;C_ToDo_ModelClass.rtfd;;¬ Previous Section ;E_ToDo_MultiAppBasics.rtfd;;¬ Next Section

4. To Do Tutorial

Subclass Example: Adding Data and Behavior (CalendarMatrix)

The calendar on To Do's interface is an instance of a custom subclass of NSMatrix. CalendarMatrix dynamically updates itself as users select new months, notifies a delegate when users select a day, and reflects the current day (today) and the current selection by setting button attributes.

TD_SubclassExample1.eps ¬

Creating a subclass of a class that is farther down the inheritance tree poses more of a challenge for a developer than a simple sublcass of NSObject. A class such as NSMatrix is more specialized than NSObject and carries with it more baggage: It inherits from NSResponder, NSView, and NSControl, all fairly complex Application Kit classes. And since CalendarMatrix inherits from NSView, it appears on the user interface; it is an example of a view object in the Model-View-Controller paradigm, and as such it is highly reusable.

Why NSMatrix?

When you select a specialized superclass as the basis for your subclass, it is important to consider what your requirements are and to understand what the superclass has to offer. To Do's dynamic calendar should:

SquareBullet.eps ¬ Arrange numbers (days) sequentially in rows and columns
210268_SquareBullet.eps ¬ Respond to and communicate selections of days

318599_SquareBullet.eps ¬ Understand dates
413854_SquareBullet.eps ¬ Enable navigation between months

If you then started to peruse the reference documentation on Application Kit classes, and looked at the section on NSMatrix, you'd read this:

> *NSMatrix is a class used for creating groups of NSCells that work together in various ways. It includes methods for arranging NSCells in rows and columns....An NSMatrix adds to NSControl's target/action paradigm by allowing a separate target and action for each of its NSCells in addition to its own target and action.*

So NSMatrix has an inherent capability for the first of the requirements listed above, and part of the second (responding to selections). Our CalendarMatrix subclass thus does not need to alter anything in its superclass. It just needs to supplement NSMatrix with additional data and behavior so it can understand dates (and update itself appropriately), navigate between months, and notify a delegate that a selection was made.

1 Define the CalendarMatrix class in Interface Builder.

From Project Builder, open **ToDo.nib**.
In Interface Builder, choose Document arrow.eps ¬ New Module 731093_arrow.eps ¬ New Empty to create a new nib file.
Save the nib file as **ToDoDoc.nib**.
In the Classes display of the nib file window, select NSMatrix.
Choose Subclass from the pull-down list.
Name the new class ªCalendarMatrixº.
Select the new class.
Add the outlets and actions shown in the example at right.

TD_SubclassExample2.eps ¬

When you created subclasses of NSObject in the previous two tutorials, the next step was to instantiate the subclass. Because CalendarMatrix is a view (that is, it inherits from NSView), the procedure for generating an instance for making connections is different.

2 Put a custom NSView object (CalendarMatrix) on the user interface.

Drag a window from the Windows palette.
Resize the window, using the example below as a guide.
Turn off the window's resize handle.
Drag a CustomView from the Views palette onto the window.
Resize and position the CustomView, using the example below as a guide.
In the Attributes display of the inspector, select CalendarMatrix from the list of available classes.

TD_SubclassExample3.eps ¬

The selection of the class for the CustomView creates an instance of it that you can connect to other objects in the nib file. Now put the controls and fields associated with CalendarMatrix on the window.

3 Put the objects related to CalendarMatrix on the window.

Drag a label object for the month-year from the Views palette and put it over the CalendarMatrix.
Make seven small labels for each day of the week.
Drag a button onto the interface and set its attributes to unbordered and image only.
Drag **left_arrow.tiff** from **/NextDeveloper/Examples/AppKit/ToDo** and drop it over the button.
To the attention panel that asks ªInsert image left_arrow in project?º click Yes.
Repeat the same button procedure for **right_arrow.tiff**.

TD_SubclassExample4.eps ¬.

Next connect CalendarMatrix to its satellite objects.

