4 Making and Managing Connections

It could be said of me that in this book I have only made up a bunch of other men's flowers, providing of my own only the string that binds them together.

Montaigne, *Essais*

Let him look to his bond.

Shakespeare, *Merchant of Venice*

Communicating With Other Objects: Outlets and Actions

Outlets

An outlet is an instance variable that points to another object. Objects use outlets to communicate with other objects; they simply send messages to the object identified by the outlet.

Using Interface Builder, you can declare and set outlets for the custom objects in your application. You can also set ready-made outlets in many Application Kit objects, such as browsers. Once initialized, the connection information for the outlet is stored in the nib file. At run time, the nib file is unarchived and the outlet is reinitialized with the connection information.

The Application Kit defines two types of outlets that you can use to establish specialized connections with other objects: delegates and targets.

Delegates

A delegate is an object that acts on behalf of another object. Many Application Kit classes define delegate outlets as an alternative to subclassing. All your object must do is register itself as a delegate of the Application Kit object. At important junctures in its life cycle, the Application Kit object sends messages to its delegate, giving it an opportunity to participate in processing and sometimes the chance to veto behavior.

For example, browsers ask their delegates to supply the cells for browser columns, and the application informs its delegate when it is initialized, hidden, and activated.

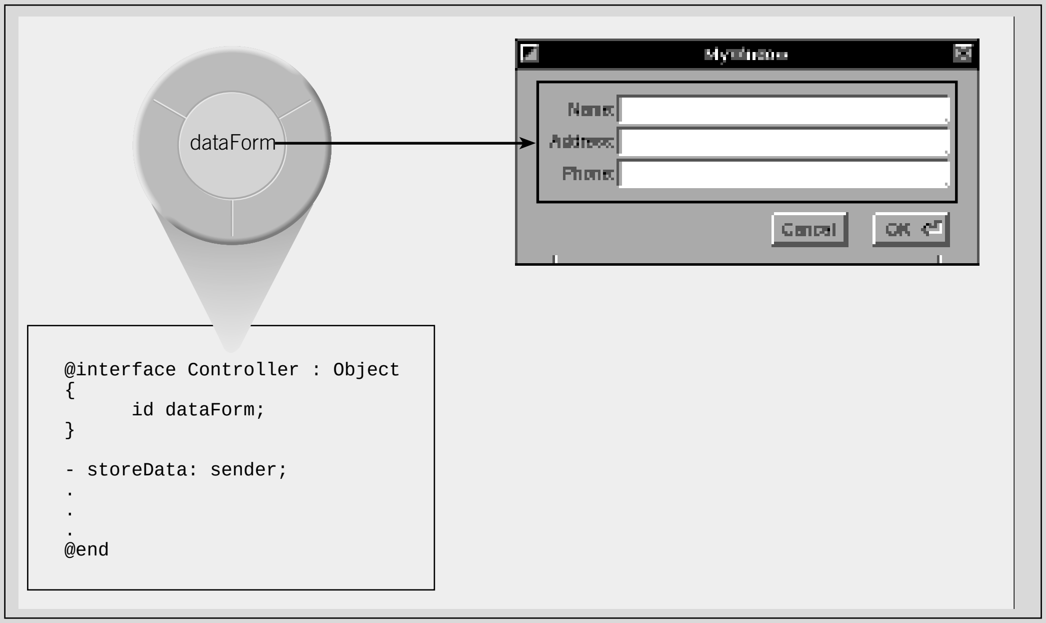
Targets

Targets are a special kind of outlet. They identify objects that can respond to action messages. When a user activates an NSControl object (for instance, clicking a button or moving a slider), that object sends an action message to the target. The action message gives application-specific meaning to the original mouse or key event. For example, you could connect a custom object in your application as the target of a button so that when the button is clicked, your object performs a method that fills all of the text fields in the window with appropriate information.

Like a delegate, a target must implement methods to respond to the messages it's sent. But unlike a delegate, which receives messages chosen from a limited set defined by another object, a target responds to any action message you choose to define.

You can also make one object a target of a second object programmatically by sending **setTarget:** to the second object.

Outlet



```
@interface Controller : Object
{
      id dataForm;
}

- storeData: sender;
.
.
.
@end
```

Actions

When a user manipulates an NSControl object, the object receives an event message, which it translates into a message that is meaningful within the application. It then send this message to another object. These application-specific messages initiated by an NSControl object are called *action messages*, and the methods they invoke are called *action methods*.

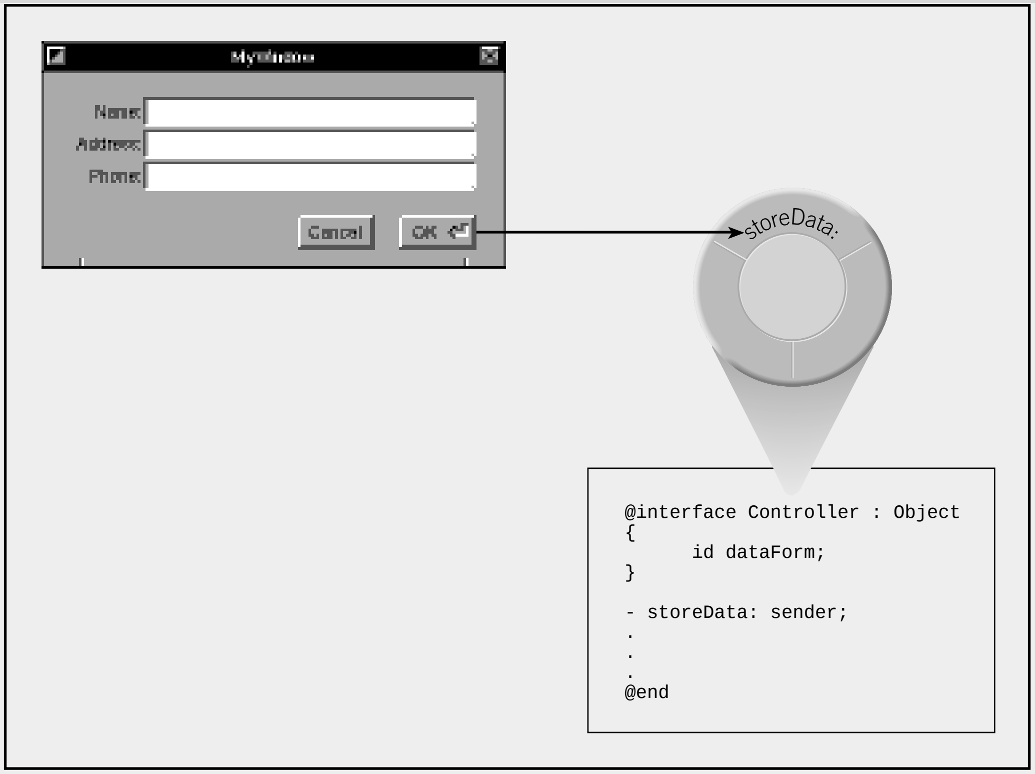
NSControl, an abstract class, defines for its many subclasses (such as NSButton, NSScroller, NSTextField, and NSForm) a paradigm for inter-object communication—action messages. But NSControl objects don't act alone: they always contain one or more NSActionCells (or one of its subclasses). The NSActionCell superclass defines instance variables for the two elements essential to an action message:

- **target** the object that's responsible for responding to the user's action
- **action** the method that specifies what the target is to do

Action methods take a single argument, the **id** of the NSControl object that sends the message. This argument enables the receiver to ask the control for more information, if it's needed.

An NSControl can send a different action message to a different target for each NSActionCell it contains. Different NSControls dispatch action messages differently; for instance, an NSButton generally sends action messages on a mouse-up event, but an NSSlider usually sends action messages continuously, as long as the mouse button is pressed.

