FCS of OpenStep Developer for Mach and Pre-release 3 of OpenStep Developer for Windows

This source will build either on OpenStep for Mach or OpenStep for Windows.

Note that a major change (besides OpenStep-ification) is that Draw now saves its files out in ASCII Property List format instead of archiving subclasses of View. While ASCII Property Lists are not a very efficient format for storing files, they are very good for debugging and such. Also, archiving objects as part of your document format is probably not such a good idea because you can then only provide backwards compatibility, not forwards compatibility. With a property list, even older versions of the application could make some sense of a file written by a newer version of the application.

About Draw

This file contains some useful information which should make it easier to understand what the program does and how it works.

Don't be put off by the size of this application. The functionality is largely compartmentalized (e.g. the NSImage-related stuff is in Image.m, the Application-related stuff is in DrawApp.m, the window delegate stuff is in DrawDocument.m, the Pasteboard-related stuff is in gvPasteboard.m, the Object Links stuff is mostly in gvLinks.m, etc.). Feel free to cut and paste code out of the source of this program. For the most part, the code is fairly simple, it's just that this program does a lot of stuff one wants to do in a multi-document application.

This document starts with an overview of all the classes in the application followed by a list of the files in the application directory and a description of each.

Next, there are some notes about the application including an overview of its functionality.

A summary of many of the NeXTSTEP features exemplified in the Draw application is listed as well as a pointer to where to look in the application to see example code.

Major classes in the application

GraphicView	The heart of the functionality of the program. A subclass of the AppKit's View class. Manages any number of Graphic (class) objects, allowing selection, grouping, printing, resizing, etc. Essentially independent of all other classes in the application except the Graphic's.
Graphic	The base class for the objects manipulated in a GraphicView. The vast majority of the functionality of the Graphics is contained in this class.
DrawDocument	This class manages the external representation of a GraphicView. This includes saving the GraphicView to disk as well as managing the window the GraphicView is in.
DrawApp	Application subclass which manages global state. Primary functions are to create new documents and take care of the application-wide tool palette.
Inspector	Used to edit the Graphic objects. Currently has only one form (i.e. no subclasses) and only allows modification of any attributes that the base Graphic class has (fortunately, this is almost everything). In a more complete implementation, this class would be subclassed for specific Graphics.
DrawPageLayout	A subclass of the PageLayout panel. Customizes the panel to allow the user to specify the margins of the page to be printed on.
GridView	The active area of the Grid Inspector modal panel. Since the interesting part of the panel is the custom view, the workings of the panel are collected with it into this class.
Image	This allows importing of TIFF or EPS images (from the Pasteboard or from a file).
Group	A subclass of Graphic which can contain other graphics inside of it. This is how grouping is done.
TextGraphic	A subclass of Graphic which allows editing of rich text using the Text object.

	This is an excellent place to look if you are interested in using the Text object to edit text in arbitrary locations in a View.
Rectangle	The simplest of Graphic subclasses.
Circle	Another very simple Graphic subclass.
Line, Curve	Curve is a subclass of Line which is a subclass of Graphic. These two are also fairly simple and show how two classes can share a lot of code via inheritance.
Polygon, Scribble	Polygon is a subclass of Scribble which is a subclass of Graphic. These two are an example of using user paths to draw many line segments (the drawing of the grid is another good example of that).
SyncScrollView, Ruler	SyncScrollView is a subclass of the Application Kit's ScrollView which adds two Ruler views to the Draw document and keeps the Ruler views in sync as the document is scrolled around. This should be replaced in OpenStep with the new NSRuler API.

Other files

Draw.nib	The user-interface of the application.
InfoPanel.nib	The panel which comes up when the user clicks in the Info... menu item
InspectorPanel.nib	The Inspector panel.
GridView.nib	The Grid Inspector panel.
gv*.m	GraphicView categories. Some of the functionality of GraphicView is broken out into separate files for easier understanding.
draw.psw	pswraps used by some of the Graphics.

*.tiff	Cursors which appear depending on the tool the user is using.
Draw_main.m, PB.project, Makefile	Created by Project Builder
Makefile.preamble	Sets up draw.p as a precompiled header.
Makefile.postamble	Defines header dependencies for the precompiled header.
*.rtf	Various programmer notes about such things as Object Links, Dragging, Undo, etc.

Notes

This program is by no means an ideal implementation of a drawing program. It is intended to give example code for as many features available in NeXTSTEP as is possible in a single application.

The program makes heavy use of the First Responder mechanism to simplify the code.

In trying to understand how the program works, it is important to take a look at the Interface Builder files (Draw.nib, et. al.) and see where messages are being sent. That is as much a part of understanding the program as understanding what the messages do on the receiving end.

The most important thing to remember when examining this program is that it was written purely as an example of how to do things using NeXTSTEP. All methods are commented with an explanation of what the method does and what its place in the application is.

