PHPlot Reference Manual

The PHPlot Documentation Team

L J Bayuk
Primary author and editor of the manual, and current maintainer of PHPlot
Miguel de Benito
Developer and maintainer of PHPlot
Afan Ottenheimer
Original developer of PHPlot

PHPlot Reference Manual

The PHPlot Documentation Team

by L J Bayuk, Miguel de Benito, and Afan Ottenheimer

Release 2011-01-15 for PHPlot-5.3.1

Copyright © 2005-2011 The PHPlot Documentation Team

Table of Contents

Preface

PHPlot is a PHP class for on-the-fly graphs generation. It was started by Afan Ottenheimer in 2000 as an Open Source project, and is developed on SourceForge [http://sourceforge.net]. It is distributed[1] under the terms of the GNU Lesser General Public License [http://www.opensource.org/licenses/lgpl-2.1.php] version 2.1.

Here are some of the features in PHPlot:

- Creates many graph types, including line plots, point plots, bar charts, and pie charts, also data/error graphs.

- Draws 3-D shading of pie charts and bar graphs.

- Customizable line color, width, solid or dashed patterns.

- Flexible labels, tick marks, axes, titles, legend, grid lines.

- TrueType fonts are supported, but not required.

- Use linear or logarithmic axes.

- Image output uses the GD Library, with supported formats including PNG, GIF, and JPEG.

PHPlot's home page is at http://phplot.sourceforge.net [http://phplot.sourceforge.net/], and project development takes place at http://sourceforge.net/projects/phplot/ [http://sourceforge.net/projects/phplot/].

---

[1] Versions prior to PHPlot-5.0.7 were distributed under the GPL and PHP licenses.

Part I. PHPlot Programming

This first part of the PHPlot Reference Manual includes instructions on installing and using PHPlot. There is a chapter to help get you started, and a chapter of examples.

Chapter 1. PHPlot Installation

This chapter explains how to install PHPlot.

1.1. Prerequisites

Before you can use PHPlot, you need a recent version of PHP with the GD extension. PHPlot-5.0.4 and earlier require at least PHP version 4.3.0, and PHPlot-5.0.4 was tested with PHP-4.4.7 and PHP-5.2.4. PHPlot-5.0.5 and up will only support PHP5, since PHP4 reached its official end-of-life as of the end of 2007. In general, you should always use the latest available version of PHP5.

If you want to display PHPlot charts on a web site, you need a PHP-enabled web server. You can also use PHPlot with the PHP CLI (command line interface) without a web server.

You need the GD extension to PHP either built in to PHP or loaded as a module. The version of GD which is bundled with recent versions of PHP will work fine.

If you aren't sure what extensions you have in PHP, create this PHP script called `phpinfo.php` somewhere in your web server's document tree:

```
<?php phpinfo(); ?>
```

Access this page with your browser to view your PHP configuration. Look for the section labeled 'gd'. If you have no 'gd' section in the PHP Info output, you don't have the required 'gd' extension. Consult the PHP documentation to find out how to get it. Here's what the GD section of the PHP Info listing might look like:

gd

GD Support	enabled
GD Version	bundled (2.0.34 compatible)
FreeType Support	enabled
FreeType Linkage	with freetype
FreeType Version	2.3.9
T1Lib Support	enabled
GIF Read Support	enabled
GIF Create Support	enabled
JPEG Support	enabled
libJPEG Version	6b
PNG Support	enabled
libPNG Version	1.2.37
WBMP Support	enabled
XBM Support	enabled

The text in the 'gd' section of the PHP Info output will tell you what version of GD you have (you need 2.0 or higher), and what output formats it supports. Check the table for *PNG Support*, since PNG is the default output format for PHPlot. If you want to create GIF or JPEG (JPG) format images, check the table to see if they are available. If the image formats you want are not available, you will have to rebuild PHP.

Another thing to look for in the 'gd' section of the PHP Info output is *FreeType Support*. If you have it enabled, you can use TrueType fonts in PHPlot. If your GD does not have FreeType support enabled, you can still make decent-looking

plots with PHPlot, using the built-in GD fonts. Note that even if you have FreeType Support enabled, you need some actual TrueType font files in order to use TrueType fonts with PHPlot. PHPlot does not include any TrueType font files.

While you have the PHP Info report up, look in the Configuration section for PHP Core, at the top of the report, and make a note of the `include_path` setting. If you have a local include directory in this path, you can use it for installing PHPlot, as described below.

Note

Don't leave the `phpinfo.php` file in your web server document tree, as there may be security implications in the information it tells people about your web server.

Once you have a web server with PHP and the GD extension, you are ready to install PHPlot.

1.2. Installing

Unpack the PHPlot distribution into a convenient directory. Recent releases of PHPlot are available both as ZIP files, and as gzip-compressed TAR files. Use whichever format is more convenient for you. In the example below, the TAR format is unpacked.

```
$ tar -xvzf phplot-5.*.tar.gz
$ cd phplot-5.*
```

(Use the appropriate filename.) Check the distribution for README and/or INSTALL files which may contain newer instructions.

Installation of PHPlot simply involves copying the script files somewhere your PHP application scripts will be able to find them. The scripts are: `phplot.php`, the main script file, and `rgb.inc.php`, an optional script file containing a large color map. Make sure the protections on these files allow the web server to read them. For example:

```
$ chmod 644 *.php
```

Then, simply copy the files into a directory where PHP scripts will be able to include them. The ideal place is a directory outside your web server document area, and on your PHP include path (that you noted above in the PHP Info report). You can add to the include path in the PHP configuration file; consult the PHP manual for details. For example, if `/usr/local/share/php` is on your PHP include path, you can install PHPlot with:

```
$ cp phplot.php rgb.inc.php /usr/local/share/php
```

1.3. Next Step

You can test PHPlot with any of the examples in Chapter 5, *PHPlot Examples*, or by entering this minimal script into a file called (for example) `plottest.php`.

```php
<?php
require 'phplot.php';
$data = array(array('', 10), array('', 1));
$plot = new PHPlot();
$plot->SetDataValues($data);
$plot->SetTitle('First Test Plot');
$plot->DrawGraph();
```

Access this script through your browser, and you should see a very simple plot. Note: Since PHPlot returns image data, not text, you will generally not see error messages in the output. If a script using PHPlot has a syntax error, or calls

an undefined function, you will get a blank page returned, and you will probably have to check the web server error log for the reason. You might find that debugging your PHPlot applications is easier using the PHP CLI (command line interface), as described at the start of Chapter 5, *PHPlot Examples*.

If you are installing PHPlot for use by some web application (rather than to develop your own applications), proceed with that application's setup instructions. If you want to develop your own applications using PHPlot, you can start by looking at some of the examples in Chapter 5, *PHPlot Examples*, or go right to the introductory material in Chapter 2, *Getting Started with PHPlot*. Experienced programmers may want to skip right to Chapter 3, *PHPlot Concepts* to learn about PHPlot concepts and features in depth.

Chapter 2. Getting Started with PHPlot

This chapter will help you get started with PHPlot.

The material in this chapter was originally from the PHPlot Quick Start and Examples document, by Afan Ottenheimer and Miguel de Benito, distributed with PHPlot. It has undergone much editing and any mistakes are not their fault.

2.1. Introduction

Many web sites need to create real-time or dynamic charts and graphs from live data sets. Many users have found PHP a great way for this dynamic creation of images using the GD library. The advantage of using the server to create an image (that is, using a server-side scripting language rather than a client-side language such as Java) is that one does not have to worry about browser compatibility or client operating system compatibility issues.

PHPlot is a specialized graphics library which provides a means for your PHP-enabled web server to create and manipulate graphs as objects and display the completed graph as an image. PHPlot uses the GD library to create elementary shapes such as lines, rectangles, and text, but hides the details of GD from the application programmer.

Data sets are passed to PHPlot using PHP arrays, in a convenient format for database-driven sites.

First, lets discuss how PHPlot works in general with some terminology. (More complete definitions can be found in Section 3.1, "Definitions".) A PHPlot *image* can consist of several *graphs* (but usually has only one), each graph consisting of several *elements* (like lines, axes, and labels).

To use PHPlot, you create a PHP object from the PHPlot class, for example:

```
$plot = new PHPlot;
```

Then you set the properties of the object, by using a series of function calls (actually methods of the class). These define the characteristics of the image, the graph or graphs, and their elements. This includes setting the array containing the data to be plotted, defining titles if you want them, and many optional elements and style settings. You can think of this as "drawing" elements into the image, but in fact PHPlot just notes your intentions and doesn't do much until you are finished.

When you are done describing a graph, you instruct PHPlot to "draw" the graph into the image. When you are done with all graphs in an image, you need to instruct PHPlot to "print" (output) the image. Since most images contain only one graph, PHPlot simplifies this process by default. Unless instructed otherwise, PHPlot will "print" the image (output it to the browser) as soon as you tell it to "draw" (render) the first graph.

Usually, PHPlot will "print" the image directly to the user's browser. The result will be a complete HTTP response with headers, so your PHP script must not produce any other output (except for optional headers). The user will be see the image either as a result of accessing your script directly with a URL, like `http://www.example.com/graphs/myplot.php`, or you can embed the image in a web page using an image tag, like this:

```
<IMG SRC="http://www.example.com/graphs/myplot.php">
```

Instead of sending the image to the browser, your application can instead choose to write the PHPlot image to a file on the server. This could be useful if you want to implement server-side caching of image files. (PHPlot itself does not currently provide caching.)

Before continuing, we need to mention coordinates. PHPlot uses two coordinate spaces: one for the image, and one for the data you are plotting. *World Coordinates* are the X,Y coordinates of your data, relative to the axis origin, in the units of the data sets. Your data array uses world coordinates, as do tick mark labels on the X and Y axis. *Device Coordinates* measure pixels in the image according the the GD convention, with the origin in the upper left corner of the image. These are also called Pixel Coordinates. PHPlot tries to place elements on your graph appropriately, but if you need to override its choices you will use device coordinates to position the elements.

The rest of this chapter explains how to write a PHP script which creates a plot using PHPlot. Information on PHP can be found at www.php.net [http://www.php.net/]. Information about the GD library which PHP uses to create images can be found at libgd.org [http://libgd.org/]. More information about PHPlot can be found at phplot.sourceforge.net [http://phplot.sourceforge.net/].

2.2. Creating the Object

You create a PHPlot object by first including the code to be used, and then defining the variable:

```php
<?php
require_once 'phplot.php';  // here we include the PHPlot code
$plot = new PHPlot;    // here we define the variable

//Rest of code goes below
```

The above code assigns the PHPlot object to the variable `$plot`.