Action



```
@interface Controller : Object
{
      id dataForm;
}

- storeData: sender;
.
.
.
@end
```

Connecting objects

1 **Select an object.**

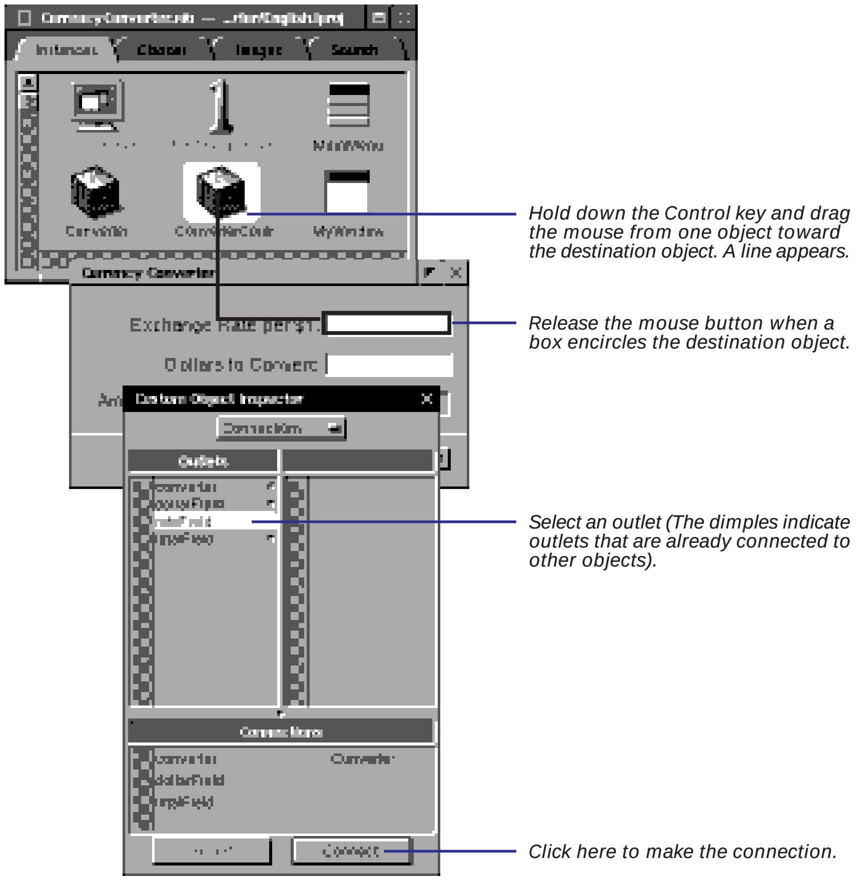
2 **Control-drag a connection to another object.**

3 **In the Inspector panel's Connections display, select an outlet or action.**

4 **Click the Connect button.**

In an object-oriented application, isolated objects have little value; they need to send messages to each other to get the work of the application done. Interface Builder gives you a way to establish connections between objects.

When you Control-drag between two objects, the Inspector panel becomes the key window. Its Connections display shows the current and potential connections for the destination object.

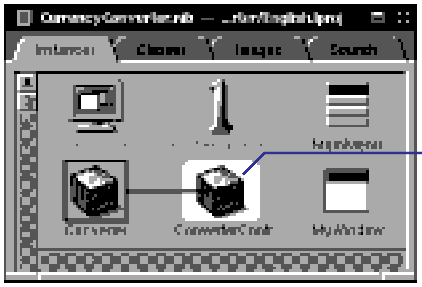

Instances
Classes
Images
Sounds
MainMenu
ConvWin
ConverterCode
MyWindow
Currency Converter
Exchange Rate per $1:
Dollars to Convert:
Custom Object Inspector
Connections
Outlets
converter
rateField
Connections
Connect
Hold down the Control key and drag the mouse from one object toward the destination object. A line appears.
Release the mouse button when a box encircles the destination object.
Select an outlet (The dimples indicate outlets that are already connected to other objects).
Click here to make the connection.

If the Connect button doesn't become active when you select an outlet or action, you probably have connections locked. See "When You Don't Want to Disconnect" in this chapter.

Outlet Connections

In the previous example, the connection is made from a *controller* object—a custom object that manages the application—to a text field. The controller object (ConverterController) declares several *outlets*—identifiers of destination objects—as instance variables.

The example shows a connection between an object in the nib file window Instances display and an object in the interface. You can also make outlet connections between two objects in the Instances display.



Control-drag a connection line and release the mouse button when a box appears around the destination object.

When you make a connection between objects, the first column of the Connections display shows the source object's outlets ("source" meaning the object from which a connection line is drawn).

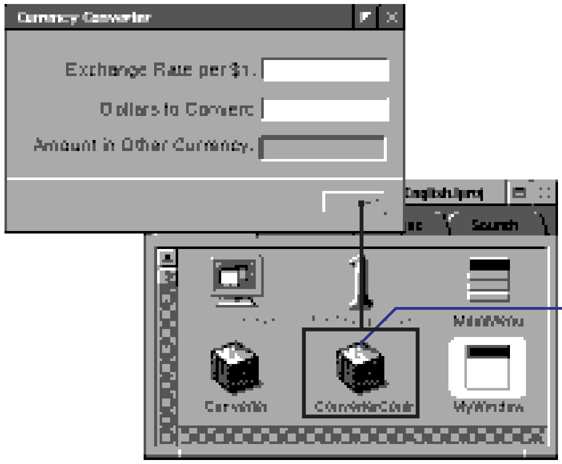
Action Connections

When you make a connection by dragging a line *from* an NSControl object in the interface—a button, slider, text field, menu command, pop-up list, or matrix—odds are that the destination object is a *target*

and that you can complete the connection by selecting an *action* method.

Outlets are destination objects specified as instance variables. Actions are methods that NSControl objects (such as buttons) invoke in another object. See "Communicating With Other Objects: Outlets and Actions" in this chapter for more information.

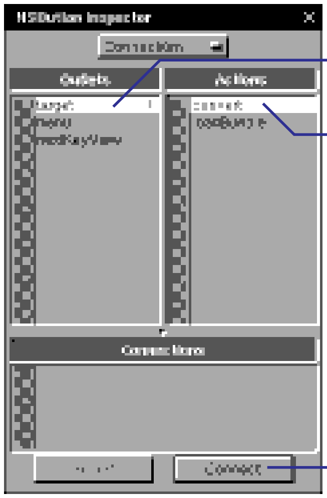
Chapter 6, "Subclassing," describes connecting the outlets and actions of custom objects in the context of creating a class.



To make a connection involving an action message, Control-drag a line from an NSControl object to the object that responds to the message you want the NSControl to perform.

The destination object in an action connection is frequently a custom object that manages the application or a particular window (controller object).

When you make a connection from an NSControl object, the Inspector panel shows the Connections display for the destination object.



Click here to display actions currently defined for the target object. Actions appear listed under the second column.

Click to select an action.

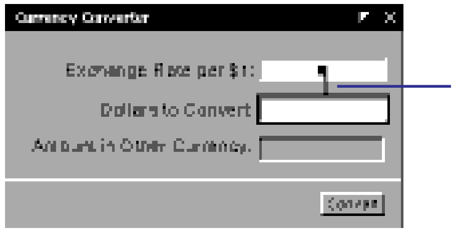
Click to make the connection.

See “Compound Objects” in Chapter 3 for descriptions of the interaction between NSControl objects and NSCell objects, and of the role NSMatrix objects play.

When the user manipulates the NSControl object, such as clicking a button or moving a slider, the action message is sent to the destination object (the target).

Connections Within the Interface

Sometimes you can connect two objects on an interface. These connections can involve both outlets and actions.



Control-drag a connection line from one object to another, then release the mouse button.

Connections within an interface can also involve two Application Kit objects. Two examples are interconnecting text fields (so the user can tab from field to field), and connecting a menu command such as Print to an NSText object.

Tip: To enable printing of an NSText object, drag a connection line from the Print menu command (or other NSControl object that initiates printing) and select the **print:** action in the Connections display.

You can connect text fields and form fields so that when the user presses the Tab key, the cursor moves to another field. See “Enabling inter-field tabbing” in this chapter for information on this procedure.

