;AddingExistingClassesToYourNibFile.rtfd;;¬ Next Section ;MakingYourClassADelegate.rtfd;;¬ Previous Section

ImplementingasubclassofNSView;¬Implementing a subclass of NSView

1. **Identify the class and its outlets and actions.**
2. **Place and resize an object from the Views palette on a window or panel.**
3. **Assign your class as the class of the object.**
4. **Connect the instance to other objects in the interface.**
5. **Generate code files.**
6. **Complete programming tasks necessary for any object.**
7. **Complete programming tasks specific to NSView objects:**

arrow.eps ¬ Initialize an NSView object.
arrow.eps ¬ Draw an NSView object.
arrow.eps ¬ If necessary, handle events.

Making a subclass of the NSView class is a procedure that differs from making a subclass of the NSObject class. But it starts out the same. In the Classes display of Interface Builder, choose Subclass from the Operations menu while NSView or one of its subclasses is highlighted in the browser. Then name your class and add outlets and actions.

NibClassesNSViewSubclass.tiff ¬

Making an Instance of an NSView Subclass

Place an instance of your class on your interface. If you're subclassing an NSView subclass (such as NSButton or NSTextField), drag the object that represents that class (that is, the button or the text field) from the palette window into your interface's the window. If you're subclassing NSView directly, use the

CustomView object on the Views palette.

_ImplementingSubclassNSView1.eps ¬

Position and resize the object, and while it's still selected, bring up the Custom Class display of the Inspector panel by typing Command-5. Assign a class name to the object; this creates an instance of your NSView subclass.

_ImplementingSubclassNSView2.eps ¬

Tip: Make sure you choose the appropriate superclass. If you subclass an NSView subclass, rather than subclassing NSView directly, you can still set the that class's attributes for your object using the Inspector panel's Attributes display. If you subclass NSView directly, you lose the ability to set attributes using the Inspector panel.

The next three steps that you must complete are the same tasks that follow the instantiation of NSObject subclasses:

SquareBullet.eps ¬ Connect the instance to other objects in the interface (ªConnecting your class's outletsº and ªConnecting your class's actionsº). But now the instance appears as part of the interface, and not as an icon in the Instances display of the nib file window.

SquareBullet.eps ¬ Generate code files and have them inserted in your project (ªGenerating source code filesº).

SquareBullet.eps ¬ Switch over to the project in Project Builder that contains the nib file, and open your class's code files.

Since NSView inherits from NSObject, next complete some of the same programming tasks recommended for subclasses of NSObject:

SquareBullet.eps ¬ Declaring new instance variables
SquareBullet.eps ¬ Implementing accessor methods
SquareBullet.eps ¬ Implementing target/action methods
SquareBullet.eps ¬ Archiving and unarchiving

To create a functional subclass of NSView, you must complete two additional steps (and might want to

complete another), which are described on the following pages.

Initializing NSView Objects

Subclasses of NSView override **initWithFrame:** instead of **init**. In **initWithFrame:** (NSView's designated initializer) you initialize a just-allocated instance of your class, setting its attributes to an initial state. The method's sole argument is the rectangle in which drawing is to occur (usually the frame of the view).

In this example, **initWithFrame:** initializes instance variables of varying types and performs other housekeeping chores.

```
- (id)initWithFrame:(NSRect)frameRect {
     [super initWithFrame:frameRect];
     glist = [[NSMutableArray allocWithZone:[self zone]]
        init];
     slist = [[NSMutableArray allocWithZone:[self zone]]
        init];
     cacheImage = [self createCacheWithSize:[self bounds].size];
     [self cache:[self bounds] andUpdateLinks:NO];
     gvFlags.grid = 10;
     gvFlags.gridDisabled = 1;
     [self allocateGState];
     gridGray = DEFAULT_GRID_GRAY;
     PSInit();
     currentGraphic = [Rectangle class];          /* default graphic */
/* trick to allow NSApp to control currentGraphic */
     currentGraphic = [self currentGraphic];
     editView = [self createEditView];
     [[self class] initClassVars];
     [self registerForDragging];
     spellDocTag = 0;
     return self;
}
```

Notes on the code: The implementation of an **initWithFrame:** method begins by invoking **super**'s **initWithFrame:** method, ends by returning **self**, and in between sets the instance variables to initial values. Often the attributes set have a visual aspect, and affect how the view is drawn.

As with NSObject subclasses, you might have to implement the **dealloc** method to deallocate dynamically allocated storage.