2.3. A Simple Graph

We will start with a simple line graph.

```php
<?php
//Include the code
require_once 'phplot.php';

//Define the object
$plot = new PHPlot();

//Define some data
$example_data = array(
    array('a',3),
    array('b',5),
    array('c',7),
    array('d',8),
    array('e',2),
    array('f',6),
    array('g',7)
);
$plot->SetDataValues($example_data);

//Turn off X axis ticks and labels because they get in the way:
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');

//Draw it
$plot->DrawGraph();
```

And that's it! What we get is the following graph:
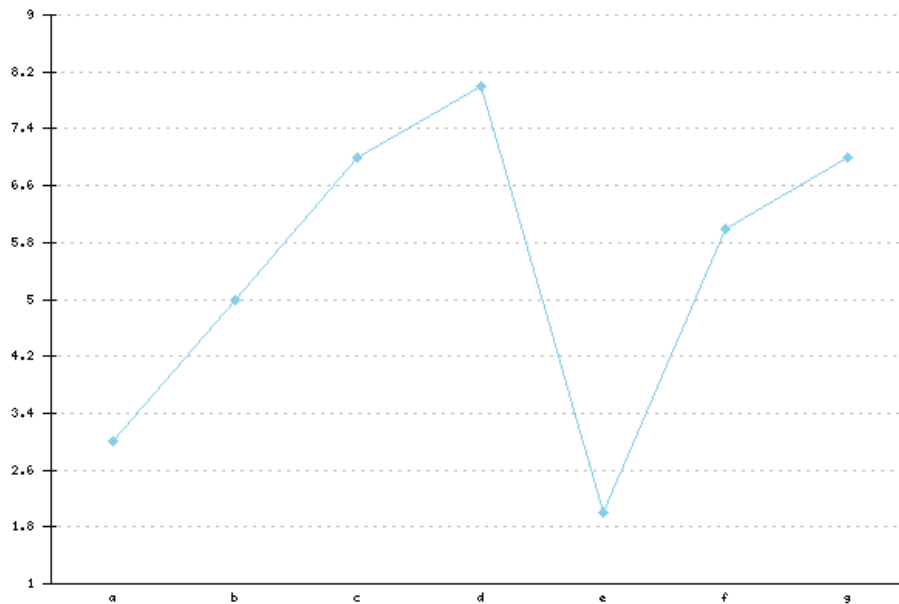
That's a great start, but now we would like to specify the width and height of the image, and add some titles.

2.4. Different Size Images and Titles

Let's say we want our plot to be bigger, 800 pixels wide and 600 pixels high. So instead of having the line

```
$plot = new PHPlot;
```

We replace it with

```
$plot = new PHPlot(800,600);
```

and you have specified the size in pixels of the image to be created.

A couple of things to note:

- The default is not to use TrueType fonts.

- Since there was only one graph on the image, we didn't have to use PrintImage. DrawGraph took care of it for us.

- We did not specify the data type. If you do not specify the data type, PHPlot assumes `text-data`.

- We did not specify the file type (GIF, PNG, JPEG, ...). PHPlot 5.0 (and newer) makes PNG images by default.

- The data is passed in as an array of arrays. This may seem awkward now, but as we add functionality this will be beneficial.

OK, now we're ready to add some customization to the plot. Let's change the size, the title and the X/Y axis labels. All we need to do is use additional methods of the `$plot` object before printing the image. Here is the complete result:

```
<?php
//Include the code
require_once 'phplot.php';
```

```
//create a PHPlot object with 800x600 pixel image
$plot = new PHPlot(800,600);

//Define some data
$example_data = array(
    array('a',3),
    array('b',5),
    array('c',7),
    array('d',8),
    array('e',2),
    array('f',6),
    array('g',7)
);
$plot->SetDataValues($example_data);

//Set titles
$plot->SetTitle("A Simple Plot\nMade with PHPlot");
$plot->SetXTitle('X Data');
$plot->SetYTitle('Y Data');

//Turn off X axis ticks and labels because they get in the way:
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');

//Draw it
$plot->DrawGraph();
```

And that's it! What we get is the following graph:
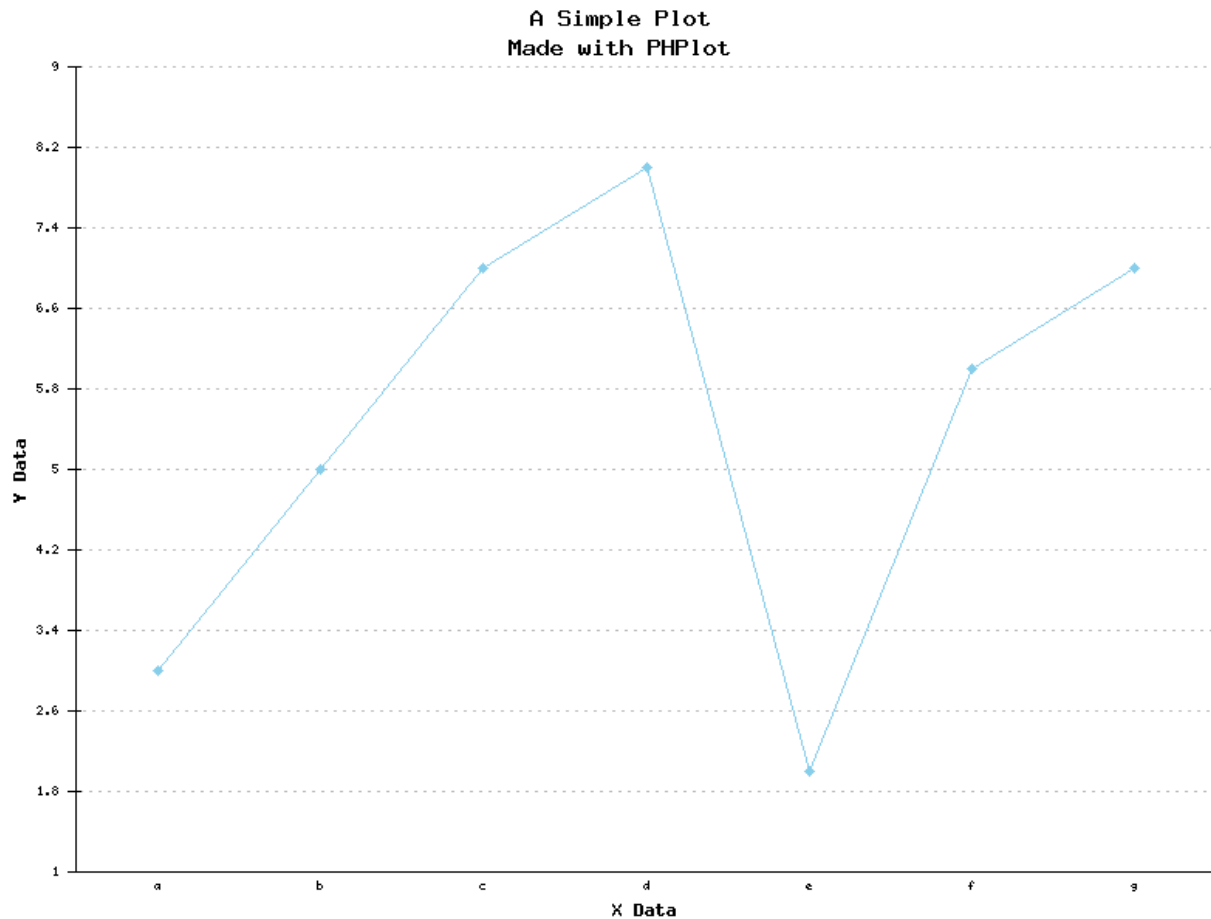
Note that the newline character "\n" separates multiple lines in titles, and you must use double quotes around the title string for PHP to recognize the newline.

2.5. Multiple Lines Per Graph

Let's say we want to plot not just one dataset, but several Y values for each X position. With PHPlot, it is easy to specify the multiple data lines by just passing in all the Y values for a given X value at once. So instead of `array('label', y)`, we specify `array('label', y₁, y₂, y₃, ...)`. This is very convenient when working with rows of data from databases.

Now our data will have three Y values for each position on the X axis.

```php
<?php
//Include the code
require_once 'phplot.php';

//Define the object
$plot = new PHPlot(800,600);

//Set titles
$plot->SetTitle("A 3-Line Plot\nMade with PHPlot");
$plot->SetXTitle('X Data');
```

```
$plot->SetYTitle('Y Data');


//Define some data
$example_data = array(
      array('a',3,4,2),
      array('b',5,'',1),   // here we have a missing data point, that's ok
      array('c',7,2,6),
      array('d',8,1,4),
      array('e',2,4,6),
      array('f',6,4,5),
      array('g',7,2,3)
);
$plot->SetDataValues($example_data);

//Turn off X axis ticks and labels because they get in the way:
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');

//Draw it
$plot->DrawGraph();
```

Which gives us:
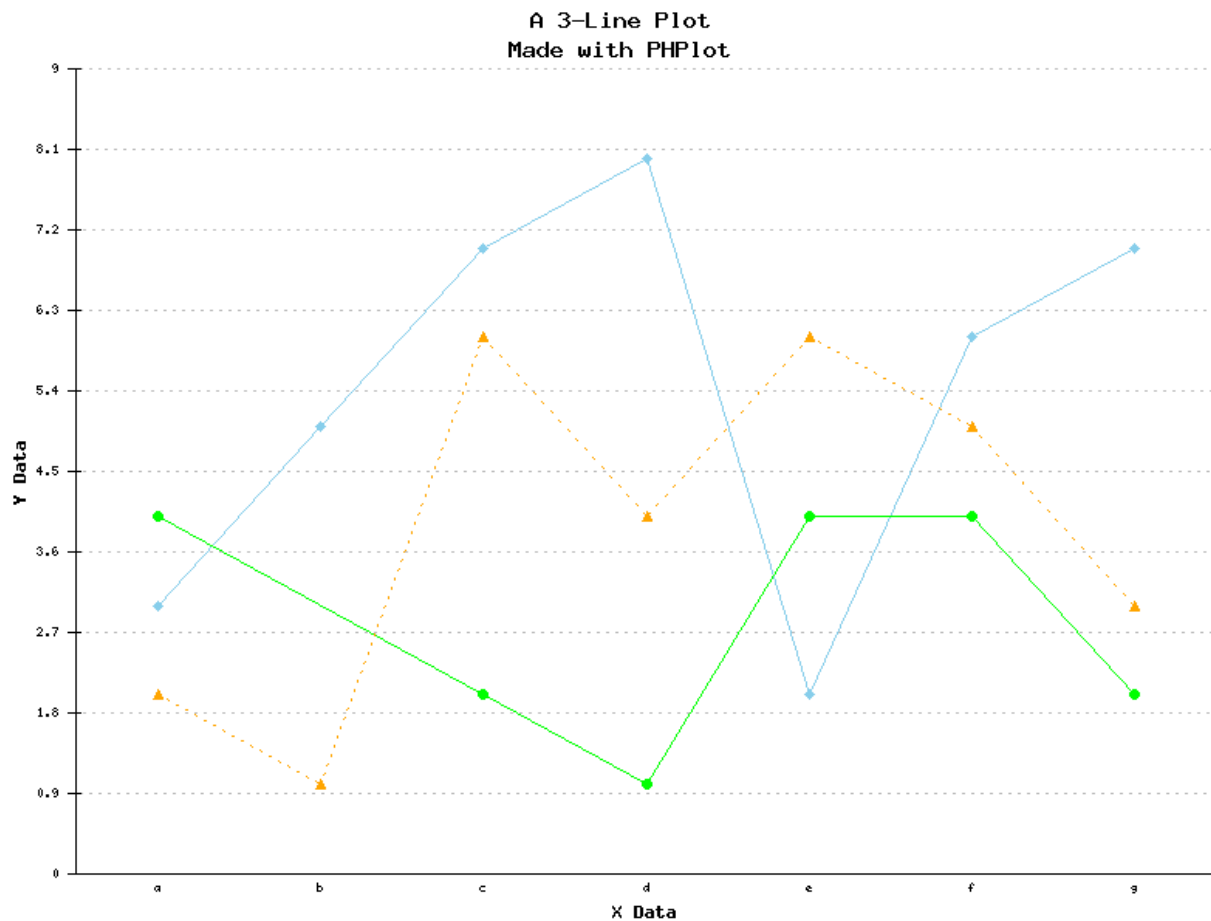
Notice that each set of Y data gets a different color. Also the missing data point (label 'b' on the green line) is skipped. This behavior can be adjusted with SetDrawBrokenLines.