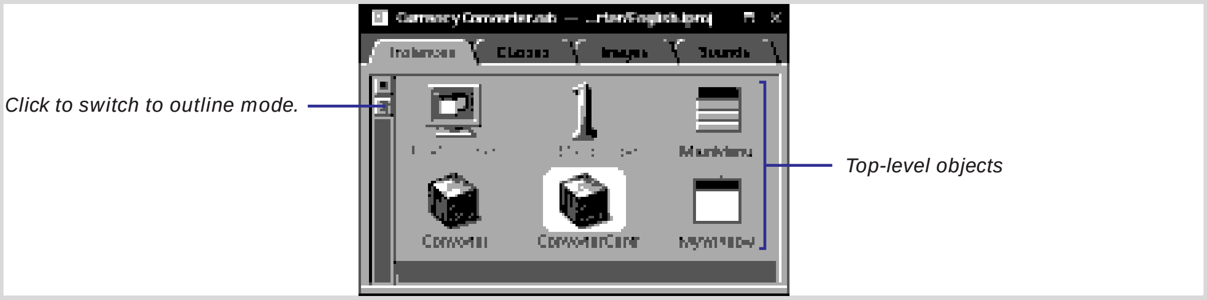
The Modes of the Instances Display

When you open a nib file in Interface Builder, the Instances display of the nib file window first shows objects as icons. This icon mode doesn't show all objects, just the *top-level objects*—those objects that are not contained by another object. Windows and panels and most controller objects (that is, objects that manage an application or a window) are top-level objects; although they may contain other objects (for instance, a window contains one or more views), no other object contains them.

The graphical representation of objects in icon mode makes it an ideal interface for many operations. Its simple, intuitive, and uncluttered nature makes it easy to do the basic things, such as making connections between top-level and interface objects.

For more complex operations, the Instances display has another mode—outline mode—that shows more detail about objects in the nib file, including their connections with each other.

Icon Mode



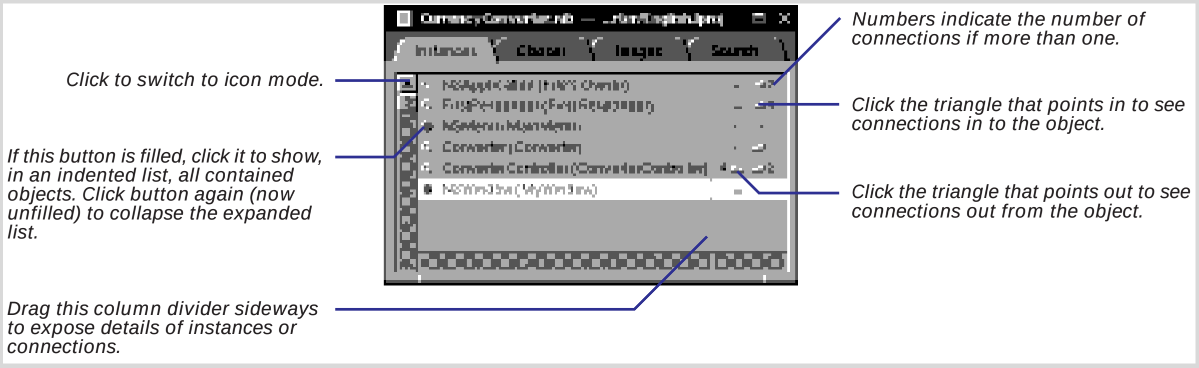
The most important advantage of the outline mode is that it shows *all* objects in the nib file, not merely the top-level objects. It also shows all connections, both connections into an object and connections from an object to other objects.

The outline mode starts by listing the top-level objects in the nib file. By clicking the open button next to an instance, you can see what other objects it contains. Click a connection button (triangle button) to see what connections go into or out of an object.

You can connect objects in outline mode; there's no need to drag a connection line to the interface. Outline mode also has facilities that make it easy to identify objects in the interface and to disconnect objects.

Objects in outline mode are identified first by class name and then, in parentheses, by title. If the title is obscured, you can resize the nib file window until it is visible.

Outline Mode

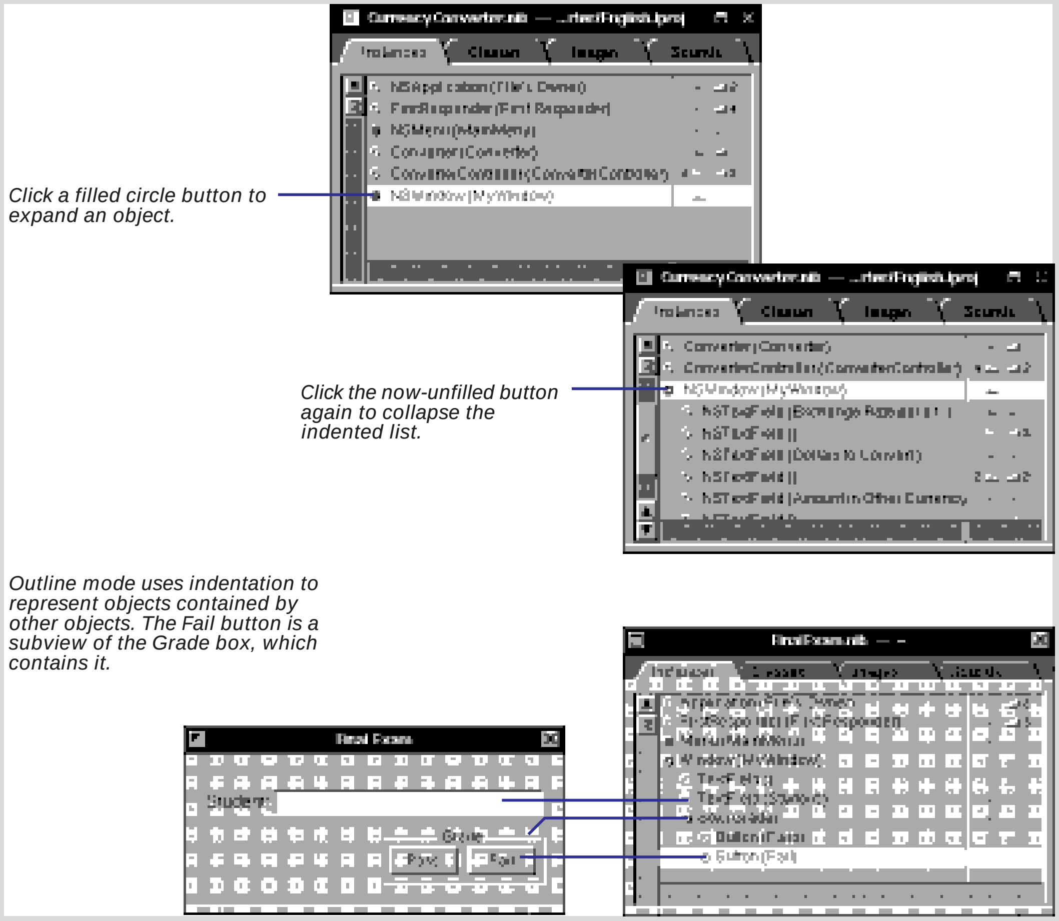

Click to switch to icon mode.
If this button is filled, click it to show, in an indented list, all contained objects. Click button again (now unfilled) to collapse the expanded list.
Drag this column divider sideways to expose details of instances or connections.
Numbers indicate the number of connections if more than one.
Click the triangle that points in to see connections in to the object.
Click the triangle that points out to see connections out from the object.
Instances
Classes
Images
Sounds

Expanding Objects in Outline Mode

In outline mode, objects that contain other objects have a small circle button to their left that is filled with gray. The subordinate objects are usually subviews of a window, panel, or another view object, but can be objects that are part of another object not visible on the screen. You display these contained objects by expanding the container object.

Click a circle button to expand an object into a list of its component objects; click it again to collapse the list. Expansions can be nested many levels. To expand everything within an object, Command-click the circle button. Collapse the list back to the original level by Command-clicking the circle button again.

See “The View Hierarchy” in this chapter for a description of the relationship between superview and subview.


CurrencyConverter.nib — ...rter/English.lproj
Instances
Classes
Images
Sounds
NSApplication (File's Owner)
FirstResponder (First Responder)
NSMenu (MainMenu)
Converter (Converter)
ConverterController (ConverterController)
NSWindow (MyWindow)
NSTextField (Exchange Rate ...)
NSTextField ()
NSTextField (Dollars to Convert)
NSTextField ()
NSTextField (Amount in Other Currency)
FinalExam.nib
Final Exam
Student:
Grade
Pass
Fail
Button (Fail)

Click a filled circle button to expand an object.

Click the now-unfilled button again to collapse the indented list.

Outline mode uses indentation to represent objects contained by other objects. The Fail button is a subview of the Grade box, which contains it.