4 Connect CalendarMatrix to its outlet and to the controls sending action messages.

Name	Connection	Type
TableHeadRule.eps ¬		
monthName	From CalendarMatrix to the label field above it	outlet
TableRule.eps ¬		
leftButton	From CalendarMatrix to the left-pointing arrow	outlet
701729_TableRule.eps ¬		
rightButton	From CalendarMatrix to the right-pointing arrow	outlet
810318_TableRule.eps ¬		
monthChanged:	From both arrows to CalendarMatrix	action

5 Finish up in Interface Builder.

Save **ToDoDoc.nib**.

Select CalendarMatrix and in the Classes display and choose Create Files from the Operations pull-down menu.

Confirm that you want the source-code files added to the project.

You might have noticed that there's an action message left unconnected: **choseDay:**. Because it is impossible in Interface Builder to connect an object with itself, you will make this connection programmatically.

6 Add declarations to the header file CalendarMatrix.h.

(Existing declarations are indicted by ellipsis.)

```
@interface CalendarMatrix : NSMatrix
{
 /* ... */
```

```
    NSCalendarDate *selectedDay;
    short startOffset;                                  /* 1 */
}
 /* ... */
- (void)refreshCalendar;
- (id)initWithFrame:(NSRect)frameRect;
- (void)dealloc;
- (void)setSelectedDay:(NSCalendarDate *)newDay;
- (NSCalendarDate *)selectedDay;
@end

@interface NSObject(CalendarMatrixDelegate)             /* 2 */
 - (void)calendarMatrix:(CalendarMatrix *)obj
     didChangeToDate:(NSDate *)date;
 - (void)calendarMatrix:(CalendarMatrix *)obj
     didChangeToMonth:(int)mo year:(int)yr;
@end
```

There are a couple of interesting things to note about these declarations:

1. The cells in CalendarMatrix are sequentially ordered by tag number, left to right, going downward. **startOffset** marks the cell (by its tag) on which the first day of the month falls.

2. CalendarMatrixDelegate is a category on NSObject that declares the methods to be implemented by the delegate. This technique creates what is called an *informal protocol*, which is commonly used for delegation methods.

7 Implement CalendarMatrix's initialization methods.

Select **CalendarMatrix.m** in the project browser.
Write the implementation of **initWithFrame:** (below).
Implement **dealloc**.

```
- (id)initWithFrame:(NSRect)frameRect
{
    int i, j, cnt=0;
    id cell = [[NSButtonCell alloc] initTextCell:@""];
    NSCalendarDate *now = [NSCalendarDate date];          /* 1 */

    [super initWithFrame:frameRect                        /* 2 */
                    mode:NSRadioModeMatrix
               prototype:cell
            numberOfRows:6
         numberOfColumns:7];
    // set cell tags                                      /* 3 */
    for (i=0; i<6; i++) {
        for (j=0; j<7; j++) {
            [[self cellAtRow:i column:j] setTag:cnt++];
        }
    }
    [cell release];
    selectedDay = [[NSCalendarDate dateWithYear:[now yearOfCommonEra]
        month:[now monthOfYear]                           /* 4 */
          day:[now dayOfMonth]
         hour:0 minute:0 second:0
     timeZone:[NSTimeZone localTimeZone]] copy];
```

```
    return self;
}
```

The **initWithFrame:** method is an initializer of NSMatrix, NSControl and NSView.

1. This invocation of **date**, a class method declared by NSDate, returns the current date (ªtodayº) as an NSCalendarDate. (NSCalendarDate is a subclass of NSDate.)
2. This message to **super** (NSMatrix) sets the physical and cell dimensions of the matrix, identifies the type of cell using a prototype (an NSButtonCell), and specifies the general behavior of the matrix: radio mode, which means that only one button can be selected at any time.
3. Set the tag number of each cell sequentially left to right and down. Tags are the mechanism by which CalendarMatrix sets and retrieves the day numbers of cells.
4. This NSCalendarDate class method initializes the **selectedDay** instance variable to midnight of the current day, using the year, month, and day elements of the current date. The **localTimeZone** message obtains an NSTimeZone object with an suitable offset from Greenwich Mean Time.