This gives you the basics of how to create a graph in PHPlot. A nice start, but now we'd like to add some customization, namely different fonts, margins and types of graphs.

2.6. Customization

PHPlot can draw these types of plots:

• Bars (with optional shadows) and Stacked Bars, both vertical and horizontal

• Lines

• Linepoints (a faster way of plotting when you want both points and lines)

• Area and Stacked Area

• Points (lots of point types here)

• Pie (2D or 3D)

• Thinbarline (sometimes also called impulse), both vertical and horizontal

• Error bar (which can also be used for stock market data graphs)

• Squared (steps)

You specify which type of plot you want with the SetPlotType function.

There are many ways we can change the look and feel of the graph. Almost every parameter of the graph, including ticks, grids, and data labels, can be adjusted using PHPlot functions. A categorized list of these functions can be found in Chapter 6, *PHPlot Functions By Category*. Each of the functions is described in detail in PHPlot Function Reference.

2.7. What's Next?

If you want to see more pictures and sample code, take a look at the examples in Chapter 5, *PHPlot Examples*.

Otherwise, you can continue with Chapter 3, *PHPlot Concepts* where PHPlot concepts are described in more detail.

Chapter 3. PHPlot Concepts

This chapter explains the operation and use of PHPlot. For advanced topics, see Chapter 4, *PHPlot Advanced Topics*.

3.1. Definitions

This section contains definitions of terms used throughout the PHPlot Reference Manual.

Alpha value
: A component in a color system which represents the amount of transparency, or opacity. At one extreme, an alpha value indicates an opaque object which covers or hides whatever was drawn before it. At the other extreme, it indicates a completely transparent object which has no affect on whatever was drawn before it.

Data Set
: A set of data points which represent some function, trend, samples, etc.

Device Coordinates
: The coordinate space used by GD to create images. The origin is at the upper left corner, X increases to the left, Y increases down, and the units are pixels. Also known as Pixel Coordinates or GD Coordinates.

Element
: A component of a graph, such as a label, tick mark, axis, or plot.

GD
: A programming library used to create and manipulate images. GD can be found at the GD Graphics Library home page [http://libgd.org/]. You can think of GD as a software implementation of a video card. GD is also available as a PHP extension, and that is what PHPlot uses to create images.

Graph
: A complete, labeled, graphical representation of some data sets. In PHPlot, a graph contains a single plot and other elements such as axes, tick marks, and labels.

Horizontal Plot
: A plot in which the Y axis represents the independent variable, and the X axis represents the dependent variable values. In a horizontal bar chart, for example, the bars extend to the right from the Y axis.

Image
: A graphical image, represented as data. For example, a PNG file is an image in PNG format which is stored in a file. PHPlot creates images using the GD library. A PHPlot image contains one or more graphs (but usually only one).

Palette Image
: A color image file, or image in memory, in which each pixel value is an index into a color map which contains the actual color. Typically, the pixel value is 8 bits, limiting this type of image to 256 distinct colors. See also Truecolor Image.

Plot
: The graphical representation of one or more data sets. In PHPlot, each graph contains a single plot, but a plot can contain representation of multiple data sets (for example, you can have 3 line charts on a plot).

Less formally, the term 'plot' is often used for the overall output of PHPlot: an image with a complete graphical representation of data, with labels, legend, title, etc.

Truecolor Image

A color image file, or image in memory, in which each pixel value has a complete representation of the color of that pixel. The pixels may use 24 bits, with 8 bits each for red, blue, and green color components, or 32 bits with an additional alpha component. See also Palette Image.

Vertical Plot

A plot in which the X axis represents the independent variable, and the Y axis represents the dependent variable values. This is the usual orientation for plots. Compare with horizontal plot.

World Coordinates

The coordinate space of the plotted data. This is the real world coordinate space, in the units of whatever the plotted data measures. The origin of the world coordinate space is the 0,0 point on the X and Y axes. The X coordinate increases to the right, and the Y coordinate increases upwards. Note that the Y direction of world coordinates is opposite that of device coordinates.

3.2. Programming Overview

This section contains an overview of how to use PHPlot.

3.2.1. How It Works

To create a plot with PHPlot, your PHP script will generally do the following:

1. Include the phplot.php source using require_once.

2. Create an object which is an instance of the PHPlot class.

3. Use PHPlot functions (methods of the class object) to select the plot type, present the data array, and optionally change settings which control the appearance of the plot.

4. Output the plot, typically to the user's browser but possibly to a file instead.

Note

It is important to remember that if you are writing a PHP script that uses PHPlot to create an image for a web page, that PHP script must output *only* the image data. If you want to embed the image into a web page with text and other images, you need at least two scripts. Your main script returns an HTML page which includes an IMG (Image) tag for the plot. The IMG tag has a SRC attribute which references the second script, and it is this second script which creates the PHPlot image. An example of this can be found in Section 5.23, "Example - Complete Web Form with Plot". You will most likely need a way to communicate parameters from your main script to your image script. Two good ways to do this are using URL parameters, and with PHP session variables.

3.2.2. Annotated Example

Here is a simple, annotated example of a script which produces an image. More examples can be found in Chapter 5, *PHPlot Examples*.

```
require_once 'phplot.php';
```

This brings in the PHPlot source into your script. For this to work, PHP needs to be able to find the PHPlot source file. A good way to arrange this is to install PHPlot into a directory outside your web server's document root and on the PHP Include Path. Other ways are to include a full path to phplot.php when including it, or to copy phplot.php into the same directory as your script.

```
$plot = new PHPlot();
```

Here we create a new PHPlot object and call it `plot`. Everything else we do with the plot will be through the $plot object.

```
$plot->SetPlotType('lines');
$plot->SetDataType('text-data');
```

Here we select the plot type 'lines', for a line plot (see [Section 3.4, "PHPlot Plot Types"](#)), and indicate our data will be represented in the 'text-data' format (see [Section 3.3, "PHPlot Data Types"](#)).

```
$plot->SetDataValues($data);
```

The data array $data is where we store the values to be plotted. We haven't shown where the data came from, but in a typical application it might be from a database query. How the data array is constructed is described in [Section 3.3, "PHPlot Data Types"](#).

```
$plot->SetXDataLabelPos('none');
$plot->SetLineWidths(3);
$plot->SetDrawXGrid(True);
```

These three functions illustrate how to change the appearance of the plot.

```
$plot->DrawGraph();
```

This final function call outputs the plot. More accurately, this function creates the plot using all the data and settings which were established by previous functions, and then outputs the plot. This is a crucial point when using PHPlot: Until you call DrawGraph, PHPlot is simply recording all the settings resulting from the functions you call, and saving a copy of your data array. Nothing really happens until you complete the plot with DrawGraph.

3.3. PHPlot Data Types

This section describes how data need to be organized for use with PHPlot.

3.3.1. Available Data Types

The data values to be plotted are presented to PHPlot with [SetDataValues](#). In all cases, the data values are stored in a PHP array. This data array contains elements, themselves also arrays, which are called records. Each record contains labels and/or data values. The 'data type' of the data array determines how PHPlot will interpret the records in the data array. To set the data type, use [SetDataType](#).

The following data types are available in PHPlot:

text-data
> Each record contains a label, followed by one or more Y values: `array('label', y1, y2, ...)`. The corresponding X value for all Y values in a record is implicit. PHPlot assigns x=0.5 to the first data record, x=1.5 to the second, etc. This data type works with all plot types.

data-data
> Each record contains a label, an X value, then one or more Y values: `array('label', x, y1, y2, ...)`. This is the same as 'text-data' except the X values are explicitly given. This data type works with all plot types except bars and stackedbars.

Note that with data type 'data-data', it is possible to create a data array with duplicate X values, or X values out of order. Depending on the plot type, this may or may not make sense. For example, with a point plot (which puts a marker at each data point), the data array can legitimately contain duplicate and out-of-order X values. With a line plot (which connects adjacent points in the data array with a line), it probably makes no sense to have out-of-order or duplicate X values in the data array.

data-data-error

Each record contains a label, an X value, then sets of 3 values for each Y point: the Y value, error in the positive direction, and error in the negative direction: `array('label', x, y1, e1+, e1-, y2, e2+, e2-, ...)`. This data type works with plot types lines, points, and linepoints only.

Note that both errors (e+ and e-) are given as positive numbers. They represent the absolute value of the error in the positive and negative directions respectively.

text-data-single

Each record contains a label and a single value: `array('label', factor)`. This data type is only for the pie chart plot type.

text-data-yx

Each record contains a label, followed by one or more X values: `array('label', x1, x2, ...)`. The corresponding Y value for all X values in a record is implicit. PHPlot assigns y=0.5 to the first data record, y=1.5 to the second, etc. This data type is for horizontal plots, and works with bar, stackedbar, and thinbarline plot types.

data-data-yx

Each record contains a label, a Y value, then one or more X values: `array('label', y, x1, x2, ...)`. This is the same as 'text-data-yx' except the X values are explicitly given. This data type is for horizontal plots, and only works with the thinbarline plot type.

3.3.2. Building Data Arrays

In most of the examples in this manual, the data array is built from constant values in PHP code. For example:

```
$data = array(
  array('',  0,   0,   0,   0),
  array('',  1,   1,   1, -10),
  array('',  2,   8,   4, -20),
  array('',  3,  27,   9, -30),
  array('',  4,  64,  16, -40),
  array('',  5, 125,  25, -50),
);
```

Which contains 6 records, each with an empty label, an X value (assuming the data type is 'data-data'), and then 3 Y values representing 3 data sets to plot.

In a real application, of course, the data values will most likely come from a calculation, perhaps using values from a database. This section provides a few sample code fragments which construct data arrays. We use the PHP ability to append a new value to the end of an array using `$array[] = ...`.

This code fragment creates a data array of type 'text-data' with three data sets for Y=X+1, Y=X*X/2, and Y=X*X*X/3.

```
$data = array();
for ($x = 0; $x <= 5; $x++) $data[] = array('', $x+1, $x*$x/2, $x*$x*$x/3);
```

This code fragment creates a data array of type 'data-data' with about 100 points from the equation X * Y = 10.

```
$data = array();
```

```
for ($x = 1.0; $x <= 10.0; $x += 0.1) $data[] = array('', $x, 10.0/$x);
```

The next code fragments use database queries to build data arrays for PHPlot. In many cases, you can create a query such that the returned columns correspond to the format of a PHPlot data array record. The first query result column should be the data label, the second (for data type 'data-data' only) should be the X value, and subsequent column results should be one or more Y values (depending on the number of datasets you are plotting). (Pie charts work differently - see Section 3.4.8, "Plot Type: pie (Pie Chart)".) You aren't limited to simple table lookups - you can use the full power of the SQL language to combine tables and perform calculations on the data. Be sure to use ORDER BY in your SQL query to order the results, or you will not get predictable plots.

