;I_ToDo_Inspector.rtfd;linkMarkername ;¬ Previous Section ;K_ToDo_Timers.rtfd;linkMarkername ;¬ Next Section

4. To Do Tutorial

Subclass Example: Overriding and Adding Behavior (ToDoCell)

Buttons in the Application Kit are two-state controls. They have twoÐand only twoÐstates: 1 and 0 (often expressed as Boolean YES and NO, or ON and OFF). For the To Do application, a three-state button is preferable. You want the button to indicate, with an image, three possible states: notDone (no image), done (an ªXº), and deferred (a check mark). These states correspond to the possible statues of a ToDoItem.

The ToDoCell class, which you will implement in this section, generates cells that behave as three-state buttons. These buttons also display the time an item is due.

TD_ToDoCell.eps ¬

The superclass of ToDoCell is NSButtonCell. In creating ToDoCell you will add data and behavior to NSButtonCell, and you will override some existing behavior.

1 Add header and implementation files to the project.

Chose New in Project from the File menu.

In the New File In ToDo panel, select the Class suitcase, click Create header, type ªToDoCellº after Name, and click OK.

2 Complete ToDoCell.h.

Make the superclass NSButtonCell.

Add the instance-variable and method declarations shown below.
Add the **enum** constants for state values (as shown).

```
enum _ToDoButtonState {notDone=0, done, deferred} ToDoButtonState;

@interface ToDoCell : NSButtonCell
{
    ToDoButtonState triState;
    NSImage *doneImage, *deferredImage;
    NSDate *timeDue;
}
- (void)setTriState:(ToDoButtonState)newState;
- (ToDoButtonState)triState;
- (void)setTimeDue:(NSDate *)newTime;
- (NSDate *)timeDue;
@end
```

The **triState** instance variable will be assigned ToDoButtonState constants as values. The NSImage variables hold the ªXº and check mark images that represent statuses of completed and deferred (that is, rescheduled for the next day). The **timeDue** instance variable carries the time the item is due as an NSDate; for display, this object will be converted to a string.

Related Concept: ;ToDoConcepts.rtfd;linkMarkername WhyChoseNSButtonCellasSuperclass?;, Why Chose NSButtonCell as Superclass?

3 Initialize the allocated ToDoCell instance (and deallocate it).

Select **ToDoCell.m** in the project browser.
Implement **init** as shown below.

Implement **dealloc**.

```
- (id)init
{
    NSString *path;
    [super initTextCell:@""];

    triState = notDone;
    [self setType:NSToggleButton];                          /* 1 */
    [self setImagePosition:NSImageLeft];
    [self setBezeled:YES];
    [self setFont:[NSFont userFontOfSize:12]];
    [self setAlignment:NSRightTextAlignment];
                                                            /* 2 */
    path = [[NSBundle mainBundle] pathForImageResource:@"X.tiff"];
    doneImage = [[NSImage alloc] initByReferencingFile:path];
    path = [[NSBundle mainBundle]
        pathForImageResource:@"checkMark.tiff"];
    deferredImage = [[NSImage alloc] initByReferencingFile:path];

    return self;
}
```

1. Sets some superclass (NSButtonCell) attributes, such as button type, image and text position, font of text, and border.
2. Through NSBundle's **pathForImageResource:**, gets the pathname for the cell images and creates and stores the images using the pathname.

4 Implement the accessor methods related to state.

Write the methods that get and set the triState instance variable.
Override the superclass methods that get and set state.

```
- (void)setTriState:(ToDoButtonState)newState            /* 1 */
{
    if (newState == deferred+1)
        triState = notDone;
    else
        triState = newState;
    [self _setImage:triState];
 }

- (ToDoButtonState)triState {return triState;}

- (void)setState:(int)val                               /* 2 */
{
}

- (int)state                                            /* 3 */
{
    if (triState == deferred)
        return (int)done;
    else
        return (int)triState;
}
```

Accessing state information is a dual-path task in ToDoCell. It involves not only setting and getting the new state instance variable, **triState**, but properly handling the inherited instance variable by overriding the

superclass accessor methods for state.

1. If the new value for **triState** is one greater than the limit (**deferred**), reset it to zero (**notDone**); otherwise, assign the value. The reason behind this logic is that (as you'll soon learn) when users click a ToDoCell, **setTriState:** is invoked with an argument one more than the current value. This way users can cycle through the three states of ToDoCell.

2. Overrides **setState:** to be a null method. The reason for this override is that NSCell intervenes when a button is clicked, resetting state to zero (NO). This override nullifies that effect.