> The NSView class offers your subclass a wealth of inherent functionality. It includes methods for managing the view hierarchy, for converting coordinates and modifying the coordinate system, for managing cursors and events, and for focusing, clipping, scrolling, dragging, and printing. See the description of the NSView class in the *Application Kit Reference*. ;/NextLibrary/Frameworks/AppKit.framework/Resources/English.lproj/Documentation/Reference/Classes/NSView.rtfd;;¬

Drawing NSView Objects

An NSView object draws itself with the **drawRect:** method. To invoke **drawRect:**, another object must send **display** to the NSView object. The **drawRect:** method is also invoked automatically when windows are resized and exposed, when NSViews are scrolled, and when similar events happen.The NSRect argument passed to **drawRect:** indicates how much of the NSView needs to be drawn.

```
- (void)drawRect:(NSRect)rect
{

  int grid;
  float gray;


  grid = [spacing intValue];
  grid = MAX(grid, 0.0);
  PSsetgray(NSWhite);
  NSRectFill(rect);
  if (grid >= 4) {
       gray = [grayField floatValue];
       gray = MIN(gray, 1.0);
       gray = MAX(gray, 0.0);
       PSsetgray(gray);
       PSsetlinewidth(0.0);
```

```
        [self drawGrid:grid];
    }
    PSsetgray(NSBlack);
    NSFrameRect([self bounds]);
}
```

Notes on the code: The example above shows **drawRect:**. This example fills in the view with a white background, draws a grid using a user-selectable gray value, then uses **NSFrameRect()** to draw a black border around the view.

In implementing **drawRect:**, write whatever code helps to draw your NSView. You can call **pswrap**-generated functions to send PostScript code to the Window Server. You can send messages to bitmap objects, requesting them to composite source images stored in off-screen windows. You can change font styles and text colors. If your NSView uses an NSCell to do any of its drawing, you can send **drawWithFrame:inView:** or **drawInteriorWithFrame:inView:** to the NSCell within **drawRect:**.

> The PostScript functions and operators available for use are described in *DPSClientLibrary Reference*.
>
> **pswrap** is a program that creates a C function to correspond to a sequence of PostScript code. Note that your custom **pswrap** code (extension **.psw**) must go in Project Builder under Other Sources. **pswrap** is described in detail in Adobe Systems' *pswrap Reference Manual*.

The **drawRect:** method defines an NSView's static appearance on the screen. Your subclass can also add other methods for dynamic drawing in response to user events. In these methods you might highlight the NSView, drag it from one place to another, or animate it. The Application Kit locks focus automatically when **drawRect:** is invoked. In dynamic-drawing contexts you must lock and unlock focus yourself when drawing.

If you want your view to respond to mouse clicks, key presses, or other user events, you must do at least two things:

SquareBullet.eps ¬Re-implement NSView's **acceptsFirstResponder** method to return YES.

SquareBullet.eps ¬Decide which event types you want to respond to and implement the appropriate methods: **mouseUp:**, **mouseDown:**, **keyDown:**, **mouseEntered:**, and so on.

The event methods are defined in the NSResponder class, where the default implementation is to forward the event message to the next responder.

When it invokes an event method, the input system passes in an NSEvent object. This object holds details related to the event: the type of event, the mouse's location (in the window's base coordinates), the window number, a time value associated with the event, flags indicating modifier keys and mouse buttons, and supplementary data.

You can find or derive much of the information required for handling an event in the NSEvent parameter. For instance, you can convert the NSEvent mouse location to your NSView's base coordinate system with **convertPoint:fromView:**. You can check for modifier keys or mouse buttons using the keyboard-state flags masks.

The following example illustrates several of these techniques.

```
- (void)mouseDown:(NSEvent *)event
{
     NSPoint p, start;
     int grid, gridCount;

     start = [event locationInWindow];
     start = [self convertPoint:start fromView:nil];
     grid = MAX([spacing intValue], 1.0);
     gridCount = (int)MAX(start.x, start.y) / grid;
     gridCount = MAX(gridCount, 1.0);

     event = [[self window] nextEventMatchingMask:
        NSLeftMouseDraggedMask|NSLeftMouseUpMask];
     while ([event type] != NSLeftMouseUp) {
        p = [event locationInWindow];
        p = [self convertPoint:p fromView:nil];
        grid = (int)MAX(p.x, p.y) / gridCount;
        grid = MAX(grid, 1.0);
        if (grid != [spacing intValue]) {
            [form abortEditing];
            [spacing setIntValue:grid];
```

```
            [self display];
        }
        event = [[self window] nextEventMatchingMask:
            NSLeftMouseDraggedMask|NSLeftMouseUpMask];
    }
}
```

Tip: If you want your NSView to handle target/action messages sent to the First Responder (for example, copy and paste), be sure to override **acceptsFirstResponder** to return YES, and then implement the appropriate methods **(copy:** and **paste:)**.

The NSEvent class is described in the *Application Kit Reference*. ;/NextLibrary/Frameworks/AppKit.framework/Resources/English.lproj/Documentation/Reference/Classes/NSEvent.rtfd;;¬