Database access methods differ. This code is for PostgreSQL; for MySQL there are similar functions like mysql_fetch_row().

```
$r = pg_query($db, 'SELECT ...');
if (!$r) exit();
$data = array();
$n_rows = pg_num_rows($r);
for ($i = 0; $i < $n_rows; $i++) $data[] = pg_fetch_row($r, $i);
...
$plot->SetDataValues($data);
```

This works because pg_fetch_row assigns the result columns from the query to sequentially numbered elements in the array.

Using data arrays from database query results also works if the result columns are in an array which is indexed by the field name, because PHPlot converts the data array to use numeric indexes. So with PostgreSQL you can use pg_fetch_assoc(). You can also use pg_fetch_array(), but only if you specify the type as PGSQL_ASSOC or PGSQL_NUM. The default type PGSQL_BOTH will not work, because the result array will contain the data values duplicated under both number and field-name indexes, and PHPlot will see both copies of the data.

Going even further, with a properly designed query you can use pg_fetch_all() to fetch the entire query result and assign it to a data array with one statement.

```
$r = pg_query($db, 'SELECT ...');
if (!$r) exit();
$data = pg_fetch_all($r);
...
$plot->SetDataValues($data);
```

This uses field-name indexes in the array representing each row, but as noted above PHPlot will convert the data array to use numeric indexes.

3.3.3. Missing Data in Data Arrays

The lines and squared plot types support the concept of missing points. A missing point is represented in your data array with an empty string instead of a Y value. For example:

```
$data = array( array('1996', 45.5),
               array('1997', 53.8),
               array('1998', ''),   # No data available for 1998
               array('1999', 34.1));
```

By default, PHPlot will act as if the missing point does not exist, connecting the points before it and after it. You can use SetDrawBrokenLines to leave a gap at the missing point instead.

3.3.4. Data Array Indexes

There are some rules you need to follow when building data arrays, in order for PHPlot to correctly process your data. The following rules apply to the array indexes, or keys, in your data array.

• Your data array must be indexed using sequential integers starting with zero. This is automatically true if you build an array with the empty-brackets syntax (`$mydata[] = ...`), or if you use the `array(...)` construct without specifying keys. Note that this refers only to the data array itself, not the elements of the data array - the records.

• The data records, which are elements of the data array, are also arrays. These record arrays are processed by PHPlot using the `array_values()` function. This means the array keys are ignored, and the elements of the record are processed in the same order as they were assigned. As with the data array itself, you can use the empty-brackets syntax, or the array() language construct, to build records in the data array. You can also use words (such as database query result fields) as indexes, as long as the assignments are made in the correct order.

3.4. PHPlot Plot Types

This section describes the PHPlot plot types and their individual data type requirements.

Plot types determine the overall look of the graphical representation of the data values. To select the plot type, use SetPlotType. The following plot types are available:

Plot Type	Description
area	Filled areas between lines. Also known as *cumulative line plot* or *component line plot*.
bars	Filled bars with optional 3-D look. Multiple datasets are offset.
candlesticks	An Open/High/Low/Close (OHLC) financial plot using filled and unfilled candlesticks.
candlesticks2	An Open/High/Low/Close (OHLC) financial plot using filled candlesticks.
linepoints	Lines between points, with a marker at each point, and optional error bars.
lines	Straight lines between data points, with optional error bars.
ohlc	A basic Open/High/Low/Close (OHLC) financial plot using lines and ticks.
pie	Pie chart with or without 3-D affects.
points	Draws a marker at each data point, with optional error bars.
squared	Stepped lines
stackedarea	Filled areas between lines, with multiple data sets accumulated.
stackedbars	Filled bars with optional 3-D look. Multiple data sets are accumulated and the sum is graphed.
thinbarline	Vertical lines from the X axis to the value, or horizontal lines from the Y axis to the value. Also known as *impulse*.

3.4.1. Plot Type: area (Area Plot)

This plot type draws filled areas between lines. This is often called a *cumulative line plot* or *component line plot*. Each data set (set of corresponding Y values from each record in the data array) is plotted in order, with the area between each line and the next line filled solid. The area between the last line and the X axis is also filled. The data must be arranged so the values are (generally) decreasing within each row, because later drawn filled areas will cover previously drawn areas.

This plot type works with data types text-data and data-data. For 'text-data', the data X values are assumed to be at 0.5+N for N=0,1,2...

This plot type uses the absolute value of each supplied Y, because negative values do not make sense here. Missing values are taken as zero. All records in the data array must have the same number of Y values.

The areas are filled with colors as set with SetDataColors.

An example of this plot type can be seen in Section 5.3, "Example - Area Plot".

3.4.2. Plot Type: bars (Bar Graph)

This plot type draws a bar chart, with filled rectangles. Both vertical and horizontal bar charts are available. The bars are centered on the X values (for vertical charts), or on the Y values (for horizontal charts). The rectangles can have a 3-D look, or be flat with a border. Multiple data-set plots work, with each one producing a set of bars offset from the previous set.

For vertical bars, use data type text-data. The data X values are assumed to be at 0.5+N for N=0,1,2... For horizontal bars, use data type text-data-yx. The data Y values are assumed to be at 0.5+N for N=0,1,2... No other data type works with bar graphs.

If shading is on with SetShading (default is on with value 5 pixels), then the bars have a 3-D look. If shading is off (SetShading(0)), the bars are flat rectangles with borders. The filled rectangle color is set with SetDataColors. The border color (if shading is off) can be set with SetDataBorderColors.

Examples of this plot type can be seen in Section 5.4, "Example - Bar Chart", Section 5.5, "Example - Unshaded Bar Chart", Section 5.6, "Example - Bar Chart, Label Options", Section 5.19, "Example - Bar Chart with Data Value Labels", and Section 5.27, "Example - Horizontal Bar Chart".

Horizontal bar plots were added in PHPlot-5.1.2.

3.4.3. Plot Type: candlesticks (OHLC Candlesticks Plot)

This plot type represents the changing price of a financial instrument (such as a stock or other security) over time. At each time point, 4 values are plotted: the opening price, the highest price, the lowest price, and the closing price. The candlesticks plot type is one of three Open/High/Low/Close (OHLC) plot types available. It shows the opening and closing prices as the top and bottom of a narrow rectangle (the candlestick), with an upper wick showing the highest price, and a lower wick showing the lowest price. The candlestick body is drawn solid filled if the closing price is lower than the opening price, and as an outline (unfilled) if the closing price is higher than the opening price. (For a variation on this, see candlesticks2.)

This plot type works with data types text-data and data-data. For 'text-data', the data X values are assumed to be at 0.5+N for N=0,1,2...

This plot type requires exactly 4 Y values for each X. These represent the Open, High, Low, and Close prices in that order. Multiple data sets are not supported. Missing Y values are not allowed.

This plot type uses 4 colors from the data colors array. If the security closed lower than it opened for that time period, then color 0 is used for the candlestick body, and color 2 is used for the upper and lower wicks. If the security closed higher or the same as it opened for that time period, then color 1 is used for the candlestick body, and color 3 is used for the upper and lower wicks.

SetLineWidths can be used to set line widths. Index 0 will be used to draw the candlestick bodies, and index 1 will be used to draw the wicks.

An example of this plot type can be seen in Section 5.31, "Example - Candlesticks OHLC (Open, High, Low, Close) Financial Plot".

This plot type was added in PHPlot-5.3.0.

3.4.4. Plot Type: candlesticks2 (OHLC Filled Candlesticks Plot)

This plot type represents the changing price of a financial instrument (such as a stock or other security) over time. The candlesticks2 plot type is the same as the candlesticks plot type, except the candlestick bodies are always drawn filled, regardless of whether the security closes up or down. Color usage and line width usage are the same.

Be sure to set the data colors with SetDataColors for this plot type. Unlike candlesticks, candlesticks2 plots use only color to show the difference between a security closing up or closing down.

This plot type works with data types text-data and data-data. For 'text-data', the data X values are assumed to be at 0.5+N for N=0,1,2... As with candlesticks, this plot type requires exactly 4 Y values for each X: Open, High, Low, and Close prices.

An example of this plot type can be seen in Section 5.32, "Example - Filled Candlesticks OHLC (Open, High, Low, Close) Financial Plot".

This plot type was added in PHPlot-5.3.0.

3.4.5. Plot Type: linepoints (Lines and Points)

This plot type draws a line graph with a marker at each point, thus combining the 'line' and 'point' plot types.

This plot type works with data types text-data and data-data. For 'text-data', the data X values are assumed to be at 0.5+N for N=0,1,2... It also works with data-data-error for error bars.

Line and marker colors for each line are set with SetDataColors. Marker styles for each line are set with SetPointShapes. Marker sizes for each line are set with SetPointSizes. Line widths for each line are set with SetLineWidths. Line styles (solid or dashed) for each line are set with SetLineStyles.

You can also suppress the line, or the markers, for individual data sets in a graph. This allows you combine points-only, lines-only, and line/points plots. Refer to SetLineStyles and SetPointShapes for details.

For error-bar plots with data type 'data-data-error' only: Error bar colors for each line are set with SetErrorBarColors. Error bar shape (tee or line) is set with SetErrorBarShape. If tee-shaped error bars are used, the width of the upper and lower 'tee' is set with SetErrorBarSize. Error bar line width is set with SetErrorBarLineWidth.

An example of this plot type can be seen in Section 5.7, "Example - Line/Point Plot, Point Shapes".

3.4.6. Plot Type: lines (Lines Graph)

This plot type simply draws a line from each point to the next.

This plot type works with data types text-data and data-data. For 'text-data', the data X values are assumed to be at 0.5+N for N=0,1,2... It also works with data-data-error for error bars.

Line colors for each line are set with SetDataColors. Line widths for each line are set with SetLineWidths. Line styles (solid or dashed) for each line are set with SetLineStyles.

For error-bar plots with data type 'data-data-error' only: Error bar colors for each line are set with SetErrorBarColors. Error bar shape (tee or line) is set with SetErrorBarShape. If tee-shaped error bars are used, the width of the upper and lower 'tee' is set with SetErrorBarSize. Error bar line width is set with SetErrorBarLineWidth.

Examples of this plot type can be seen in Section 5.1, "Example - Line Plot" and Section 5.2, "Example - Line Plot: Functions".

3.4.7. Plot Type: ohlc (Basic OHLC Plot)

This plot type represents the changing price of a financial instrument (such as a stock or other security) over time. At each time point, 4 values are plotted: the opening price, the highest price, the lowest price, and the closing price. The ohlc plot type is one of three Open/High/Low/Close (OHLC) plot types available. It shows a vertical line connecting the low and high prices, with small horizontal tick marks showing the opening and closing prices. The opening price tick mark is on the left of the vertical line, and the closing price tick mark is on the right.

This plot type works with data types text-data and data-data. For 'text-data', the data X values are assumed to be at 0.5+N for N=0,1,2...

This plot type requires exactly 4 Y values for each X. These represent the Open, High, Low, and Close prices in that order. Multiple data sets are not supported. Missing Y values are not allowed.

This plot type uses 4 colors from the data colors array. If the security closed lower than it opened for that time period, then color 0 is used for the vertical line, and color 2 is used for the open and close tick marks. If the security closed higher or the same as it opened for that time period, then color 1 is used for the vertical line, and color 3 is used for the open and close tick marks.

SetLineWidths can be used to set line widths. Index 0 will be used to draw the vertical lines, and index 1 will be used to draw the tick marks.

An example of this plot type can be seen in Section 5.30, "Example - Basic OHLC (Open, High, Low, Close) Financial Plot".