3. Overrides **state** to return a reasonable value to client objects that invoke this accessor method.

5 Set the cell image.

Declare the private method **_setImage:**.
Implement the **_setImage:** method.

```
@interface ToDoCell (PrivateMethods)
- (void)_setImage:(ToDoButtonState)aState;                    /* 1 */
@end
/* ... */
- (void)_setImage:(ToDoButtonState)aState
{
    switch(aState) {                                          /* 2 */
       case notDone: {
           [self setImage:nil];
           break;
       }
       case done: {
           [self setImage:doneImage];
```

```
            break;
        }
        case deferred: {
            [self setImage:deferredImage];
            break;
        }
    }
    [(NSControl *)[self controlView] updateCell:self];    /* 3 */
}
```

This portion of code handles the display of the cell's image by doing the following:

1. In a category of ToDoCell in **ToDoCell.m**, it declares the private method **_setImage:**. Private methods, which by convention begin with an underscore, are methods that you don't want clients of your object to invoke. In this case, you don't want the image to be set independently from the cell's **triState** value.
2. In a switch statement, evaluates the tri-state argument and sets the cell's image appropriately (**setImage:** is an NSButtonCell method).
3. Sends **updateCell:** to the control view of the cell's control (a matrix) to force a re-draw of the cell.

6 Track mouse clicks on a ToDoCell and reset state.

Override two NSCell mouse-tracking methods as shown in this example.

```
- (BOOL)startTrackingAt:(NSPoint)startPoint inView:
€€(NSView *)controlView
{
    return YES;
}
```

```
- (void)stopTracking:(NSPoint)lastPoint at:(NSPoint)stopPoint
€€inView:(NSView *)controlView mouseIsUp:(BOOL)flag
{
    if (flag == YES) {
        [self setTriState:([self triState]+1)];
    }
}
```

When you create your own cell subclass, you might want to override some methods that are intrinsic to the behavior of the cell. Mouse-tracking methods, inherited from NSCell, are among these. You can override these methods to incorporate specialized behavior when the mouse clicks the cell or drags over it. ToDoCell overrides these methods to increment the value of **triState**.

SquareBullet.eps ¬ Overrides **startTrackingAt:inView:** to return YES, thus signalling to the control that the ToDoCell will track the mouse.

733193_SquareBullet.eps ¬ Overrides **stopTracking:at:inView:mouseIsUp:** to evaluate flag and, if it's YES, to increment the **triState** instance variable. (The **setTriState:** method ªwrapsº the incremented value to zero (**notDone**) if it is greater than 2 (**deferred**)).

7 Get and set the time due, displaying the time in the process.

Implement **setTimeDue:** as shown in this example.

Implement **timeDue** to return the NSDate.

```
- (void)setTimeDue:(NSDate *)newTime
{
    if (timeDue)
        [timeDue autorelease];
    if (newTime) {
```

```
        timeDue = [newTime copy];
        [self setTitle:[timeDue descriptionWithCalendarFormat:
            @"%I:%M %p" timeZone:[NSTimeZone localTimeZone]
            locale:nil]];
    }
    else {
        timeDue = nil;
        [self setTitle:@"-->"];
    }
}
```

The **setTimeDue:** method is similar to other ªsetº accessor methods, except that it handles interpretation and display of the NSDate instance variable it stores. If **newTime** is a valid object, it uses NSDate's **descriptionWithCalendarFormat:timeZone:locale:** method to interpret and format the date object, then displays the result with **setTitle:**. If **newTime** is **nil**, no due time has been specified, and so the method sets the title to ª-->º.

You've now completed all code required for ToDoCell. However, you must now ªinstallº instances of this class in the To Do interface.

8 At launch time, create and install your custom cells in the matrix.

Select **ToDoDoc.m** in the project browser.

Insert the code below in **awakeFromNib**.

```
- (void)awakeFromNib
{
    int i;
/* ... */
    i = [[markMatrix cells] count];
    while (i--) {
```

```
            ToDoCell *aCell = [[ToDoCell alloc] init];
            [aCell setTarget:self];
            [aCell setAction:@selector(itemChecked:)];
            [markMatrix putCell:aCell atRow:i column:0];
            [aCell release];
        }
    }
```

This block of code substitutes a ToDoCell for each cell in the left matrix (**markMatrix**) you created for the To Do interface. It creates a ToDoCell, sets its target and action message, then inserts it into the **markMatrix** by invoking NSMatrix's **putCell:atRow:column:** method.

Finally, you must implement the action message sent when the matrix of ToDoCells is clicked. (This response to mouse-down is for objects external to ToDoCell, while the mouse-tracking response sets state internally.)

9 Respond to mouse clicks on the matrix of ToDoCell's.

In **ToDoDoc.m**, implement i**temChecked:**.

```
- (void)itemChecked:sender
{
    int row = [sender selectedRow];
    ToDoCell *cell = [sender cellAtRow:row column:0];
    if (cell && [currentItems count]) {
        id item = [currentItems objectAtIndex:row];
        if (item && [item isKindOfClass:[ToDoItem class]]) {
            [item setItemStatus:[cell triState]];
            [[sender window] setDocumentEdited:YES];
        }
    }
}
```

This method gets the ToDoCell that was clicked and the object in the corresponding text field. If that object is a ToDoItem, the method updates its status to reflect the state of the ToDoCell. It then marks the window as containing an edited document.

Related Concepts: ;ToDoConcepts.rtfd;linkMarkername AShortGuidetoDrawingandCompositing;, A Short Guide to Drawing and Compositing
;ToDoConcepts.rtfd;linkMarkername MakingaCustomView;,
Making a Custom View