Making connections in outline mode

1 **Select an object.**

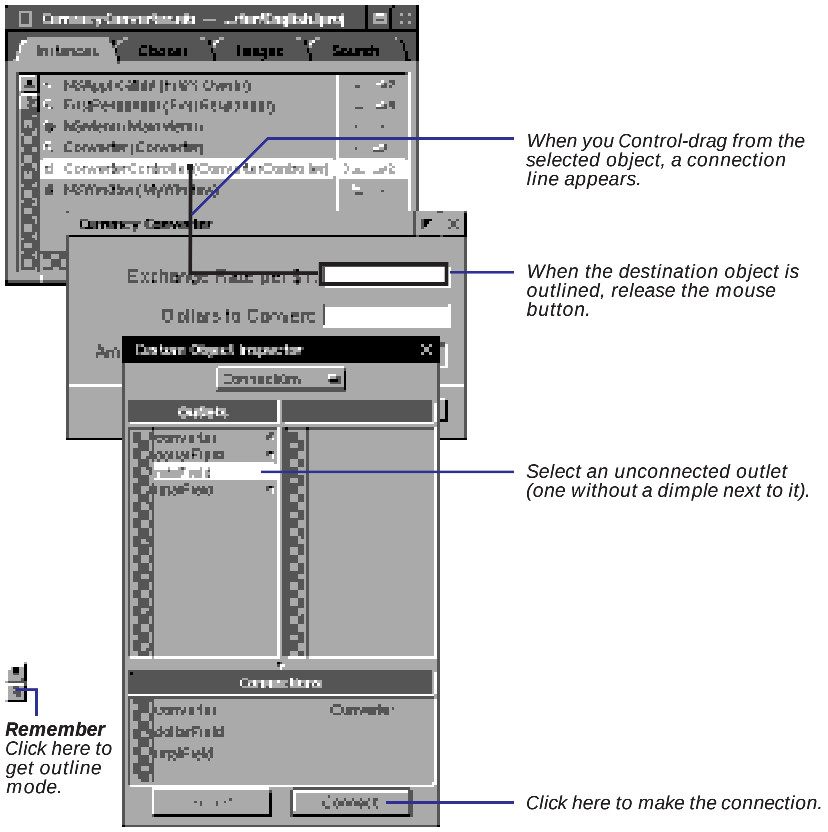
2 **Control-drag a connection to another object.**

3 **In the Inspector panel's Connections display, select an outlet or action.**

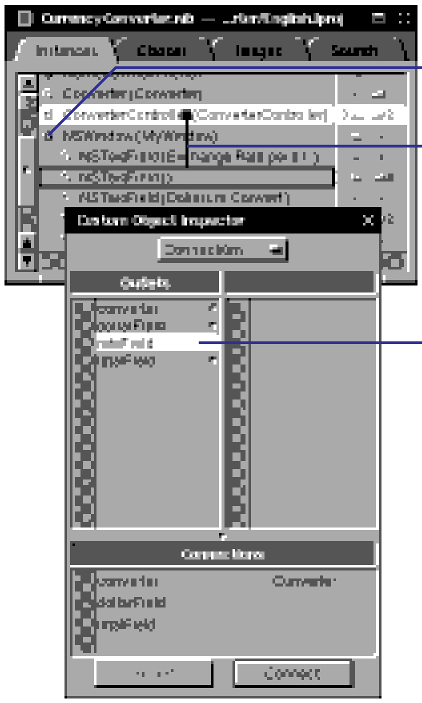
4 **Click the Connect button.**

You can make connections between objects in the outline mode of the Instances display as well as its icon mode. The connections can be between an object in the outline and an object in the interface or between two objects listed in the outline.

Before you make a connection involving an object in outline mode, make sure that the object is visible in the display. (You might have to expand the object's "parents" in outline mode to do this.)


Instances
Classes
Images
Sounds
Currency Converter
Exchange Rate per $1:
Dollars to Convert:
Custom Object Inspector
Connections
Outlets
Connect
When you Control-drag from the selected object, a connection line appears.
When the destination object is outlined, release the mouse button.
Select an unconnected outlet (one without a dimple next to it).
***Remember** Click here to get outline mode.*
Click here to make the connection.

The outline mode offers a useful capability for making connections without leaving the nib file window. In this example, the same connection is made as in the previous example.



If necessary, expand the object's parent so that you can see both objects.

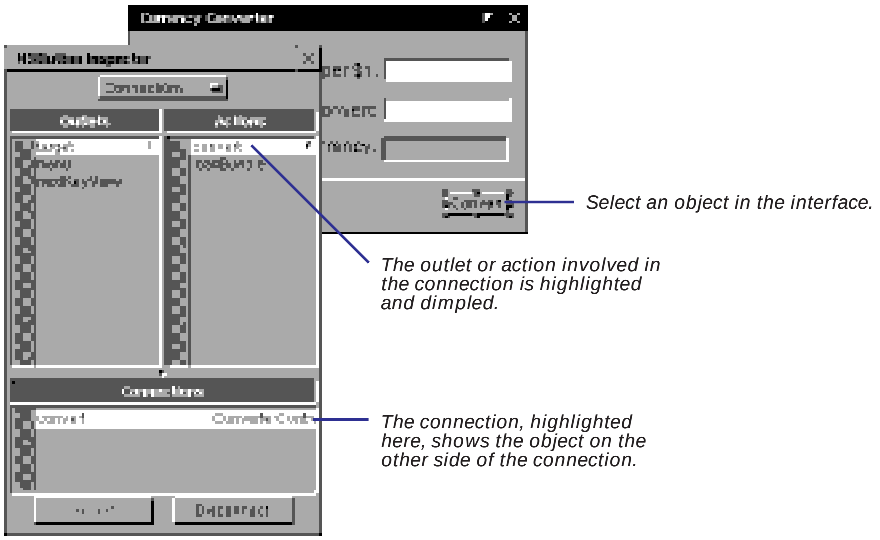
Control-drag a connection line between two objects.

Complete the connection as before.

Examining connections

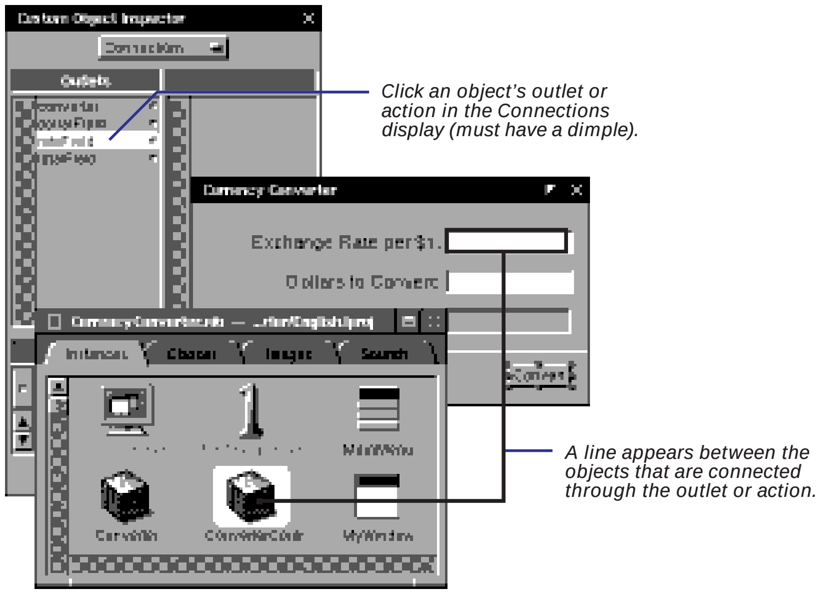
- **In the interface:**
  Select an object and look at the Connections display of the Inspector panel.
- **In the Instances display:**
  Select an object and look at the Connections display of the Inspector panel.
- **In the Connections display:**
  Click a dimpled outlet to see the connection line drawn.
- **In outline mode:**
  Click a triangle button in the column to the right of an object.

Interface Builder gives you many ways to examine and verify connections between objects. It makes it easy, for example, to discover what outlets and actions are associated with an object in the interface.