This plot type was added in PHPlot-5.3.0.

3.4.8. Plot Type: pie (Pie Chart)

This plot type draws pie charts. The pie chart can have a 3-D look or be drawn flat. The first pie segment starts at 0 degrees (East, or 3:00PM) and they go around the pie in a counter-clockwise direction. Each segment is labeled with its percentage.

This plot type works with data types text-data, data-data, and text-data-single. Data arrays for pie charts are handled differently from with other plot types, so the data types are described in more detail below.

If shading is on with SetShading (default is on with value 5 pixels), then the pie chart has a 3-D look. If shading is off (SetShading(0)), the pie chart is drawn flat (circular rather than oval). The position of the segment percentage labels is set with SetLabelScalePosition.

Examples of this plot type can be seen in Section 5.8, "Example - Pie Chart, text-data-single" (text-data-single), Section 5.9, "Example - Pie Chart, text-data" (text-data), and Section 5.10, "Example - Pie Chart, flat with options" (unshaded).

3.4.8.1. Pie Chart with data type: 'text-data-single'

The data array for pie charts with 'text-data-single' data type is structured as follows. Each record in the data array represents a pie segment. The record is an array of 2 elements: label and value, but the label is ignored. The value sets the relative size of a pie segment. PHPlot totals up the values and computes the relative size of each segment.

For example:

```
$data = array(array('', 1), array('',4), array('',5));
```

This makes a pie chart with 3 segments, with sizes 10%, 40%, and 50%.

3.4.8.2. Pie Chart with data type: 'text-data'

The data array for pie charts with 'text-data' data type is structured as follows. Each record in the data array is an array of a label followed by N data values. The label is ignored. The pie chart will be produced with N segments. The relative weight of the first segment is the sum of the first data values in each record. The relative weight of each subsequent segment is the sum of the corresponding data values in each record.

For example:

```
$data = array(array('', 10, 10, 20, 10),
              array('', 15, 10, 15, 10));
```

This results in 4 segments with sizes 25%, 20%, 35%, and 20%.

3.4.8.3. Pie Chart with data type: 'data-data'

The data array for pie charts with 'data-data' data type is structured the same as 'text-data', except that the first two values in each record are ignored (the positions usually used for label and X value). Each element in the data array represents a record. Each record is an array of a label, X value, then N data values. The label and X value are ignored. The pie chart will be produced with N segments. The relative weight of the first segment is the sum of the first data values in each record. The relative weight of each subsequent segment is the sum of the corresponding data values in each record.

For example:

```
$data = array(array('', 1, 10, 10, 20, 10),
              array('', 2, 15, 10, 15, 10));
```

This results in 4 segments with sizes 25%, 20%, 35%, and 20%. The empty strings and '1' and '2' are ignored.

3.4.9. Plot Type: points (Styled Dot Plot)

This plot type draws a point marker at each X,Y value.

This plot type works with data types text-data and data-data. For 'text-data', the data X values are assumed to be at 0.5+N for N=0,1,2... It also works with data-data-error for error bars.

Marker colors for each line are set with SetDataColors. Marker styles for each line are set with SetPointShapes. Marker sizes for each line are set with SetPointSizes.

For error-bar plots with data type 'data-data-error' only: Error bar colors for each line are set with SetErrorBarColors. Error bar shape (tee or line) is set with SetErrorBarShape. If tee-shaped error bars are used, the width of the upper and lower 'tee' is set with SetErrorBarSize. Error bar line width is set with SetErrorBarLineWidth.

Examples of this plot type can be seen in Section 5.12, "Example - Points Plot / Scatterplot", and Section 5.11, "Example - Points Plot with Error Bars".

3.4.10. Plot Type: squared (Step Plot)

This plot type makes stepped lines. For each point, you get a horizontal line from the previous point to the current X, then a vertical line to the current Y.

This plot type works with data types text-data and data-data. For 'text-data', the data X values are assumed to be at 0.5+N for N=0,1,2...

Line colors per line are set with SetDataColors. Line widths per line are set with SetLineWidths. Line style (solid or dashed) per line are set with SetLineStyles.

An example of this plot type can be seen in Section 5.13, "Example - Squared Plot".

3.4.11. Plot Type: stackedarea (Stacked Area Plot)

This plot type draws filled areas between lines, similar to area except the values are accumulated as in stackedbars plots. The area between the X axis and the first data set (the line resulting from the first Y value from each record) is filled first. Then the area above that line, up to the sum of the first and second Y values in each record, is filled next. This repeats until filling the area up to the top-most line, which is the sum of all the Y values from each record.

This plot type works with data types text-data and data-data. For 'text-data', the data X values are assumed to be at 0.5+N for N=0,1,2...

This plot type uses the absolute value of each supplied Y, because negative values do not make sense here. Missing values are taken as zero. All records in the data array must have the same number of Y values.

The areas are filled with colors as set with SetDataColors. Note that data sets are processed with stacked area plots in reverse order relative to area plots. With stacked area plots, the first data set (using the first data color) will be at the bottom, but with area plots it will be at the top. Compare Section 5.3, "Example - Area Plot" and Section 5.21, "Example - Stacked Area Plot".

An example of this plot type can be seen in Section 5.21, "Example - Stacked Area Plot".

This plot type was added in PHPlot-5.1.1.

3.4.12. Plot Type: stackedbars

This plot type draws a bar chart with stacked bars. Both vertical and horizontal stacked bar charts are available. The bars are centered on the X values (for vertical charts), or on the Y values (for horizontal charts). Each data set value contributes one segment of a stack. That is, the first data set is drawn from the axis in the first color, then the second data set is drawn stacked on the first using the second color, etc.

Data values greater than zero result in an upward (or rightward) bar. Data values less than zero result in a downward (or leftward) bar. Mixing negative and positive values within a row does work, but the results are not generally useful. The first non-zero value in any given row determines the direction of that bar stack. PHPlot keeps a running total for each row, but does not draw any segment unless it increases the bar stack height (or length, for horizontal plots). It also does not draw any segment on the wrong side of the axis (which is normally at 0).

For vertical bars, use data type text-data. The data X values are assumed to be at 0.5+N for N=0,1,2... For horizontal bars, use data type text-data-yx. The data Y values are assumed to be at 0.5+N for N=0,1,2... No other data type works with stacked bar plots.

If shading is on with SetShading (default is on with value 5 pixels), then the bars have a 3-D look. If shading is off (SetShading(0)), the bars are flat rectangles with borders. The filled rectangle colors for each stacked segment are set with SetDataColors. The border color (if shading is off) can be set with SetDataBorderColors.

Examples of this plot type can be seen in Section 5.14, "Example - Stacked Bars, Shaded", Section 5.15, "Example - Stacked Bars, Unshaded", Section 5.20, "Example - Stacked Bars with Y Data Value Labels", and Section 5.28, "Example - Horizontal Stacked Bar Chart".

Horizontal stacked bar plots were added in PHPlot-5.1.3. Support for negative values in stacked bar plots was added in PHPlot-5.2.0.

3.4.13. Plot Type: thinbarline

This plot type draws lines from the axis to the data value. Other implementations call this type of plot *impulses*. Both vertical and horizontal thinbarline plots are available. Plotting multiple data sets does not work, because the lines are drawn on top of each other and only one can typically be seen. This plot type can be more readable than a bar chart when there are a large number of data points.

For vertical plots, use data type text-data or data-data. For 'text-data', the data X values are assumed to be at 0.5+N for N=0,1,2... For horizontal plots, use data type text-data-yx or data-data-yx. For 'text-data-yx', the data Y values are assumed to be at 0.5+N for N=0,1,2...

The width of the plot lines can be controlled with SetLineWidths.

Examples of this plot type can be seen in Section 5.16, "Example - Thin Bar Line Plot", Section 5.17, "Example - Thin Bar Line Plot, Wider Lines", and Section 5.29, "Example - Horizontal Thin Bar Line Plot".

3.4.14. Plot Types and Data Types

The following table indicates which plot types support which data types. Refer to Section 3.3, "PHPlot Data Types" for more information on data types.

Plot Type:	Data Type:					
	text-data	**data-data**	**data-data-error**	**text-data-single**	**text-data-yx**	**data-data-yx**
area	Yes	Yes				
bars	Yes				Yes	
candlesticks	Yes	Yes				
candlesticks2	Yes	Yes				
linepoints	Yes	Yes	Yes			
lines	Yes	Yes	Yes			
pie	Yes	Yes		Yes		
ohlc	Yes	Yes				
points	Yes	Yes	Yes			
squared	Yes	Yes				
stackedarea	Yes	Yes				
stackedbars	Yes				Yes	
thinbarline	Yes	Yes			Yes	Yes

3.5. Colors

This section contains information about using colors in PHPlot. Functions described in Section 6.3, "Colors and Line Styles" in the Reference chapter control the use of colors in PHPlot.

This section describes Palette images. Starting with PHPlot-5.1.1, a second color model is available in PHPlot: Truecolor images. Refer to Section 4.2, "Truecolor Images" for more information.

3.5.1. Color Parameter Forms

Individual colors as arguments to PHPlot functions can take one of the following forms:

1. A color name, as defined by SetRGBArray or from a built-in color map if SetRGBArray was not called. Note that color names are case sensitive.

2. Numeric color component values, in the form #rrggbb. Here rr is red, gg is green, and bb is blue, and each component value is represented as a 2-digit hexadecimal number between 00 and ff. For example, #0000ff is full-saturation blue.

3. A PHP array of red, green, and blue color component values, each value being in the range 0 to 255 inclusive, for example array(0,0,255) for blue.

Additional color forms can be found in Section 4.2.4, "Color Parameter Form Extensions". Those forms are more useful with truecolor images.

Note

You cannot use the (red, green, blue) array form as a color value in those functions (like SetDataColors) which accept either a single color or an array of colors. The functions are unable to distinguish between an array of colors and a single color represented as an array. However, you can work around this restriction by using an array containing the array with the colors, for example: array(array(102, 0, 192)).

3.5.2. Built-in Color Maps

There are 36 colors defined in the 'small' internal color map. This is the set of colors available by default, unless SetRGBArray is used to load in a different color map. The colors and their names are shown in the figure below.

PHPlot Default Colors

Here are the color names again.

DarkGreen	DimGrey	PeachPuff	SkyBlue	SlateBlue	YellowGreen
aquamarine1	azure1	beige	black	blue	brown
cyan	gold	gray	green	grey	ivory
lavender	magenta	maroon	navy	orange	orchid
peru	pink	plum	purple	red	salmon
snow	tan	violet	wheat	white	yellow

The color names and values in the 'small' internal color map are selected from the X11 RGB Color Database. If you use SetRGBArray to pick the 'large' color map, PHPlot loads a much larger list of colors equivalent to the entire X11 RGB Color Database. Note that there are some duplicate colors in the maps, as they include alternate spellings (like 'gray' and 'grey').

3.5.3. Plotting Colors

Each data set plotted on a graph uses the next color in the Data Colors list. By default, the Data Colors list contains the following 16 colors in order.

Data Set:	Color Name:	Color Sample:
1	SkyBlue	
2	green	
3	orange	
4	blue	
5	red	
6	DarkGreen	
7	purple	
8	peru	
9	cyan	
10	salmon	
11	SlateBlue	
12	YellowGreen	
13	magenta	
14	aquamarine1	
15	gold	
16	violet	

An additional color list is used with error plots. These have data type data-data-error (see Section 3.3.1, "Available Data Types"). The positive and negative error bars use a color map that is set using SetErrorBarColors. By default, this color list contains the same colors as the data color list, so each data set and its error bars will be in the same color. If you change the data colors list with SetDataColors, you probably want to use the same color list for the error bars.

