PlotView.h

```
/* PlotView.h -- Interface file for the PlotView class */

#import <appkit/View.h>

@interface PlotView:View
{
    id     delegate;
    id     points;
    id     crossCursor;
    id     readOut;
    float  radius;
    BOOL   needsClearing;
}
- initFrame:(const NXRect *)frameRect;
- setDelegate:anObject;
- drawSelf:(const NXRect *)rects :(int)rectCount;
- sizeTo:(NXCoord)width :(NXCoord)height;
- clear:sender;
- plot:sender;
- mouseDown:(NXEvent *)theEvent;
- registerPoint:(NXPoint *)aPoint;
- setRadius:(float)aFloat;
- (float)radius;
- read:(NXTypedStream *)stream;
- write:(NXTypedStream *)stream;
- awake;
- (const char *)inspectorName;
@end

@interface Object(PlotViewDelegate)
- plotView:sender providePoints:(NXStream **)stream;
- plotView:sender pointDidChange:(NXPoint *)aPoint;
@end
```

PlotView.m

```
/* PlotView.m -- Implementation file for the PlotView class */

#import "PlotView.h"
#import "drawingFuncs.h"
#import <objc/Storage.h>
#import <appkit/NXCursor.h>
#import <appkit/Application.h>
#import <appkit/Window.h>
#import <appkit/Cell.h>
#import <appkit/Pasteboard.h>
#import <NXCType.h>
#import <math.h>
#import <dpsclient/psops.h>
#import <dpsclient/wraps.h>
#import <stdlib.h>
#import <string.h>

extern float getNumber(NXStream *stream);
extern int getSeparator(NXStream *stream);

#define MAXNUMLENGTH 50
#define DEFAULTRADIUS 1.0

@implementation PlotView

- initFrame:(const NXRect *)frameRect
/*
 * Initializes the new PlotView object.  First, an initFrame: message
 * is sent to super to initialize PlotView as a View.  Next, the
 * PlotView sets its own state -- that it is opaque and that the
 * origin of its coordinate system lies in the center of its area.
 * It then creates and initializes its associated objects,a Storage
 * object, an NXCursor, and a Cell.  Finally, it loads into the
 * Window Server some PostScript procedures that it will use in drawing
 * itself.
 */
{
    NXPoint  spot;

    [super initFrame:frameRect];
    [self setOpaque:YES];
    [self setRadius:DEFAULTRADIUS];
    [self translate:floor(frame.size.width/2) :floor(frame.size.height/2)];
    points = [[Storage alloc] initCount:0
                              elementSize:sizeof(NXPoint)
                              description:"{ff}"];
    crossCursor = [NXCursor newFromImage:[NXImage newFromSection:"cross.tiff"]];
    spot.x = spot.y = 7.0;
    [crossCursor setHotSpot:&spot];
```

```
    /*
     * Normally, the next two message expressions would be combined, as:
     *     readOut = [[Cell alloc] initTextCell:""];
     * but are here separated for clarity.
     */
    readOut = [Cell alloc];
    [readOut initTextCell:""];
    [readOut setBezeled:YES];
    loadPSProcedures();
    return self;
}


- setDelegate:anObject
/*
 * Sets the PlotView's delegate instance variable to the supplied object.
 */
{
    delegate = anObject;
    return self;
}


- drawSelf:(const NXRect *)rects :(int)rectCount
/*
 * Draws the PlotView's background and axes.  If there are any points,
 * these are drawn too.
 *
 * (Note:  For simplicity, although PlotView only repaints the background
 * of the update region, it redraws the entire axes.  For better performance,
 * only those parts of the axes that fall within the update region should
 * be redrawn.)
 */
{
    unsigned int  i;
    NXPoint      *aPoint;

    if (rects == NULL) return self;

    /* If we're printing, we need to load drawing procedures to printing context */
    if (NXDrawingStatus != NX_DRAWING)
        loadPSProcedures();

    /* paint visible area white then draw axes */
    PSsetgray(NX_WHITE);
    NXRectFill(&rects[0]);
    PSsetgray(NX_DKGRAY);
    drawAxes(bounds.origin.x, bounds.origin.y, bounds.size.width,
        bounds.size.height);

    /* now take each point and draw it */
    PSsetgray(NX_BLACK);
    i = [points count];
    while (i--) {
        aPoint = (NXPoint *)[points elementAt:i];
        drawCircle(aPoint->x, aPoint->y, radius);
    }
    return self;
}

- clear:sender
/*
 * Clears the PlotView by emptying its Storage object of all points
 * and then redisplaying the PlotView.  This action is taken only if
 * the PlotView's state requires it.
 */
{
    if (needsClearing) {
        [points empty];
        [self display];
        needsClearing = NO;
    }
    return self;
}

- sizeTo:(NXCoord)width :(NXCoord)height
/*
 * Ensures that whenever the PlotView is resized, the origin of its
 * coordinate system is repositioned to the center of its area.
 */
{
    [super sizeTo:width :height];
    [self setDrawOrigin:-floor(width/2) : -floor(height/2)];
    return self;
}

- registerPoint:(NXPoint *)aPoint
/*
 * Adds a point to the list the PlotView keeps in its Storage object.
 */
{
    [points addElement:aPoint];
    return self;
}
```

```
- mouseDown:(NXEvent *) theEvent
/*
 * Responds to a message the system sends whenever the user presses the mouse
 * button when the cursor is over the PlotView.  The PlotView changes the
 * cursor to a cross-hairs image and then starts asking for mouse-dragged or
 * mouse-up events.  As it receives mouse-dragged events, the PlotView updates
 * the readOut text Cell with the cursor's coordinates.  If the user releases
 * the mouse button while the cursor is over the PlotView, the PlotView
 * registers the point and then sends a message to its delegate notifying
 * it of the new point.
 */
{