Implement **awakeFromNib** as shown below.

```
- (void)awakeFromNib
{
    [monthName setAlignment:NSCenterTextAlignment];
    [self setTarget:self];
    [self setAction:@selector(choseDay:)];
    [self setAutosizesCells:YES];
    [self refreshCalendar];
}
```

The **awakeFromNib** method performs additional initializations (some of which could just have easily been done in **initWithFrame:**). Most importantly, it sets **self** as its own target object and specifies an action method for this target, **choseDay:**, something that couldn't be done in Interface Builder. Other methods to note:

127842_SquareBullet.eps ¬ **setAutosizesCells:** causes the matrix to resize its cells on every redraw.
379877_SquareBullet.eps ¬ **refreshCalendar** (which you'll write next) updates the calendar.

The **refreshCalendar** method is fairly long and complexÐit is the workhorse of the classÐso you'll approach it in sections.

Related Concept: ;ToDoConcepts.rtfd;linkMarkername DatesandTimesinOpenStep;, Dates and Times in OpenStep

8 Implement the code that updates the calendar.

Initialize the **MonthDays[]** array and write the **isLeap()** macro.

Determine the day of the week at the start of the month and the number of days in the month.

```
static short MonthDays[] =
    {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
#define isLeap(year) (((((year) % 4) == 0 && (((year) % 100) != 0))
     €€€€€€€€€€€|| ((year) % 400) == 0))


- (void)refreshCalendar
{
    NSCalendarDate *firstOfMonth, *selDate = [self selectedDay],
        *now = [NSCalendarDate date];
```

```
int i, j, currentMonth = [selDate monthOfYear];
unsigned int currentYear = [selDate yearOfCommonEra];
short daysInMonth;
id cell;

firstOfMonth = [NSCalendarDate dateWithYear:currentYear    /* 1 */
               month:currentMonth
                 day:1 hour:0 minute:0 second:0
            timeZone:[NSTimeZone localTimeZone]];
[monthName setStringValue:[firstOfMonth                    /* 2 */
    descriptionWithCalendarFormat:@"%B %Y"]];
daysInMonth = MonthDays[currentMonth-1]+1;                 /* 3 */
/* correct Feb for leap year */
if ((currentMonth == 2) && (isLeap(currentYear))) daysInMonth++;
startOffset = [firstOfMonth dayOfWeek];                    /* 4 */
```

Before it can start writing day numbers to the calendar for a given month, CalendarMatrix must know what cell to start with and how many cells to fill with numbers. The **refreshCalendar** method begins by calculating these values.

1. Creates an NSCalendarDate for the first day of the currently selected month and year (computed from the **selectedDay** instance variable).
2. Writes the month and year (for example, ªFebruary 1997º) to the label above the calendar.
3. Gets from the **MonthDays** static array the number of days for that month; if the month is February and it is a leap year, this number is adjusted.
4. Gets the day of the week for the first day of the month and stores this in the **startOffset** instance variable.

Write the refreshCalendar code that writes day numbers to the cells and sets cell attributes.

```
    for (i=0; i<startOffset; i++) {
     cell = [self cellWithTag:i];
     [cell setBordered:NO];
     [cell setEnabled:NO];
     [cell setTitle:@""];
     [cell setCellAttribute:NSCellHighlighted to:NO];
}
for (j=1; j < daysInMonth; i++, j++) {
    cell = [self cellWithTag:i];
    [cell setBordered:YES];
    [cell setEnabled:YES];
    [cell setFont:[NSFont systemFontOfSize:12]];
    [cell setTitle:[NSString stringWithFormat:@"%d", j]];
    [cell setCellAttribute:NSCellHighlighted to:NO];
}
for (;i<42;i++) {
    cell = [self cellWithTag:i];
    [cell setBordered:NO];
    [cell setEnabled:NO];
    [cell setTitle:@""];
    [cell setCellAttribute:NSCellHighlighted to:NO];
}
```

The first and third for-loops in this section of code clear the leading and trailing cells that aren't part of the month's days. Because the current day is indicated by highlighting, they also turn off the highlighted attribute.

The second for-loop writes the day numbers of the month, starting at **startOffset** and continuing until **daysInMonth**, and resets the font (since the selected day is in bold face) and other cell attributes.