For bar charts and stackedbar charts, if shading is off, the color of the bar borders can be set with SetDataBorderColors. By default, unshaded bar charts use a black outline around the bars for all data sets.

Note: PHPlot through version 5.0.7 used 8 colors in the default Data Colors list: SkyBlue, green, orange, blue, orange, red, violet, and azure1. If plotting more than four data sets with PHPlot-5.0.7 or earlier, you should use SetDataColors to define your own data colors list. Otherwise you will get two data sets plotted in the same color, orange.

Instead of using sequential data colors for plotted data sets, you can control exactly which data color is used for each data value using the data color callback. For more information, see Section 4.4, "Custom Data Color Selection".

3.5.4. Transparency

You can designate one color in the color map to be transparent. This is most often used to make a plot with a transparent background. Use SetTransparentColor to designate the color, and SetBackgroundColor to use that color for the background. Use a color which is not otherwise used in the plot.

For transparency to work, the output format (see SetFileFormat) must support transparency, and the the user's viewer or browser also must support transparency. If transparency is not supported, the user will see the actual color which was designated as transparent (so don't use red, for example). Most viewers support transparency in GIF format, and newer viewers should support transparency in PNG format. JPEG format does not support transparency.

3.5.5. Plot Element Colors

The following table shows the function used to set the color of each element on a plot.

Plot element:	Function used to set color:
Bar chart bar outlines	SetDataBorderColors
Data elements (points, lines, bar fill, etc.)	SetDataColors
Data value labels	SetTextColor
Error bars	SetErrorBarColors
Image background	SetBackgroundColor
Image border	SetImageBorderColor
Legend background	SetBackgroundColor
Legend color box fill	SetDataColors
Legend color box outline	SetTextColor
Legend outline	SetGridColor
Legend text	SetTextColor
Main title	SetTitleColor
Pie chart data labels	SetGridColor
Pie slice fill	SetDataColors
Pie slice outlines	SetGridColor
Plot area background	SetPlotBgColor
Plot border	SetGridColor
X axis line	SetGridColor
X axis data labels	SetTextColor
X data lines	SetLightGridColor
X grid lines	SetLightGridColor

Plot element:	Function used to set color:
X tick labels	SetTextColor
X tick marks	SetTickColor
X title	SetXTitleColor, SetTitleColor
Y axis lines	SetGridColor
Y axis data labels	SetTextColor
Y grid lines	SetLightGridColor
Y tick labels	SetTextColor
Y tick marks	SetTickColor
Y title	SetYTitleColor, SetTitleColor

3.6. Labels

This section contains information about creating labels which identify data values on the plot. For a list of functions used to control labels, see Section 6.11, "Labels".

Several types of labels are available in PHPlot:

- Tick labels identify the values at the tick positions. There are X tick labels and Y tick labels.

- Axis data labels display the label values from your data array. X axis data labels are available for vertical plots, and Y axis data labels are available for horizontal plots.

- Data value labels display the actual value of a data point. These are available only for some plot types. Y data value labels are available for vertical plots, and X data value labels are available for horizontal plots.

- Pie chart data labels show the percentage of each pie slice.

Note

The term *data label* is often used to refer to both axis data labels and data value labels. The same PHPlot functions are used to configure axis data labels and data value labels.

Here is a sample plot with the Y tick labels and X axis data labels called out.
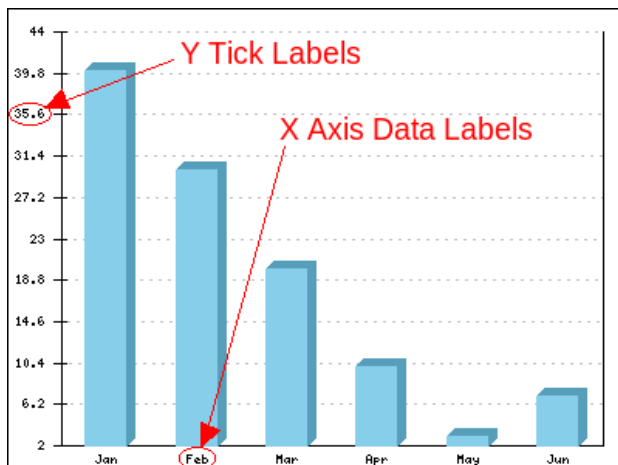
Here is a sample plot with the pie chart labels called out.
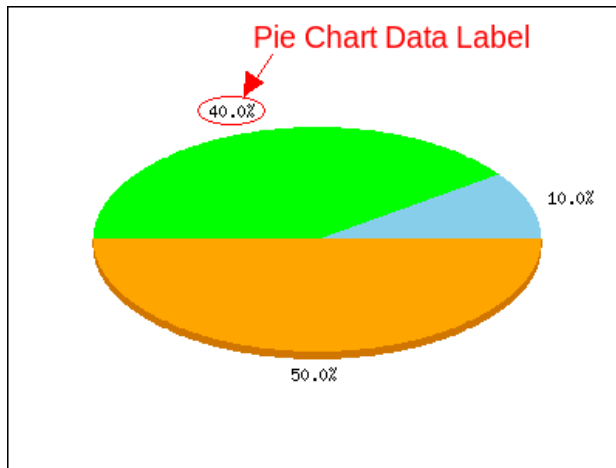
Here is a sample vertical bar plot with the Y tick labels, Y data value labels, and X axis data labels called out.
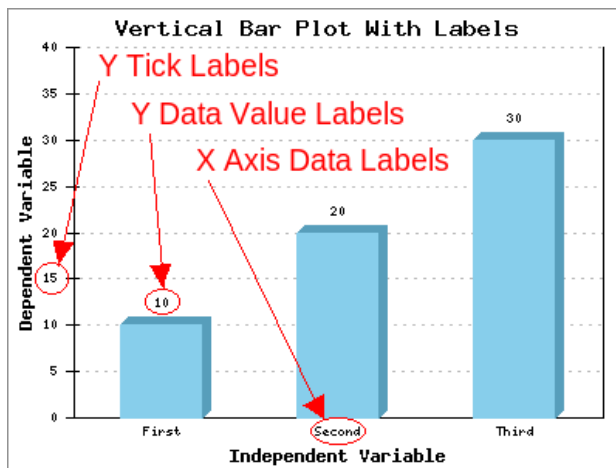
Here is a sample horizontal bar plot with the Y (axis) data labels, X data value labels, and X tick labels called out.
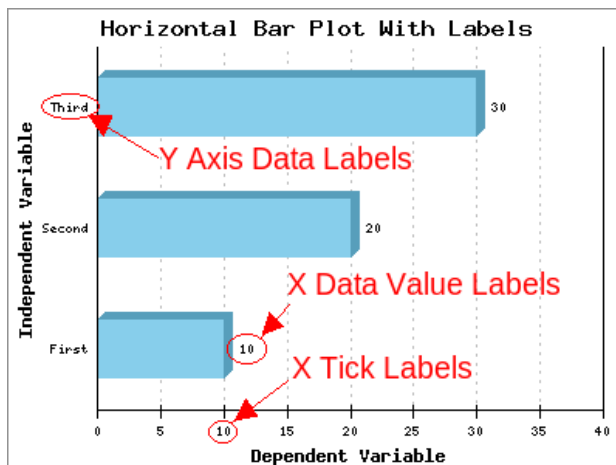
3.6.1. Tick Labels

Tick labels are calculated from the X or Y values of the data. By default, PHPlot will figure out what to use for X and Y tick labels, but the results may not be what you want. You can change the calculated tick labels by using several PHPlot functions. You can use SetXTickIncrement and SetYTickIncrement to set the spacing between tick marks (in World Coordinates), or you can use SetNumXTicks and SetNumYTicks to set the number of tick marks. These don't affect the value of the first tick mark, only the interval. To set the value for the first tick mark, you define the World Coordinate mapping with SetPlotAreaWorld. For example:

```
$plot->SetPlotAreaWorld(-10, NULL, 10, NULL);
$plot->SetXTickIncrement(1);
```

This results in the X tick labels going from -10 to 10, with a tick mark every 1 data unit.

Note that even with data type 'data-data', where explicit independent variable values for the data are supplied, the tick labels along that axis are still calculated automatically (unless modified by the functions named above). For example, for vertical plots, your supplied X values in the data array are not used for the X tick labels.

You can enable, disable, or position the tick labels with SetXTickLabelPos and SetYTickLabelPos.

3.6.2. Axis Data Labels

Axis data labels are available for the independent variable axis. This is X for vertical plots, and Y for horizontal plots. These data labels are supplied in your data array for each data point. For example, with data type text-data :

```
$data = array( array('Peaches',100),
                   array('Apples', 140),
                   array('Pears', 90));
```

The three points have data labels 'Peaches', 'Apples', and 'Pears'. For vertical plots, these data labels will be drawn at the bottom of the plot (by default) below the corresponding X values. For horizontal plots, these data labels will be drawn to the left of the plot (by default), to the left of the corresponding Y values. You can disable or reposition the data labels with SetXDataLabelPos and SetYDataLabelPos.

Note

The axis data labels are not necessarily drawn along axis lines. They are usually drawn along the bottom (for X) or left side (for Y) of the plot. Although these are also the usual positions for the X axis and Y axis, the actual axis lines may be drawn elsewhere. See SetXAxisPosition or SetYAxisPosition for more information on axis positions.

You will generally not want both tick labels and axis data labels on, because they will overlap and be unreadable. If you are not using data labels, you should either make them all empty strings in your data array, or else use SetXDataLabelPos('none') (for vertical plots) or SetYDataLabelPos('none') (for horizontal plots) to turn them off. You can also call SetXTickLabelPos (for vertical plots) or SetYTickLabelPos (for horizontal plots) to explicitly position the tick labels. PHPlot will then disable the data labels.

If you don't tell PHPlot what to do with data and tick labels, the behavior depends on the PHPlot version. PHPlot 5.1.0 and later will examine your data array to see if there are any non-empty data labels, and if so it will draw only data labels, and omit tick labels. If all of the labels in your data array are empty, tick labels will be drawn. (PHPlot through 5.0.7 will draw both tick and data labels in these cases.)

3.6.3. Data Value Labels

Data value labels are available only for some plot types. These are displayed inside the plot, and show the value at each data point, bar, or bar segment.

For vertical bar charts, Y data value labels indicate the Y value for each bar, and are drawn above the bar for positive values, or below the bar for negative values. For vertical stackedbar charts, Y data value labels indicate the total Y value for each stack, and optionally indicate the value of each segment. Use SetYDataLabelPos to enable Y data value labels.

For horizontal bar charts, X data value labels indicate the X value for each bar, and are drawn to the left or right of the end of the bar. For horizontal stackedbar charts, X data value labels indicate the total X value for each stack, and optionally indicate the value of each segment. Use SetXDataLabelPos to enable X data value labels.

For vertical plots of types lines, linepoints, points, and squared, Y data value labels indicate the value of each point and are displayed (by default) above each point. PHPlot does not attempt to prevent interference between the labels and other plot elements. Use SetYDataLabelPos to enable Y data value labels. To change the position of the labels, see Section 4.5.4, "Label Tuning". Note: data value labels for these plot types was added in PHPlot-5.3.0.