The methods in the objects are grouped functionally (e.g. all window delegate methods are grouped, all target/action methods are grouped, etc.) and general concepts are explained (where appropriate) in the comments at the beginning of the .m files.

Overview of the program's functionality

Essentially the program allows manipulation of simple graphical objects. The objects can be sized, moved and grouped. Each object's attributes (e.g. whether a circle is filled or the color of the characters in a line of text) is changeable via the Inspector and the Text and Font menus.

The application can edit any number of documents. Each document represents a piece of paper. The documents can be saved to disk and reopened by double-clicking on its icon in the Workspace (or via the OpenPanel). The documents can also be printed.

PostScript and TIFF files can be incorporated into a document simply by dragging an icon representing such a file from the Workspace into a document window. Once incorporated, the images can be scaled and moved as any other Graphic.

The document can be saved either as a draw document, an encapsulated PostScript file (.eps) or as a TIFF file (.tiff) (though the program can only read in a document in draw format).

The program supports full cut/copy/paste and can even copy and paste PostScript, TIFF and Text to/from other applications. In fact, the Draw program is an excellent way to convert images from one format to another.

Some of the objects which can be created include ovals, rectangles, straight lines, freehand drawing, polygons and text. The font of the text can be modified via the font menu and FontPanel. The text is editable at any time by clicking on it while in the Text tool. Resizing the bounding box of the text will cause it to rewrap to the new bounding box.

The Inspector allows the user to add arrows at the end of lines, set the width of lines, set closed paths to be filled, set the gray used to fill a closed path or draw a line, set the line cap and line join attributes, etc.

Topics of interest exemplefied in the Draw program:

Property Lists (propertyList.m and initFromPropertyList: in GraphicView, Graphic and subclasses)
Cross platform portability (all of Draw!)

Object Links (gvLinks.m, Links.rtf)
Services (gvServices.m, DrawDocument)

Undo (*.subproj)
Using NSImage (Image)
Making an application localizable (NSLocalString calls everywhere)
Floating panels (DrawApp's applicationDidFinishLaunching:)
Drag and drop colors (gvDrag.m)
Autosaving window locations (DrawApp's applicationDidFinishLaunching:)
File packages (DrawDocument)
Optimized NSRectFillList() (Graphic's fastKnobFill:)
Dragging icons for files into a document (gvDrag.m, Dragging.rtf)
Using the Pasteboard (gvPasteboard.m)
Lazily providing Pasteboard data (pasteboard:provideDataForType: in gvPasteboard.m)
Project Builder project management
Reading and imaging bit (TIFF) images (Image, gvPasteboard.m)
Cutting and pasting PostScript & TIFF between applications (gvPasteboard.m)
Cutting and pasting an internal format (gvPasteboard.m)
Creating Fax Cover Sheets (TextGraphic, DrawDocument, DrawApp)
Spell-checking (GraphicView's checkSpelling:)
Protocols (SyncScrollView.h)

Review Unsaved functionality on Application quit
Using zones (DrawApp's panels, DrawDocument, gvPasteboard)
Maintaining mulitple documents via the First Responder mechanism
Synchronized scrolling views (SyncScrollView)
Rulers (SyncScrollView, Ruler)
Loading interface objects from nib files (DrawApp, others)
Opening documents from the user-interface (DrawApp application:openFile:)
Compositing (GraphicView drawRect: and compositeSelection:)
ScrollView mouse tracking cursors (DrawDocument resetCursor)
Autoscrolling (GraphicView's move: and Graphic's resize:)
Modal tracking loops (GraphicView, Graphic, GridView)
Using tracking timers (Graphic, GraphicView)
Using pswraps (draw.psw and all the Graphic subclasses)
PostScript user paths (Scribble class and GraphicView's resetGUP)
Using inheritance (Graphic and its subclasses)
Methods as arguments to functions (graphicsPerform:andDraw:, saveTo:using:)

Off-screen cacheing (GraphicView)
Automatic menu updating (validateMenuItem:)
Window delegate methods (constrained window resizing, etc.) (DrawDocument)
Running a panel modally (Grid Inspector)
Customizing the SavePanel (DrawApp saveToPanel:)
Customizing the PageLayout panel (DrawPageLayout)
Using the PageLayout panel (DrawDocument changeLayout:)
Opening multiple files with single OpenPanel invocation (DrawApp open:)
Using Alerts
Defaults mechanism (DrawApp)
Peeking at incoming events to control global behaviour (DrawApp sendEvent:)
Using PopUpLists (Inspector)
Intercepting key presses (GraphicView keyDown:)
Reflecting edited document state (GraphicView's dirty, DrawDocument's dirty:)

... and much much more!

The following AppKit classes are subclassed:

View	(GraphicView, GridView)
Application	(DrawApp)
PageLayout	(DrawPageLayout)
Panel	(InspectorPanel)