    int      looping = YES, oldMask;
    NXPoint  aPoint;
    NXRect   plotRect, cellRect;
    char     buffer[100];

    [crossCursor set];
    [self getBounds:&plotRect];
    NXSetRect(&cellRect, plotRect.origin.x, plotRect.origin.y, 100.0, 20.0);

    oldMask = [window addToEventMask:NX_MOUSEDRAGGEDMASK];
    [self lockFocus];
    do {
        aPoint = theEvent->location;
        [self convertPoint:&aPoint fromView:nil];
        sprintf(buffer, "(%d, %d)", (int)aPoint.x, (int)aPoint.y);
        [readOut setStringValue:buffer];
        [readOut drawInside:&cellRect inView:self];
        [window flushWindow];
        if (theEvent->type == NX_MOUSEUP) {
            /* on mouse-up, register point, inform delegate, and clean up state */
            [readOut setStringValue:""];
            [readOut drawInside:&cellRect inView:self];
            [window flushWindow];
            if (NXPointInRect(&aPoint, &plotRect)) {
                PSsetgray(NX_BLACK);
                drawCircle(aPoint.x, aPoint.y, radius);
                [window flushWindow];
                [self registerPoint:&aPoint];
                needsClearing = YES;
                if (delegate &&
                    [delegate respondsTo:@selector(plotView:pointDidChange:)])
                        [delegate plotView:self pointDidChange:&aPoint];
            }
            looping = NO;
        }
    } while (looping && (theEvent=[NXApp getNextEvent:NX_MOUSEUPMASK
                                                    |NX_MOUSEDRAGGEDMASK]));
    [self unlockFocus];
    [window setEventMask:oldMask];
    [NXArrow set];
    return self;
}
```

```
- setRadius:(float)aFloat
/*
 * Sets the value for the radius of the PlotView's points.
 */
{
    radius = aFloat;
    return self;
}


- (float)radius
/*
 * Returns the value for the radius of the PlotView's points.
 */
{
    return radius;
}


- read:(NXTypedStream *)stream
/*
 * Unarchives the PlotView.  Initializes four of the PlotView's instance
 * variables to the values stored in the stream.
 */
{
    [super read:stream];
    delegate = NXReadObject(stream);
    NXReadTypes(stream, "@@@f", &points, &crossCursor, &readOut, &radius);
    return self;
}

- write:(NXTypedStream *)stream
/*
 * Archives the PlotView by writing its important instance variables to the
 * stream.
 */
{
    [super write:stream];
    NXWriteObjectReference(stream, delegate);
    NXWriteTypes(stream, "@@@f", &points, &crossCursor, &readOut, &radius);
    return self;
}

- awake
/*
 * Finishes initializing a PlotView that has just been unarchived (see the read:
 * method).
 */
{
    loadPSProcedures();
    return [super awake];
}
```

```
- (const char *)inspectorName
{
    return "PlotViewInspector";
}

- plot:sender
/*
 * Responds to a plot: message by asking the PlotView's delegate for a
 * stream containing the points to plot.  It then extracts number
 * pairs from the stream and creates NXPoint structures with them.
 * For each structure it creates, the PlotView sends itself a
 * registerPoint: message to add the point to its list of points.
 * This simple parser ignores input it doesn't understand.
 */
{
    int        c, sign, retval;
    float      aNum;
    NXPoint    aPoint;
    NXStream  *stream;
    BOOL       processedLine = NO, gotFirst = NO, gotSecond = NO;

    if (delegate && [delegate respondsTo:@selector(plotView:providePoints:)])
        [delegate plotView:self providePoints:&stream];
    NXSeek(stream, 0, NX_FROMSTART);
    while ((c = NXGetc(stream)) != EOF){
        sign = 1;
        retval = getSeparator(stream);
        if (retval == 1) {
            c = NXGetc(stream);
        } else if (retval == -1) {
            break;
        }
        if (!NXIsDigit(c)) {
            if ((c == '-') && NXIsDigit(c = NXGetc(stream))){
                    sign = -1;
            } else {
                while( (c != '\n') && (c != EOF)) {
                    c = NXGetc(stream);
                    processedLine = YES;
                }
            }
        }
        if (c == EOF)
            break;
        else if (processedLine) {
            processedLine = NO;
            continue;
        }
        aNum = getNumber(stream);
        if (!gotFirst) {
            aPoint.x = sign * aNum;
            gotFirst = YES;
        } else if (!gotSecond) {
            aPoint.y = sign * aNum;
            [self registerPoint:&aPoint];
            gotFirst = gotSecond = NO;
        }
    }
    [self display];
    needsClearing = YES;
    return self;
}

int getSeparator(NXStream *stream)
/*
 * Removes white space, commas, and plus signs from between
 * number pairs.
 */
{
    int c, firstChar;

    NXUngetc(stream);
    c = firstChar = NXGetc(stream);
    while (NXIsSpace(c) || (c == ',') || (c == '+'))
        c = NXGetc(stream);
    /* 1 = ate something; 0 = ate nothing; -1 means EOF */
    if (c == firstChar)
        return 0;
    else if (c == EOF)
        return -1;
    NXUngetc(stream);
    return 1;
}

float getNumber(NXStream *stream)
/*
 * Composes a floating point number from a string of number
 * characters.
 */
{
    int c, i = 0;
    char temp[MAXNUMLENGTH];

    NXUngetc(stream);
    c = NXGetc(stream);
    while ((NXIsDigit(c) || (c == '.')) && (c != '\n') && (c != EOF)) {
        temp[i++] = c;
        c = NXGetc(stream);
    }
    NXUngetc(stream);
    temp[i] = 0;
    return (atof(temp));
}

@end
```