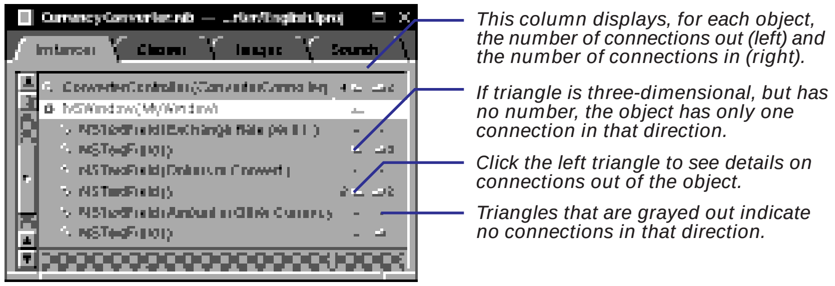

Currency Converter
NSButton Inspector
Connections
Outlets
Actions
target
menu
nextKeyView
convert:
Connections
convert
ConverterCont
Disconnect
Select an object in the interface.
The outlet or action involved in the connection is highlighted and dimpled.
The connection, highlighted here, shows the object on the other side of the connection.

You can also select an object in the Instances display (in both icon and outline modes) and examine the Inspector panel as described above to find out what object it is connected to.

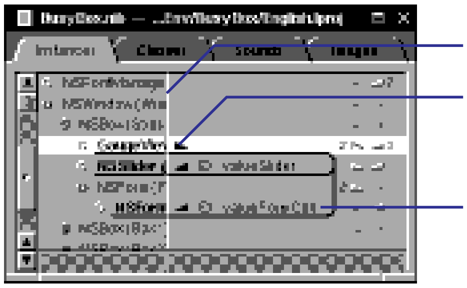
You can also examine object connections going in the other direction too, from the Connections display to the interface and the Instances display.



The Connections display allows you to see one connection at a time. The outline mode of the Instances display shows you *all* connections an object has, both connections into the object and connections from that object to other objects.


This column displays, for each object, the number of connections out (left) and the number of connections in (right).
If triangle is three-dimensional, but has no number, the object has only one connection in that direction.
Click the left triangle to see details on connections out of the object.
Triangles that are grayed out indicate no connections in that direction.
Instances
Classes
Images
Sounds

When you click a three-dimensional triangle, lines appear to show the connections between objects. The name and class of each connected object is highlighted in bold. Each connection is labelled with the name of an outlet or action.



To see more of a column, drag the column divider sideways.

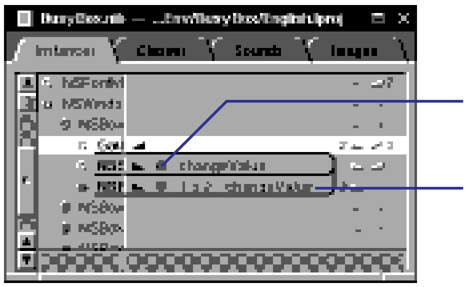
The right-pointing triangle indicates connection-out. Lines show you where the connections lead to.

The left-pointing triangle indicates connection-in. The electrical outlet icon represents an outlet;the name of the outlet follows.

⊕ *Indicates action.*

⊡ *Indicates outlet.*

Note that an object may have multiple connections with another object, both in and out, both outlets and actions. In these cases, the outline mode lets you toggle between the connections.



The cross-hairs icon represents a connection involving an action.

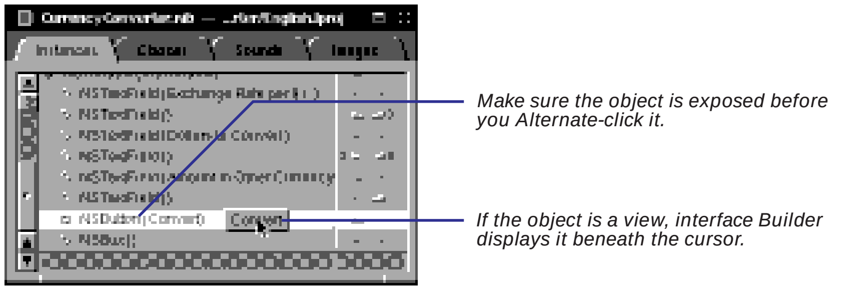
Connections with colon-separated numbers indicate multiple connections (here it means "1 of 2"). Click the colon to toggle between the connections.

To make the connection lines disappear, click the triangle button again.

Identifying objects in outline mode

- **To see a representation of an object, Alternate-click it in outline mode of the Instances display.**
- **To have an arrow point at the interface object, Control-Shift-click the object in outline mode.**

In the outline mode of the Instances display, you might want to verify what an object is before connecting it to another object. You have two graphical ways to identify an interface object. One technique displays an image representing the selected object.

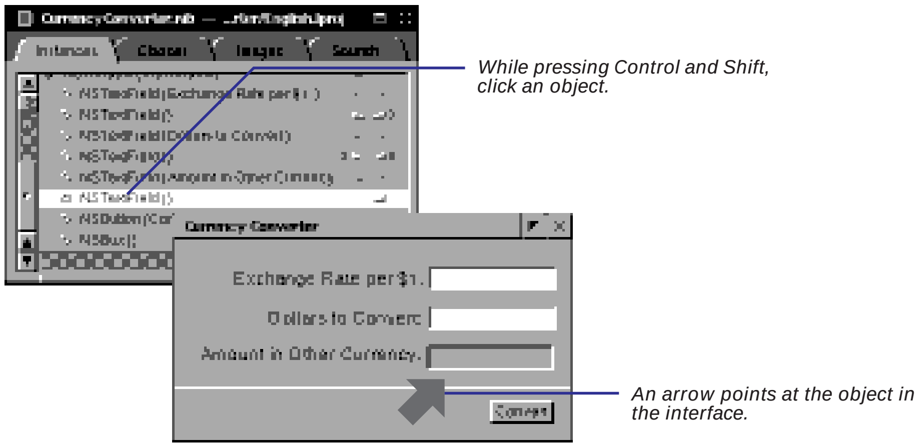


Make sure the object is exposed before you Alternate-click it.

If the object is a view, interface Builder displays it beneath the cursor.

When you Alternate-click non-view objects in outline mode, the images that represent them in icon mode are displayed (cubes for custom objects, mini-windows for panels and windows). The File’s Owner, First Responder, and Main Menu objects don’t display icons.

The second technique locates an object in the interface with a large arrow.



While pressing Control and Shift, click an object.

An arrow points at the object in the interface.

See “The Modes of the Instances Display” in this chapter for an introduction to outline mode.

Control-Shift-Clicking the File’s Owner, First Responder, and Main Menu objects has no effect.

Standard Objects in the Instances Display: File's Owner, First Responder, and Font Manager

File's Owner

Every nib file has one owner, represented by the File's Owner icon. The owner is an object, external to the nib file, that channels messages between the objects unarchived from the nib file and the other objects in your application.

Not only must the owning object be external to its nib file, it must exist before the nib file is unarchived. This is because the same method that loads a nib file (**loadNibNamed:owner:** and its variants) also specifies the file's owner.

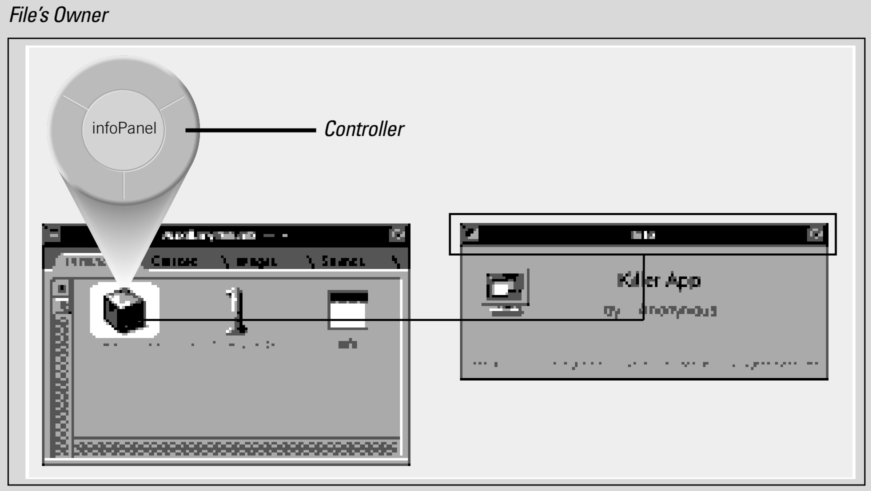
The typical owner of an auxiliary nib file (such as one containing an Info panel) is an instance of the class you assign to File's Owner in Interface Builder. This class is almost always a custom class, and is frequently the class of the object that manages your application. Once you make the assignment, File's Owner serves as a proxy instance of your class, which you can then connect to the interface. (By the way, the typical owner of an application's main nib file is NSApp, the global NSApplication object.)

