Documentation for Plot.h, Plot.c, and PlotX.c

Steven Andrews, © 2003
See the document "LibDoc" for general information about this and other libraries.

```
void InitPlot();
void EndPlot();
void MakeWindow(float left,float right,float top,float bottom);
void KillWindow();
void InvalPlot();
void SetScales(float xa,float xb,float ya,float yb);
void GetScales(float *xa,float *xb,float *ya,float *yb);
void ScalePtrs(float **xa,float **xb,float **ya,float **yb);
void SetColor(char c);
void ToPixel(float x,float y,int *a,int *b);
int ToPoint(int a,int b,float *x,float *y);
void DrawAxes();
void DrawMarks(float dx,float dy,int lo,int hi);
void PlotClear();
void PlotMove(float x,float y);
void PlotPt(float x,float y);
void PlotLine(float x,float y);
void PlotStr(float x,float y,char *s);
void ShowLimits();
void PlotFn(float (*fn)(float),float dx);
void PlotInt(float (*fn)(float),float x0,float dx);
void PlotData(float *data,int n,float xmin,float xmax,int style);
void PlotData2(float *data,float *x,int n,int style);
```

Requires: <math.h>, <Types.h>, <Memory.h>, <Quickdraw.h>, <Fonts.h>, <Events.h>, <Menus.h>, <Windows.h>, <TextEdit.h>, <Dialogs.h>, <OSUtils.h>, <ToolUtils.h>, <SegLoad.h>, <Sound.h>, <limits.h>, <stdio.h>, "Plot.h", "random.h", "Utility.h". Most of these aren't used.

Example Program: SpectFit.c.

Originally written before 8/96. Modified for Metrowerks C 9/98. Designed to work on a Mac G3. Moderate testing. Works on a Mac G4. Works on OS X, and, hopefully, still on OS 9. Added `PlotMove` 8/25/02.

These functions implement most of what is needed to draw pictures and graphs from a program, without having to directly deal with the Macintosh toolbox routines and without having to worry about details. These are easy to use, but inflexible and inelegant.

Programs that use graphics can be divided into sequence driven programs and event driven programs. The former ones, which are easier for a quick program or for simple scientific visualization, draw to the graphics window whenever they want to. Event driven programs have a list of things that are supposed to be displayed, and that are refreshed or manipulated as indicated by either user or program generated events. This library can work with either program style, although it was constructed with the former more in mind.

A program should first call `InitPlot` to initalize the relevant toolbox managers. Then, it will generally want to call `MakeWindow` to create an output window. A call to

SetScales defines the values at the edges of the window. If SetScales isn't called, default values of xa=ya =–10 and xb=yb=10 are used. Close the window with KillWindow.

If you are using sequence driven programming, no other infrastructure commands are needed, and you can just draw. On the other hand, event driven programming can be done with InvalPlot to add the entire output window to the update region, which triggers an update event. When the drawing is done, EndPlot tells the Window manager to start accruing new update events.

Functions

InitPlot initiallizes the Toolbox managers. More are initiallized than neccessary, and could be cut out.

EndPlot tells the window manager to start accruing update events.

MakeWindow creates an output window. The variables left, right, top, and bottom are give the window dimensions, in fractional terms, where the full screen is considered to have size 1x1. For example, an almost full screen window would be called with MakeWindow(0.1,0.9,0.1,0.9). The window has no scroll bars or size box.

KillWindow disposes of the output window.

InvalPlot adds the entire output window to the update region to trigger an update event.

SetScales defines the values of the edges of the window. xa, xb, ya, and yb are the left and right *x* values and bottom and top *y* values, respectively. Window coordinates vary linearly within the edges. It should be permissable to define xb<xa and/or yb<ya, but this has not been tested much.

GetScales returns the current scale values.

ScalePtrs is almost identical to GetScales, but returns the addresses of the variables for the window edges rather than their values.

SetColor sets the current drawing color, which will be used for all future commands, until SetColor is called again. This routine only supports 28 colors, making it fairly crude but also fairly convenient. The colors were chosen to be similar to world wide web colors. Also, numbered colors, and the + and - ones, match the standard resistor color coding. If the input character is none of those below, black is used.

Color codes

A	aqua	K,0	black	U	ultraviolet
B,6	blue	L	lime	V,7	violet
C	cyan	M	magenta	W,9	white
D	dark red	N	navy	X	random
E,8	grey	O,3	orange	Y,4	yellow
F	fuchsia	P	purple	Z	random
G	green	Q	quartz	1	brown
H,5	hunter green	R,2	red	-	silver
I	indigo	S	sienna	+	gold
J	olive	T	teal		

`ToPixel` converts *x* and *y* positions to pixel coordinates; *x* and *y* are sent in as `x` and `y`, while the returned coordinates are `a` and `b`.

`ToPoint` is the inverse routine, converting pixel coordinates `a` and `b` to point coordinates `x` and `y`. If the point is outside the visible region, the function returns 0; otherwise it returns 1. These functions are primarily for use within the library, but may also be called from elsewhere.

`DrawAxes` draws *x* and *y* axes on the window, which extend from edge to edge and cross at the origin; if an axis is off the visible region of the window, it doesn't show up.

`DrawMarks` puts tick marks on the axes, with *x* axis spacing `dx` and *y* axis spacing `dy`. The marks extend from `lo` pixels below the axis to `hi` pixels above the axis.

`PlotClear` erases the entire drawing window.

`PlotMove` moves the drawing pen to the coordinates given but does not draw anything.

`PlotPt` draws a single pixel dot at the coordinates given.

`PlotLine` draws a line from the previous drawing location to the coordinates given.

`PlotStr` displays a string at position `x`,`y`. It uses 9 point size and the current font and text style, which is plain geneva, as a default.

`ShowLimits` writes the *x* and *y* values of the lower left corner and the upper right corner on the graphics window, near the corners.

`PlotFn` graphs a function of a single float value; the function is calculated at intervals `dx` within the domain of the window.

`PlotInt` graphs the integral of a function of a single float value. The integral is defined to be 0 at *x*=`x0` and is calculated at intervals `dx`.

`PlotData` draws an *x*-*y* graph of uniformly spaced data. The data is passed to the routine with `data`, which is an array of `n` floats, which correspond to uniformly spaced *x* values from `xmin` to `xmax`. The graph style is given with the `style` parameter: 1 gives a bar graph, 2 gives scattered points, and 3 gives a line graph.

`PlotData2` is just like `PlotData`, but inputs a vector of `x` values that correspond to the `data` values, thus allowing unevenly spaced data, or scrambled data.