PlotController.h

```
/* PlotController.h -- Interface file for the PlotController class */

#import <objc/Object.h>
#import <dpsclient/event.h>

@interface PlotController:Object
{
    id  theScrollView;
    id  inputText;
    id  thePlotView;
}
- appDidInit:sender;
- textDidGetKeys:theText isEmpty:(BOOL)flag;
- plotView:sender providePoints:(NXStream **)stream;
- plotView:sender pointDidChange:(NXPoint *)aPoint;
- requestPlot:pasteboard userData:(const char *)userData error:(char **)msg;
@end
```

PlotController.m

```
/* PlotController.m -- Implementation file for the PlotController class */

#import "PlotController.h"
#import "PlotView.h"
#import <appkit/Application.h>
#import <appkit/Listener.h>
#import <appkit/Pasteboard.h>
#import <appkit/Window.h>
#import <appkit/ScrollView.h>
#import <appkit/Text.h>
#import <stdlib.h>
#import <string.h>
#include <mach.h>

@implementation PlotController

- appDidInit:sender
/*
 * Responds to a message that's sent just before the application
 * gets the first event.  PlotController initializes its inputText
 * instance variable.  It makes itself the inputText's delegate and
 * the services delegate for the application.  It also ensures that
 * the application's single window becomes the key window.
 */
{
    inputText = [theScrollView docView];
    [inputText setDelegate:self];
    [[NXApp appListener] setServicesDelegate:self];
    [[thePlotView window] makeKeyAndOrderFront:self];
    return self;
}

- textDidGetKeys:theText isEmpty:(BOOL)flag
/*
 * Responds to a message the Text object sends when its text changes.
 * PlotController causes the PlotView to clear itself when the text changes.
 */
{
    return [thePlotView clear:self];
}
```

```
- requestPlot:(id)pasteboard userData:(const char *)userData error: (char **)msg
/*
 * Responds to a request from another application for the plotting service.
 * PlotController copies the data from the supplied pasteboard into the Text
 * object and then sends a plot: message to the PlotView.  (Note:  In
 * services.txt, Plotter advertises this method simply as requestPlot since
 * the other elements of the method name are invariant.)
 */
{
    char  *data, *scratch;
    int    length;

    [NXApp activateSelf:NO];
    [pasteboard types];
    if ([pasteboard readType:NXAsciiPboardType data:&data length:&length]) {
        if(scratch = malloc(length+1)) {
            strncpy(scratch, data, length);
            scratch[length]='\0';
            [[inputText selectAll:self] replaceSel:scratch];
            free(scratch);
        }
        vm_deallocate(task_self(), (vm_address_t)data, (vm_size_t)length);
        [thePlotView plot:self];
    }
    return self;
}


- plotView:sender providePoints:(NXStream **)stream
/*
 * Responds to a message the PlotView sends requesting the points to plot.
 * PlotController responds by giving the PlotView access to the Text
 * object's stream.
 */
{
    *stream = [inputText stream];
    return self;
}

- plotView:sender pointDidChange:(NXPoint *)aPoint
/*
 * Responds to a message the PlotView sends notifying its delegate
 * that a point has been added.  PlotController responds by adding the point
 * to the end of the list in the Text object.
 */
{
    int   endPos;
    char  buffer[100];

    sprintf(buffer, "%d, %d\n", (int)aPoint->x, (int)aPoint->y);
    endPos = [inputText byteLength];
    [inputText setSel:endPos :endPos];
    [inputText scrollSelToVisible];
    [inputText replaceSel:buffer];
    return self;
}

@end
```

