directory as a list in one window and as a set of icons in a different window. In such windows, the correct format for the *Object Name* in the title text depends on whether the window is a primary window, as described in the following guidelines.

 If multiple primary windows show views of the same object, distinguish each of the windows by appending the suffix *:n* to the object name in the window title—for example, `Report:2`. (The letter *n* stands for an increasing integer.)

 Do not place a view number in the title bar of property windows.

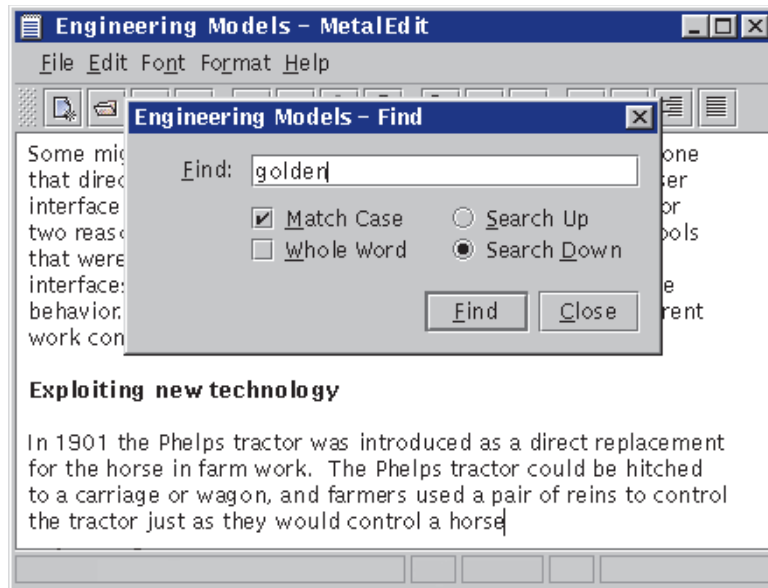
 In action windows, place a view number in the title bar only if the action produces a different result in different views.

 If your application displays multiple objects with the same name and multiple views of the same object, place the view number after the duplicate-name identifier in each window title—for example, `Report (2):3`.

Setting the State of Windows and Objects A typical window or object has properties whose value can change—for example, its screen position and size. For each window or object, the set of current values for all its changeable characteristics is known as its **state**. Applications often need to initialize or restore the state of a window or object. This section provides guidelines related to the state of windows and objects.

Positioning Secondary Windows When displaying a secondary window for the first time, applications should position that window in relation to its parent window, as shown in Figure 21.

FIGURE 21 Secondary Window Correctly Positioned in a Primary Window




In Figure 21, the secondary window is at the **golden mean** of the parent window—a point directly on the parent’s vertical midline and slightly above its horizontal midline. A secondary window centered on its parent’s golden mean is generally considered more visually pleasing than the same window centered on parent’s exact center.

☺ When a secondary window opens for the first time, ensure that it is at the golden mean of its parent window. That is, ensure that the secondary window is both:


- Centered on the vertical midline of its parent window
- Positioned so that its top is n pixels below the top of the parent window

The value of n can be derived from the following equation, in which h is the height of the parent window:


$$n = h - \left(\frac{h}{1.618} \right)$$


 When closing and reopening a secondary window during a single application session, reopen that window where it was when it closed most recently. (Alert boxes are an exception. Always reopen an alert box at its initial position.)

Restoring the State of Property Windows Users can change a property window's state by several means—for example, by rearranging the window's tabbed panes or other organizing elements.

 If a user reopens a property window after closing it during the same application session, restore that window's state. In other words, make the window look exactly as it did when the user last closed it—especially if the user has manipulated the window's components.

Alerting Users After an Object's State Changes In some applications, an object's state can change outside a user's control—even while the user is editing the object's property window.

 If the state of an object changes while a user is editing that object, display an alert box if the user tries to apply changes to the object. In the alert box, inform the user that the object's state has changed.

 In alert boxes for informing users of object-state changes, provide text at least as specific as this:

Object Name changed while you were editing it. To update it with the values you entered, click Update. To discard the values you entered and use the new ones instead, click Discard. To cancel your attempt to edit this object, click Cancel.

Replace *Object Name* with the actual name of the changed object. If your application can detect how the object changed, display text that describes the change precisely.

Multiple Document Interfaces Avoid designing a Multiple Document Interface (MDI) application—an application in which primary windows are represented as **internal windows** inside a **backing window**.

Many users have trouble manipulating the windows of MDI applications. In addition, an MDI application's main window can sometimes obscure a user's view of other applications.

 Use no internal windows in your application.