See Chapter 11, "Dynamic Loading," for more on the role of File's Owner in the loading of auxiliary nib files and for details on assigning classes to File's Owner.

First Responder

The First Responder is the object within a window that first receives keyboard events, mouse-moved events, and action messages from NSControl objects that don't have an explicit target (for example, cut and paste). The First Responder object is the active window's focus for future events. Although technically an object, First Responder is really a status conferred on an object.

Usually, when you click an object that accepts key events (such as a text field), that object becomes the window's First Responder. First Responder status also changes when you make another window key in your application. (Because of this, First Responder can be useful when you build multiple-document applications.) Over time, many different objects can become the First Responder, but at any one time only one object has this status. The First Responder icon stands for the object that currently has this status, no matter which actual object it is within your application.



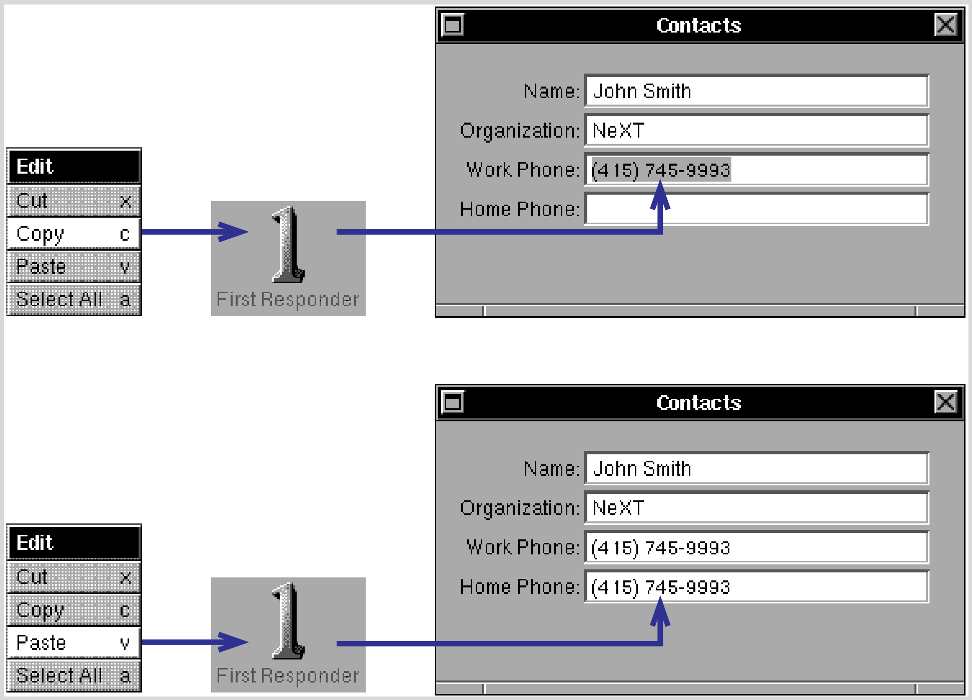
The First Responder figures into the event-handling behavior defined by the NSResponder class. In a window, objects inheriting from NSResponder (including NSView, NSApplication, and NSWindow) are part of a linked list of event-handling objects called a *responder chain*. The responder chain contains (in this general order) a view, the view's superview, the view's window, the main window, and then the application. (The application and window delegates are in this chain as well, although they aren't NSResponders.) If the First Responder can't respond to an event message, its next responder is given a chance to respond. If an NSResponder can't handle the message, the message continues to be passed up the chain from object to object in search of an NSResponder that can. Messages are passed in one direction only: up the view hierarchy toward the window and application.

In Interface Builder you can connect an NSControl object in the interface to the First Responder icon. Thereafter, when the user manipulates this NSControl (say, by clicking a menu item entitled Copy) an action message (**copy:**) is sent to the object that is currently First Responder. If you examine in Interface Builder the default connections from the Edit menu, you'll discover that its menu cells are all connected to First Responder.

Font Manager

The Font Manager icon represents an instance of the NSFontManager class that is shared among the objects of an application. Interface Builder automatically creates and adds this object to your project when you drag the Font menu into your application's menu. The Font Manager is the center of activity for font conversion. It accepts messages from font conversion user-interface objects (such as the Font menu or the Font panel) and appropriately converts the current font in the selection by sending a **changeFont:** message up the responder chain. See the documentation on the NSFontManager class for more information.

First Responder



Enabling inter-field tabbing

1 **Control-drag between the window and a view object.**

2 **In the Connections display of the Inspector panel, select initialFirstResponder and click Connect.**

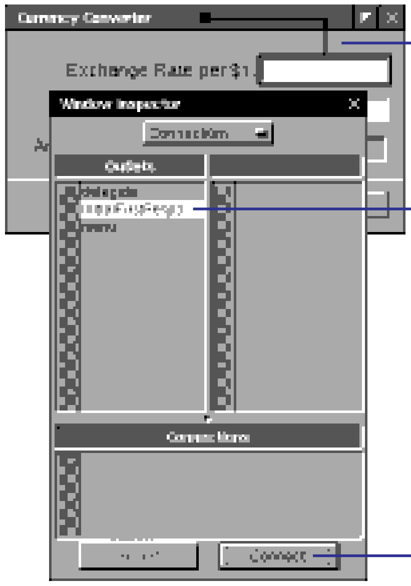
3 **Control-drag between two view objects.**

4 **In the Connections display, select nextKeyView and click Connect.**

In OPENSTEP applications, users can navigate between fields and controls on the interface solely through use of the keyboard. Users can change the first responder by pressing the Tab key or Shift-Tab, can navigate through cells in a matrix by pressing the arrow key, and can change the state of a button or select a cell in a matrix by pressing the Spacebar.

You get most of this keyboard navigation feature in your application for free; you don't have to do anything special to allow users to navigate between cells in a matrix or fields in a form. However, you'll want to control what the Tab key does, that is, which view the cursor should go to next when the user presses the Tab key. You do this by connecting NSView objects to each other through the **nextKeyView** outlet.

First, decide which view should respond to keyboard events when the window becomes key, and connect the NSWindow **initialFirstResponder** outlet to that view.

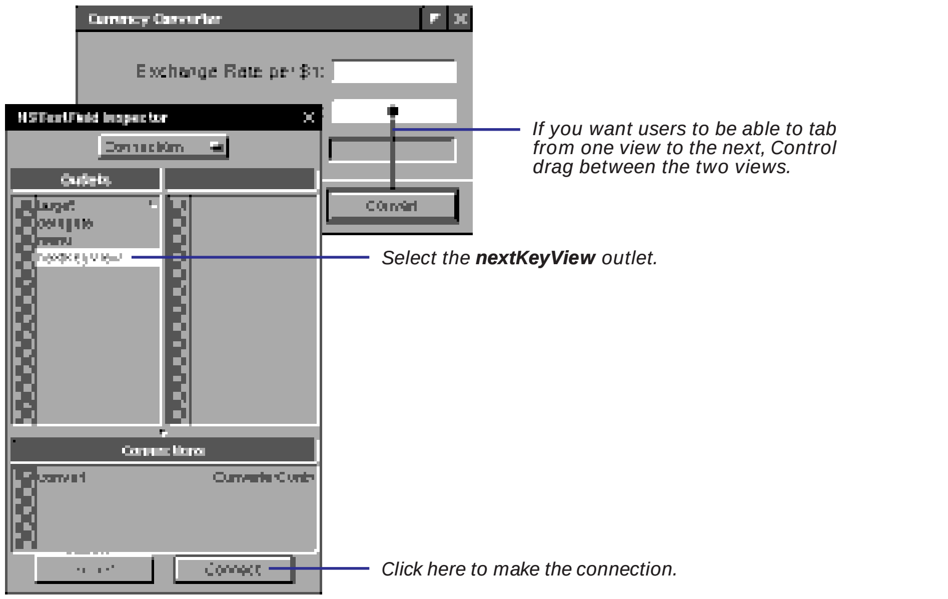


Control-drag a connection from the window to the object in the window that should initially take keyboard events.