Complete the refreshCalendar method implementation by resetting the ªtodayº cell attribute.

```
        if ((currentYear == [now yearOfCommonEra])
            && (currentMonth == [now monthOfYear])) {
         [[self cellWithTag:([now dayOfMonth]+startOffset)-1]
    setCellAttribute:NSCellHighlighted to:YES];
        [[self cellWithTag:([now dayOfMonth]+startOffset)-1]
    setHighlightsBy:NSMomentaryChangeButton];
        }
}
```

This final section of **refreshCalendar** determines if the newly selected month and year are the same as today's, and if so highlights the cell corresponding to today.

9 Implement the monthChanged: action method.

```
- (void)monthChanged:sender
{
    NSCalendarDate *thisDate = [self selectedDay];
    int currentYear = [thisDate yearOfCommonEra];
    unsigned int currentMonth = [thisDate monthOfYear];

    if (sender == rightButton) {                          /* 1 */
        if (currentMonth == 12) {
            currentMonth = 1;
            currentYear++;
```

```
        } else {
            currentMonth++;
        }
    } else {
        if (currentMonth == 1) {
            currentMonth = 12;
            currentYear--;
        } else {
            currentMonth--;
        }
    }                                                    /* 2 */
    [self setSelectedDay:[NSCalendarDate dateWithYear:currentYear
                   month:currentMonth
                     €day:1 hour:0 minute:0 second:0
                timeZone:[NSTimeZone localTimeZone]]];
    [self refreshCalendar];
    [[self delegate] calendarMatrix:self                 /* 3 */
        didChangeToMonth:currentMonth year:currentYear];
}
```

The arrow buttons above CalendarMatrix send it the **monthChanged:** message when they are clicked. This method causes the calendar to go forward or backward a month.

1. Determines which button is sending the message, then increments or decrements the month accordingly. If it goes past the end or beginning of the year, it increments or decrements the year and adjusts the month.

2. Resets the **selectedDay** instance variable with the new month (and perhaps year) numbers and invokes

refreshCalendar to display the new month.

3. Sends the **calendarMatrix:didChangeToMonth:year:** message to its delegate (which in this application, as you'll soon see, is a ToDoDoc controller object).

10 Implement the choseDay: action method.

```
- (void)choseDay:sender
{
    NSCalendarDate *selDate, *thisDate = [self selectedDay];
/* 1 */
    unsigned int selDay = [[self selectedCell] tag]-startOffset+1;
/* 2 */
    selDate = [NSCalendarDate dateWithYear:[thisDate yearOfCommonEra]
                                     month:[thisDate monthOfYear]
                                       day:selDay
                                      hour:0
                                    minute:0
                                    second:0
                                  timeZone:[NSTimeZone localTimeZone]];
/* 3 */
    [[self cellWithTag:[thisDate dayOfMonth]+startOffset-1]
        setFont:[NSFont systemFontOfSize:12]];
    [[self cellWithTag:selDay+startOffset-1] setFont:
        [NSFont boldSystemFontOfSize:12]];
/* 4 */
    [self setSelectedDay:selDate];
    [[self delegate] calendarMatrix:self didChangeToDate:selDate];
```

```
}
```

This method is invoked when users click a day of the calendar.

1. Gets the tag number of the selected cell and subtracts the offset from it (plus one to adjust for zero-based indexing) to find the number of the selected day.
2. Derives an NSCalendarDate that represents the selected date.
3. Sets the font of the previously selected cell to the normal system font (removing the bold attribute) and puts the number of the currently selected cell in bold face.
4. Sets the **selectedDay** instance variable to the new date and sends the **calendarMatrix:didChangeToDate:** message to the delegate.

11 Implement accessor methods for the selectedDay instance variable.

You are finished with CalendarMatrix. If you loaded **ToDoDoc.nib** right now, the calendar would work, up to a point.. If you clicked the arrow buttons, CalendarMatrix would display the next or previous months. The days of the month would be properly set out on the window, and the current day would be highlighted.

But not much else would happen. That's because CalendarMatrix has not yet been hooked up to its delegate.