Data value labels are not available with other plot types.

Note

The same function is used to position X axis data labels and X data value labels, and the same function is used to position Y axis data labels and Y data value labels. There is no ambiguity, because one type of label is available for each axis for vertical plots, and the other type for horizontal plots.

Example 5.19, "Bar Chart with Data Value Labels" shows a vertical bar chart with Y data value labels. Example 5.20, "Stacked Bars with Y Data Value Labels" shows a vertical stacked bar chart with Y data value labels. Example 5.33, "Linepoints Plot with Data Value Labels" shows a linepoints plot with Y data value labels.

Example 5.27, "Horizontal Bar Chart" shows a horizontal bar chart with X data value labels. Example 5.28, "Horizontal Stacked Bar Chart" shows a horizontal stacked bar chart with X data value labels.

3.6.4. Formatting Labels

Both tick and data labels are subject to format controls. There are several choices in formatting. By default, the label value itself is simply displayed. Use SetXLabelType and SetYLabelType to select one of the other format types for tick labels. Use SetXDataLabelType and SetYDataLabelType to select one of the other format types for data labels (both axis data labels and data value labels). (Note that SetXLabelType also sets the default format for X data labels, for use if SetXDataLabelType is not called. Also SetYLabelType sets the default for Y data labels, for use if SetYDataLabelType is not called.)

Label format type 'data' expects the tick or data label values to be numbers, and formats the values as floating point numbers with a separator between every group of thousands and a fixed number of decimal places. You can set the number of digits of precision, with the default being 1 digit. PHPlot will try to set the thousands grouping separator and decimal separator according to your locale, but this can be overridden if necessary.

Label format type 'time' expects the tick or data label values to be a PHP time value (number of seconds since a fixed base data, the Unix Epoch). PHPlot will format the labels according to the format string you provide. Refer to the PHP documentation for `strftime()` for details on the format string, but here are some examples for 31 December 2004 at 1:23:45 pm:

Format String:	Result:
%Y-%m-%d	2004-12-31
%b %Y	Dec 2004
%b %d, %Y	Dec 31, 2004
%d %b	31 Dec
%H:%M:%S	13:23:45

Note

If you select 'time' formatting, but don't set a time format string, PHPlot-5.0rc3 and higher will format the values as hours, minutes, and seconds as shown in the last row of the table above. (The default format was undefined before version 5.0rc3.)

Also note that there are limits to the range of this type of formatting that can make it unusable for historical data. On some platforms, dates before 1970-01-01 can not be formatted.

Starting with PHPlot-5.0.4, empty string values for data labels are ignored for 'time' and 'data' formatting. Earlier versions would format the labels as 0 (for 'data') or cause an error (for 'time').

While date/time formatting can be useful, for X values it may be easier to just format the label values in your PHP code and put the result into the label positions in the data array. If you need date/time formatting for Y values (and it is hard to imagine where that would be useful), you have no option but to use the 'time' format labels for tick values.

Two additional label format types are available. Label format type 'printf' uses a custom print format string. To use label format type 'custom', you supply a function of your own to format the labels. See SetXLabelType for more details about these format types.

3.7. Other Plot Elements

This section contains information about other elements which can be part of a plot.

3.7.1. Titles

PHPlot can draw three types of titles:

• The main plot title, which is centered at the top of the image. This is typically used to identify the plot as a whole.

• The X title, which is drawn horizontally and can appear below the plot, above the plot, or in both places. This is typically used to identify the values along the X axis.

• The Y title, which is drawn vertically and can appear to the left of the plot, to the right of the plot, or on both sides. This is typically used to identify the values along the Y axis.

For a list of functions used to control titles, see Section 6.7, "Titles".

Here is a sample plot with the titles called out.
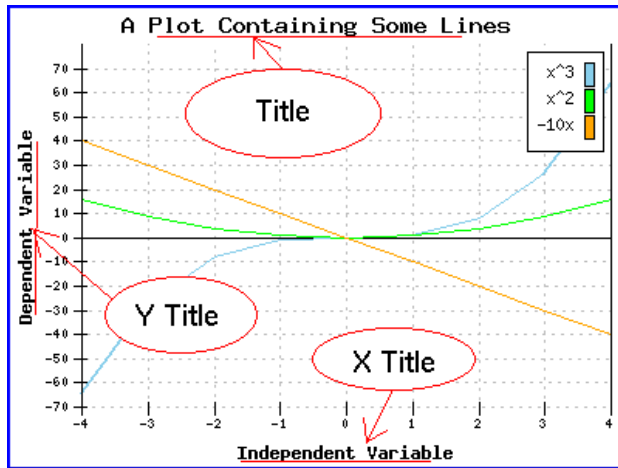
3.7.2. Legend

PHPlot can draw a legend on the plot. This is normally used with multiple data sets, to identify the data sets by color. A legend can also be used with pie charts to identify the sections. For a list of functions used to control the legend, see Section 6.8, "Legend".

Here is a sample plot with the legend called out.
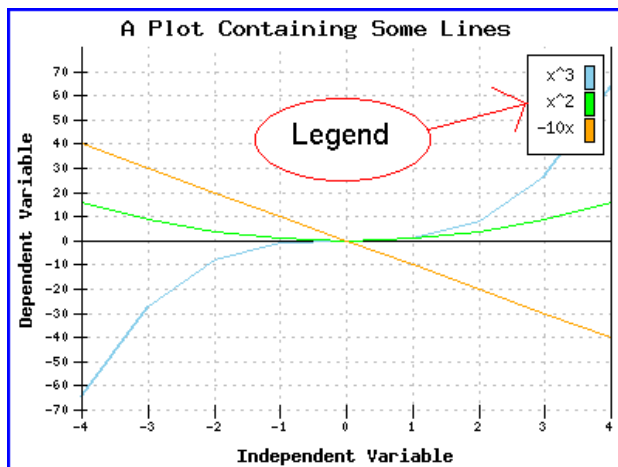
3.7.3. Grid Lines

PHPlot can draw horizontal (Y) and/or vertical (X) grid lines on a plot. You can independently enable the horizontal (Y) and vertical (X) lines in the grid, and use dashed or solid lines. For a list of functions used to control the grid, see Section 6.10, "Grid Controls".

Here is a sample plot with the X Grid and Y Grid called out.
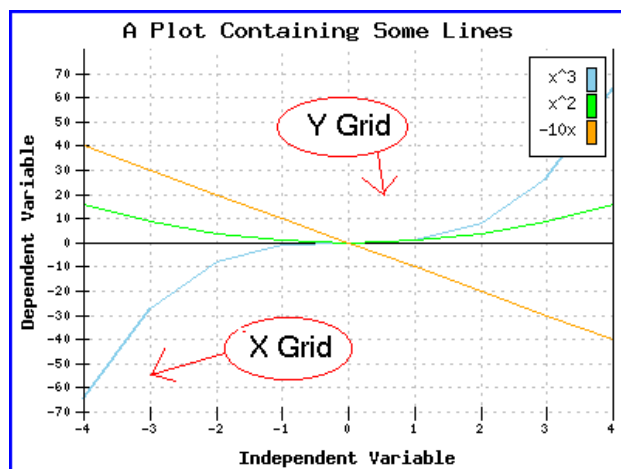
3.7.4. Tick Marks

Tick marks are drawn by default along the bottom edge of the plot (X tick marks) and the left side of the plot (Y tick marks). These usually, but not always, correspond to the X and Y axis lines. You can set the tick interval or control the number of ticks, suppress the first or last tick on an axis, and control the tick mark size. For a list of functions used to control tick marks, see Section 6.12, "Ticks".

Left to its own, PHPlot will determine tick values for X and Y axes, but it currently isn't too smart about choosing values and you may not be happy with the results. You can use PHPlot functions to set the desired tick interval or number of ticks, but you will most likely also have to call SetPlotAreaWorld to set the X and/or Y data range limits. Otherwise, you get the tick intervals you want, but the actual values will depend on a somewhat arbitrary base value.

3.8. Text Fonts

This section contains information about using text fonts in PHPlot. See Section 6.6, "Text Fonts" for PHPlot functions used with text fonts.

3.8.1. Overview

PHPlot supports both built-in GD fonts and TrueType fonts (if available on your system). TrueType fonts generally produce higher quality text, but using them requires more internal computations. The built-in GD fonts are faster to render, but are limited to one typeface and 5 available sizes. TrueType fonts can be drawn at any size and angle, and many typefaces are available. On most systems, TrueType fonts are anti-aliased for improved appearance, but under some conditions the GD fonts may be easier to read. TrueType fonts support much wider character sets, including special symbols, while the GD fonts are more limited.

The following figure shows the built-in GD fonts plus a sample TrueType font. (Depending on how you are viewing this manual, the sizes of the fonts in this figure might differ from how they would look in a PHPlot image. For more on font sizes, see the notes with SetFont.)
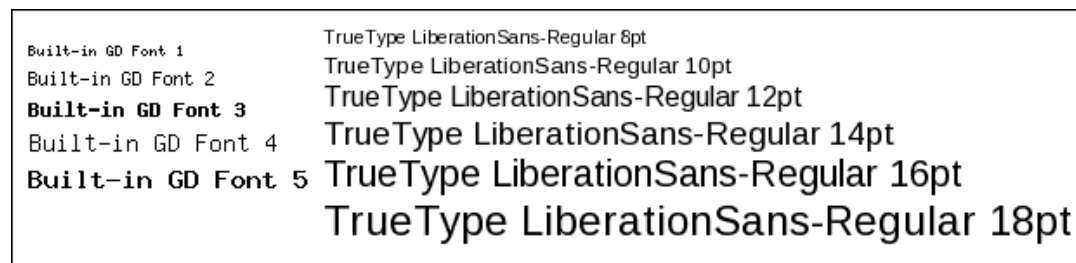
3.8.2. TrueType Font Selection

PHPlot text can use built-in GD fonts or TrueType fonts. When using GD fonts, you specify a font name as a number between 1 and 5. This selects from 5 built-in GD fonts. When using TrueType fonts, you need to specify a font filename. The rest of this section discusses only TrueType fonts.

Note

The material on specifying font files for PHPlot applies to releases starting with PHPlot-5.1.3. Through PHPlot-5.1.2, you generally need to specify the full pathname of a font file, or the full path of a font directory.

On Windows systems, you need to use the font filename, not the font name. You can get the font filename using Control Panel - Fonts. For example, Windows applications may display "Arial Black", or "Arial Black (TrueType)" as a font name, but the actual font filename is "ariblk.ttf". Since GD knows to look for fonts in the Windows font directory, you will not need to specify a full pathname to font files, unless the font is installed in some other directory.

On Windows, you can use the "Character Map" system tool to examine a font. This can also be used to find the Unicode character code of a special character. These will be displayed in hexadecimal, for example U+20AC for the Euro. See Section 3.8.4, "Using Special Characters" for more information on using special characters.

Here are some font selection examples for Windows:

```
# For titles, use Arial Bold Italic at 14 points:
   $plot->SetFontTTF('title', 'ARIALBI.TTF', 14)
   # For X Title, use Verdana at 12 points:
   $plot->SetFontTTF('x_title', 'VERDANA.TTF', 12)
```

On some Linux and similar systems, GD is able to find fonts specified without paths, but on other systems you will have to specify a font directory with either SetTTFPath or as part of the font name in SetFontTTF. If you specify a full pathname to a font, you must also supply the extension (.ttf); you may omit the extension when relying on GD to find the font. Remember that font filenames are case sensitive on most of these systems.