*Select the **initialFirstResponder** outlet.*

Click here to make the connection.

Next, use NSView's **nextKeyView** outlet to connect view objects to each other.



If you want users to be able to tab from one view to the next, Control drag between the two views.

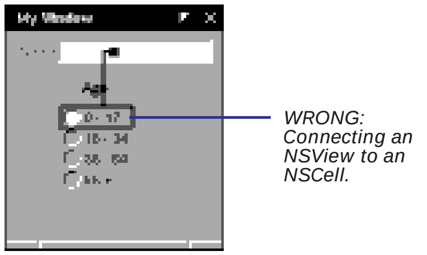
*Select the **nextKeyView** outlet.*

Click here to make the connection.

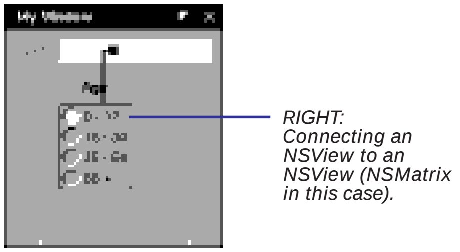
Don't connect views that the user cannot select or edit. In the example above, we skip over the gray text field because it exists to show the result of the Convert button's action. The user cannot enter text into this text field, so it does not make sense to make a **nextKeyView** connection to it. You also should be careful not to connect to NSCell objects. For example, you shouldn't connect to an individual cell of a matrix or form; instead hook the preceding object to the entire matrix or form. The NSMatrix and NSForm objects determine the keyboard navigation between their own cells.

You should also assign key equivalents to buttons. The default button typically has a Return key equivalent, and the Cancel button typically has the Esc key equivalent. See Chapter 3 for more information.

If you don't make **nextKeyView** connections, default connections are made at run time. You can use Interface Builder's Test Interface command to see if these connections are satisfactory. See "Testing the interface" in this chapter.



WRONG: Connecting an NSView to an NSCell.



RIGHT: Connecting an NSView to an NSView (NSMatrix in this case).

Disconnecting objects

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

2 **In the Connections display of the Inspector panel, select a connection.**

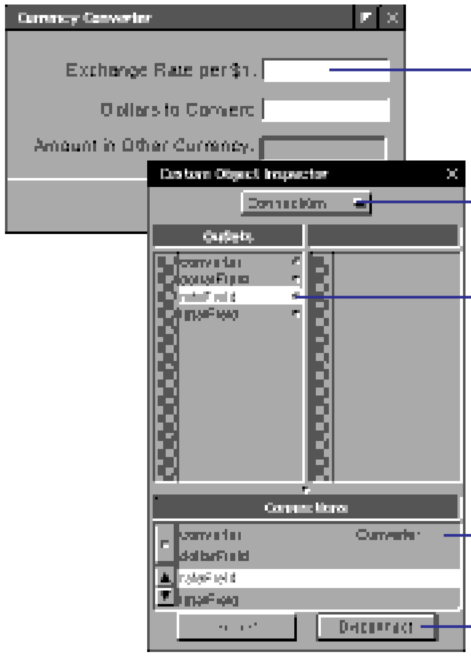
3 **Click Disconnect.**

Or

1 **In the nib file window's outline mode, click a triangle button to display a connection.**

2 **Control-click the connection line.**

Interface Builder gives you two ways to break the connections between objects. The first method uses the Inspector panel.



Make sure a single object is selected.

If the Inspector is not already displayed, choose Inspector from the Tools menu and choose Connections here.

Select an outlet or action with a dimple next to it (indicating a connection).

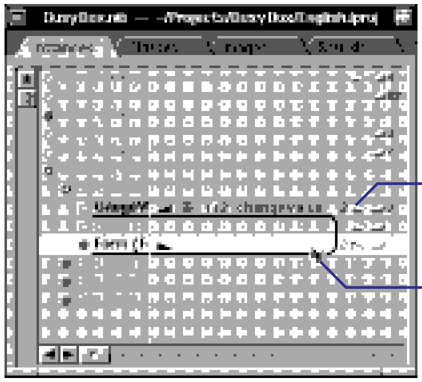
Verify the connection before you break it (the item on the right is the object on the other side of the connection).

Click here to break the connection.

You can also initiate this procedure by selecting objects in icon mode of the Instances display, and then disconnecting them in the Inspector panel as above.

See "Examining connections" in this chapter to learn how to use outline mode to display the connections between objects.

The alternative method for disconnecting objects allows you to perform the operation in one place: in the outline mode of the nib file window's Instances display. First show connections for an object by clicking a three-dimensional triangle button.


Click to show the connections for an object (left triangle for connections out, right triangle for connections in).
Control-click a connection line to server connection.

You must Control-click on the *right* side of the column divider (nearest the connection-out and connection-in triangle buttons) to get the scissors to appear, and thus be able to break the connection. When you Control-click on the *left* side of the column divider, it begins a connection operation.

When You Don't Want to Disconnect

After all of the objects in your interface are connected the way you want them, you may want to make sure that they stay that way. When you delete an object from the interface, all of the connections to that object are broken. If all you're doing is fine-tuning the interface's appearance, you want to make sure this doesn't happen.

To prevent someone from accidentally changing connections, set the Lock all connections preference on the General preferences panel display. (Choose Preferences from the Info menu to bring up the Preferences panel.) When this preference is set, you can't connect objects, disconnect objects, or delete objects that have connections.

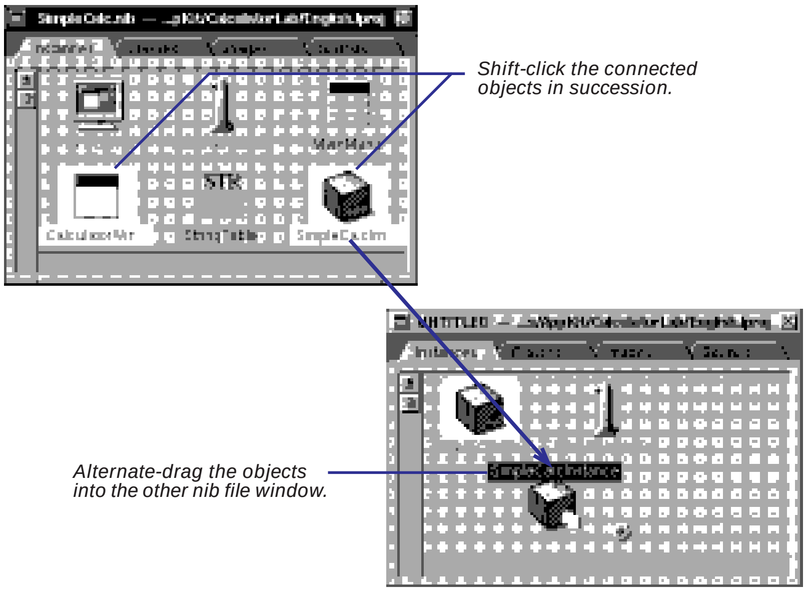
When you're localizing an application, it's a good time to use this connection locking feature. When you localize a nib file, you want the interface objects to behave the same way, but you want their titles to change. Sometimes, it's necessary to move and resize the interface objects to make room for titles in other languages that tend to have longer words. By locking connections, you make sure that you don't make a change to the interface that will change the way the application behaves.

Copying interconnected objects

1 **Select the objects that are connected.**

2 **Alternate-drag the objects into another nib file window or onto another window or panel.**

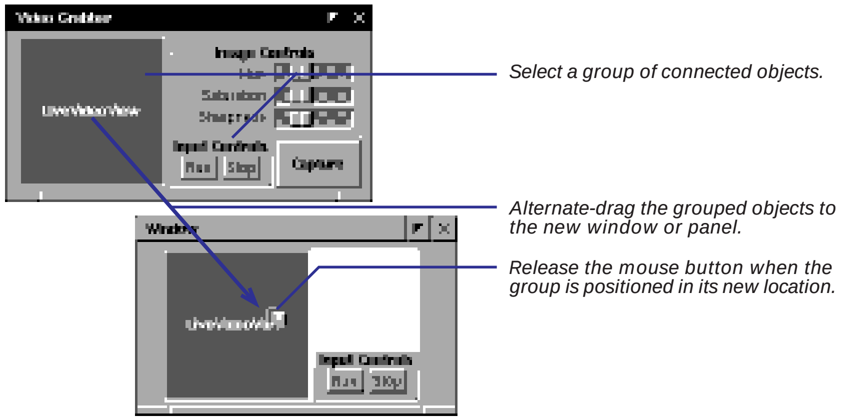
You can easily copy objects—with their connections— between nib files. You'll probably use this feature most often to copy a window and its views along with the custom object that manages those views.