drawingFuncs.psw

```
defineps loadPSProcedures()
    /xpos 0 def
    /ypos 0 def
    /ticklength [8 2 2 2 2 5 2 2 2 2] def

    % x
    /getticklength {10 idiv 10 mod abs ticklength exch get} def


    % x y length
    /vtick { moveto 0 exch rlineto stroke } def
    /htick { moveto 0 rlineto stroke } def

    % x
    /increase-x {/xpos xpos 10 add def} def
    /decrease-x {/xpos xpos 10 sub def} def

    % y
    /increase-y {/ypos ypos 10 add def} def
    /decrease-y {/ypos ypos 10 sub def} def
endps

defineps drawCircle(float x, y, radius)
    % describe the path of a circle
    newpath
    x y radius 0 360 arc
    closepath
    stroke
endps
```

```
defineps drawAxes(float x, y, width, height)
    gsave
        % set max and min of visible area
        /xmax x width add 1 sub def
        /ymax y height add def
        /ymin y 1 add def
        /xpos 0 def
        /ypos 0 def

        % draw border
        x ymin moveto
        x ymax lineto
        xmax ymax lineto
        xmax ymin lineto
        closepath
        stroke

        % now draw axes
        x 0 moveto
        xmax 0 lineto
        stroke
        0 ymin moveto
        0 ymax lineto
        stroke
        0 0 moveto
        % draw the pos x ticks
        {xpos xmax le
            {xpos getticklength neg xpos 0 vtick increase-x}{exit} ifelse
        } loop
        % draw the neg x ticks
        0 0 moveto
        {xpos x ge
            {xpos getticklength neg xpos 0 vtick decrease-x}{exit} ifelse
        } loop
        % draw the pos y ticks
        0 0 moveto
        {ypos ymax le
            {ypos getticklength neg 0 ypos htick increase-y}{exit} ifelse
        } loop
        % draw the neg y ticks
        0 0 moveto
        {ypos ymin ge
            {ypos getticklength neg 0 ypos htick decrease-y}{exit} ifelse
        } loop
    grestore
endps
```

Plotter_main.m

```
/*
 *    Plotter_main.m -- Generated by the NeXT Interface Builder.
 */

#import <stdlib.h>
#import <appkit/Application.h>

void main(int argc, char *argv[]) {
    NXApp = [Application new];
    [NXApp loadNibSection:"Plotter.nib" owner:NXApp];
    [NXApp run];
    [NXApp free];
    exit(0);
}
```

services.txt

```
Message: requestPlot
Port: Plotter
Send Type: NXAsciiPboardType
Menu Item: Plot
```

Makefile.preamble

```
LDFLAGS = -sectcreate __ICON __services services.txt
```

Makefile

```
# Makefile
#
# Generated by the NeXT Interface Builder.
#
# NOTE: Do NOT change this file -- Interface Builder maintains it.
#
# Put all of your customizations in files called Makefile.preamble
# and Makefile.postamble (both optional), and Makefile will include them.
#

NAME = Plotter

INTERFACES = Plotter.nib
CLASSES = PlotController.m PlotView.m
MFILES = Plotter_main.m
PSWFILES = drawingFuncs.psw
TIFFFILES = cross.tiff
APPICON = PlotterIcon.tiff


SOURCEMODE = 444

LIBS = -lNeXT_s -lsys_s
DEBUG_LIBS = $(LIBS)
PROF_LIBS = -lNeXT_p -lsys_p

MAKEFILEDIR = /usr/lib/nib
ICONSECTIONS =  -sectcreate __ICON app PlotterIcon.tiff

INSTALLDIR = $(HOME)/Apps
INSTALLFLAGS = -c -s -m 755

-include Makefile.preamble

include $(MAKEFILEDIR)/app.make

-include Makefile.postamble

-include Makefile.dependencies
```