Shift-click the connected objects in succession.
Alternate-drag the objects into the other nib file window.

Notice the icon representing the copied objects in the example above. Under the cursor is the icon representing the object that is actually dragged. The plus sign indicates that more than one object is involved in the operation. When the copying process completes, the new nib file window holds duplicates of the objects that include their connections to each other.

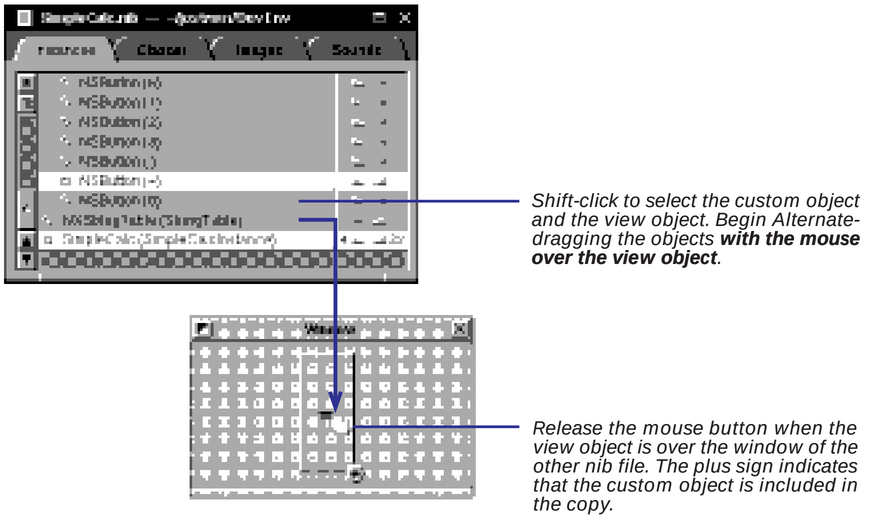
You can use the same basic technique to copy connected objects on an interface. In the next example, an instance of an NSView subclass is connected to the Run and Stop buttons. You can copy these objects and their connections by Alternate-dragging them onto a window in another nib file.

The various scenarios for copying objects and their connections between nib files is quite similar to the procedures for copying objects to dynamic palettes. See Chapter 5, "Using Dynamic Palettes," for more information on this Interface Builder feature.


Video Grabber
Image Controls
Input Controls
Capture
LiveVideoView
Window
Select a group of connected objects.
Alternate-drag the grouped objects to the new window or panel.
Release the mouse button when the group is positioned in its new location.

Another occasion for copying connected interface objects is when you want to make copies of text fields or form fields and preserve the connections between fields.

From the outline mode of the Instances display, you can copy an individual view object, a custom non-view object, and the connections between the two.


Shift-click to select the custom object and the view object. Begin Alternate-dragging the objects **with the mouse over the view object**.
Release the mouse button when the view object is over the window of the other nib file. The plus sign indicates that the custom object is included in the copy.

You can also copy interconnected interface objects to another window in the same nib file. See “Moving objects to other windows” in Chapter 2.

Testing the interface

1 **Choose Test Interface from the Document menu.**

2 **Check the functioning of OpenStep objects.**

3 **Choose Quit from the application menu or double-click the switch icon in the application dock.**

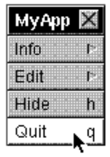
After you create an interface, Interface Builder lets you see how it works from the user's perspective.

Interface Builder's menu, windows, and panels disappear, leaving only the actual interface and (if you are testing the application's main nib file) the main menu. Give your interface a test ride. Here's some of the things you might try:

- Verify that the cursor moves from field to field when you press Tab.
- Verify that you can copy, cut, and paste text (First Responder actions).
- See if you can print (the Print menu item must be connected to an appropriate view object's **print:** action method).

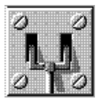
Note: When you test your interface, the behavior provided by your custom classes is not called into play (with the exception of static, compiled palette objects). You can only test the behavior that OpenStep and static palette objects exhibit in themselves and when they send messages to each other. To test all components of your application, you must compile and run it.

When you are finished testing the interface, exit from test mode.



If testing the mainnib file:

Click here to end test mode and return to Interface Builder.



If testing an auxiliarynib file:

Double-click the test mode icon in the application dock to exit test mode.

The View Hierarchy

When you expand an NSWindow object in outline mode and then expand the NSView objects indented beneath, you are looking at a *view hierarchy*. All the NSView objects within a window are linked together in this hierarchy, an abstract tree structure similar to the class inheritance hierarchy.

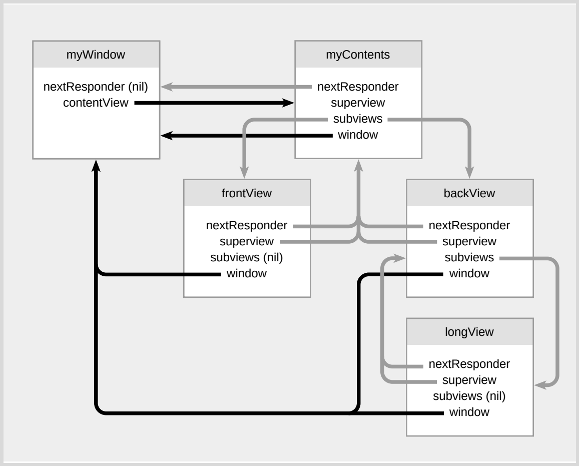
Within every window's content rectangle—the area enclosed by the border, title bar, and resize bar— is its *content view*. The content view is at the top of the view hierarchy. All other views of the window descend from it. Each view has one other view as its *superview* and can be the superview for any number of *subviews*.

What physically determines a view's place in the hierarchy is *enclosure*. A superview encloses its subviews. NSView stores pointers to three objects that reflect a view's physical relationships to other views in the window and locate the view in the hierarchy:

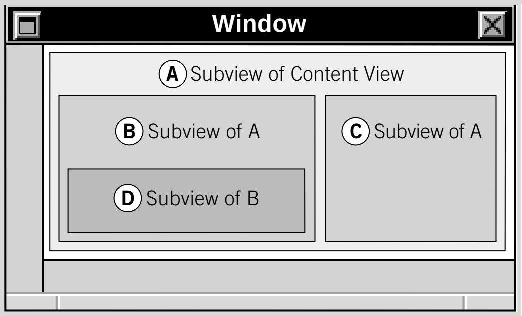
- **window** identifies the view's window (the window points to the content view)
- **superview** identifies the view's superview
- **subviews** a list of the view's subviews

The defining relationship of enclosure makes it easier to draw a view:

- It allows you to construct a view object (the superview) from its subviews.



- Views are positioned within the coordinates of their superviews, so when a view is moved or its coordinate system is transformed, all its subviews are moved and transformed with it.
- Each view has its own coordinate system for drawing. Since a view draws within its own coordinate system, its drawing instructions can remain constant no matter where it or its superview moves on the screen.



Two other attributes, the frame and bounds rectangles, set the location, dimensions, and coordinate systems of a view. **frame** holds the position and size of a view within its superview's coordinate system. The **frame** rectangle defines the area in which drawing can occur. The origin point of a frame locates the lower-left corner of the rectangle in the superview's coordinates. The **bounds** rectangle occupies the same area as the frame rectangle, but it is stated in a different coordinate system; the frame's origin becomes the origin (0.0, 0.0) of the view's drawing coordinates (**bounds.origin**). The bounds rectangle is thus expressed in the view's own drawing coordinates.

Another attribute, inherited from the NSResponder class, determines how events are handled within the view hierarchy. The **nextResponder** by default identifies a view's superview. If a view receives an event message (for example, **mouseDown:**) and cannot handle it, that message is passed on to the view identified by **nextResponder**. See the specifications of the Application Kit's NSView and NSResponder classes in the *Application Kit Reference* for more information on the view hierarchy and event handling.