The font search path for GD (bundled with PHP) includes the following directories on Linux and similar systems:

- /usr/X11R6/lib/X11/fonts/TrueType

- /usr/X11R6/lib/X11/fonts/truetype

- /usr/X11R6/lib/X11/fonts/TTF

- /usr/share/fonts/TrueType

- /usr/share/fonts/truetype

- `/usr/openwin/lib/X11/fonts/TrueType`

If your system has TrueType fonts in one of those directories, you can select them with a filename only. If not, you must use a full directory path in either the font name or with SetTTFPath.

Note

> The environment variable `GDFONTPATH` can be defined to contain a list of directories (separated by a colon ':') to search for fonts. If defined, this replaces the above list.

Your Linux system may include a tool for examining fonts. One such tool is gucharmap. This can also be used to find the Unicode character code of a special character. These may be displayed in hexadecimal, for example U+20AC for the Euro. See the next section for more information on using special characters.

Here are some font selection examples for Linux:

```
# On systems with fonts in an expected place, like Slackware Linux,
   # just use the font filename:
   # For titles, use Liberation Sans Bold Italic at 14 points:
   $plot->SetFontTTF('title', 'LiberationSans-BoldItalic.ttf', 14)
   # For X Title, use DejaVuSans Bold at 12 points:
   $plot->SetFontTTF('x_title', 'DejaVuSans-Bold.ttf', 12)

   # Ubuntu and Debian use subdirectories under a searched path.
   # You can use a partial path here.
   $plot->SetFontTTF('x_title', 'ttf-liberation/LiberationSans-Regular.ttf', 12)

   # Fedora uses subdirectories which are not under a searched path.
   # You must use full paths here.
   $plot->SetTTFPath('/usr/share/fonts/liberation/');
   $plot->SetFontTTF('x_title', 'LiberationSans-Regular.ttf', 12)
```

3.8.3. Default TrueType Font

Note

> This section applies starting with PHPlot-5.1.3.

If you try to use TrueType text without specifying a font name, PHPlot will use the default font. You can set the default font with SetDefaultTTFont. If you do not set a default font, PHPlot tries to locate a sans-serif font to use. Here are the font names that PHPlot tries in order. First it tries the filename alone, letting GD use its search path, and then it tries with the default font path, as set with SetTTFPath.

- `LiberationSans-Regular.ttf` - Likely to work on Linux and other systems with a correct GD font search path.

- `Verdana.ttf`, `Arial.ttf`, `Helvetica.ttf` - One of these is going to work on Windows, maybe other systems too.

- `ttf-liberation/LiberationSans-Regular.ttf` - This is for Debian, Ubuntu, and similar.

- `benjamingothic.ttf` - The original PHPlot default, for compatibility.

The last item on the list is used regardless of whether it can be found or not. This means if you enable TrueType fonts without setting a default, and get a fatal error from PHPlot that it can't find the font benjamingothic.ttf, this means PHPlot was unable to find any of the standard fonts in its list. On that system, then, you must provide either a font directory, or use full font pathnames.

3.8.4. Using Special Characters

You can include special characters in your PHPlot labels and titles. This refers to characters which you may not be able to type with a single key on your keyboard, including accented characters and special symbols.

PHPlot itself does not do any special processing of text strings, so you should refer to the PHP GD and Image Functions reference for more information.

Note

This mostly only works with TrueType fonts. The built-in GD fonts do have some extended characters, but they are encoded in ISO8859-2 which is probably not what you might expect, and the GD font routines do not support special character entities.

To use special characters in your PHPlot text strings, you need a TrueType font that contains the characters you want. Ideally, you want a Unicode font. You might have to examine the font using an operating system-specific tool to see if your characters are present and to find their numeric values.

There are two basic ways to include special characters in your text strings. The examples below use the Euro character, which is decimal Unicode value 8364.

• Use HTML-type character entities with decimal numeric encoding. For example, the Unicode Euro symbol is: €

• Include the UTF-8 encoding of the Unicode value in your string as a series of hex escapes. For example, the Euro symbol is: "\xe2\x82\xac".

These are shown in the example below, both of which set the Y axis title to "Items per €100".

```
$plot->SetYTitle("Items per &#8364;100"); # Numeric character entity
  $plot->SetYTitle("Items per \xe2\x82\xac100"); # UTF-8 encoding
```

You can also use PHP functions to encode your characters for including in PHPlot text strings. See the PHP documentation for the functions `html_entity_decode()` and `iconv()`. Here are some examples (sent in by Sourceforge user 'kalvaro'):

```
# Encode the Euro symbol into UTF-8:
$chars = html_entity_decode('&euro;', ENT_NOQUOTES, 'UTF-8');

# Use iconv() to convert a character value xA4 in ISO-8859-15 to UTF:
$chars = iconv('iso-8859-15', 'utf-8', chr(0xA4));
```

3.9. Error Handling

This section describes error handling in PHPlot. This information may not be accurate for PHPlot-5.0.4 and earlier.

3.9.1. Error Handling Overview

Errors detected within PHPlot are programming or installation errors. These are conditions that web application users should never see, because they should be detected and corrected before an application is deployed. Therefore, error handling in PHPlot is aimed more at the developer than the application user.

PHPlot does the following when an error is detected:

- Creates an error image - an image containing the text of the error message.

- Outputs the error image to standard output or to a file, depending on where the plot image was supposed to go.

- Triggers a user-level error condition. If an error handler has been established, it determines what happens next. Otherwise, with no error handler: Writes the error message to error output, or logs it to the web server error log, depending on the PHPlot SAPI in use. Then the script will exit with a non-zero exit status.

It is important not to have any text sent to standard output, even when an error occurs, or the image will be corrupted or PHP will display a "headers already sent" error and no image. Therefore it is necessary to turn off the PHP **display_errors** parameter, otherwise PHP will also write the error messages to standard output. This can be turned off in the php.ini configuration file, where it affects all scripts, or in an application script using:

```
ini_set('display_errors', 'off');
```

Note that an image is produced and output on error even if `SetPrintImage(False)` is used to suppress or delay the normal output of a plot image. The error image is meant for the application developer or tester, but you need to see the error message in order to fix the problem which caused it, so the image is output when the error occurs.

The following figure shows an example of an error image resulting from `$plot->SetPlotType('dots')`:

```
SetPlotType(): 'dots' not in available choices: 'bars, stackedbars, lines,
          linepoints, area, points, pie, thinbarline, squared'.
```
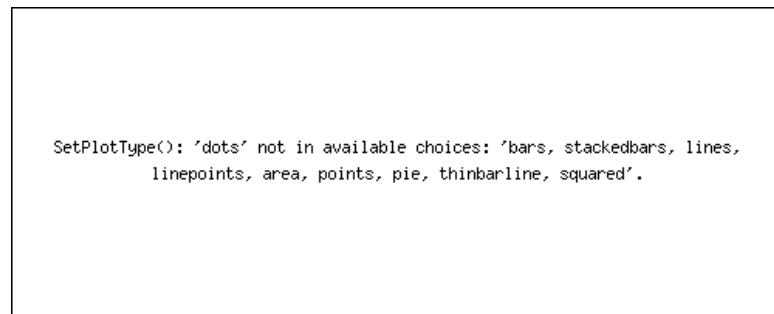
3.9.2. Types of Errors

The following types of errors can occur within PHPlot:

1. Parameter value errors: Use of an incorrect argument to a PHPlot function, such as: SetPlotType('dots') ['dots' is not a valid plot type].

2. Semantic errors: Invalid combination of parameters or data values, such as trying to use data type 'data-data' with plot type 'bars' [bar charts only work with 'text-data' data type].

3. Pathname errors: Missing font file or invalid font path; missing or invalid image file used as background. It might seem extreme to have a missing font file be a fatal error, but PHPlot has no way to substitute an appropriate font, and a missing font would indicate an application configuration or installation error.

4. Inability to create a GD image resource. Probably the only way this can happen is if there is insufficient memory, which can occur if PHP's configured per-script memory limit is reached. (See note below)

All of these result in an E_USER_ERROR level error, except for memory exhaustion when creating an image, which is E_ERROR (fatal unrecoverable). If no GD image resource was created, no error image will be output. Furthermore, if the reason was memory exhaustion, there is no way to catch the error and PHP will cause the script to immediately exit.

3.9.3. Error Handlers

It is possible to set up an error handler with PHP's `set_error_handler` to catch most errors from PHPlot. The handler can be established for all errors (the default), or just E_USER_ERROR error types (the only type PHPlot will trigger). See the PHP documentation for more details. Your handler function can perform cleanup before it exits, however it should not return. Some of the PHPlot functions will correctly handle a return from an error handler, and return FALSE to their callers, but not all. At the very least, a PHPlot object instance should be unset and not re-used after error. Use of error handlers that return is untested and unsupported.

Note that an error image will be created and output, as described above, even if you have established an error handler.

Chapter 4. PHPlot Advanced Topics

This chapter documents advanced PHPlot programming topics, going beyond the material in [Chapter 3, *PHPlot Concepts*](#).

4.1. Custom PHPlot Class

This section describes how to create a custom PHPlot class.

If you have a number of applications that use PHPlot, and you want to standardize some of the PHPlot default settings, you can define your own class which extends the PHPlot class and changes the default settings. Here is a short example of a custom PHPlot class, which changes the following defaults:

- Use a TrueType font for all text

- Change the default image size to 800 x 600

- Change the default title colors to red

To extend the PHPlot class, declare your class as shown below. Make sure your class constructor calls the PHPlot class constructor before changing any settings.

```php
<?php
# Define a custom PHPlot class

// Load the PHPlot class first
require_once 'phplot.php';

// Define a class which extends PHPlot:
class my_phplot extends PHPlot {
  function __construct($width=800, $height=600, $out=NULL, $in=NULL)
  {
    parent::__construct($width, $height, $out, $in);
    $this->SetDefaultTTFont('LiberationSans-Bold'); // System dependent
    $this->SetTitleColor('red');
  }
}
```

To use this custom PHPlot class, use `require_once` to include the file containing the class definition, then create an instance of the custom class.

```php
$plot = new my_phplot();
```

You can then use the `$plot` object exactly the same as you might use any other PHPlot object.

4.2. Truecolor Images

This section contains information about using [Truecolor images](#) in PHPlot. This material supplements the text in the [Section 3.5, "Colors"](#).

Truecolor image support was added to PHPlot-5.1.1. With Truecolor image support, you can:

- Create images with a larger number of colors

- Control color transparency with alpha blending

- Perform advanced image processing operations

An example of using Truecolor with PHPlot can be found in Section 5.24, "Example - Using Truecolor To Make a Histogram".

4.2.1. Using Truecolor Images in PHPlot

To make a Truecolor image in PHPlot, create an object of the derived class `PHPlot_truecolor` instead of the class `PHPlot`. For example, replace this:

```
$plot = new PHPlot(800, 600);
```

with this:

```
$plot = new PHPlot_truecolor(800, 600);
```

That is all you need to do in order to create truecolor images. All PHPlot methods are compatible with PHPlot_truecolor objects. An image file produced from a PHPlot_truecolor object with no other programming changes will be the same as an image file produced from a PHPlot object except as described under Section 4.2.5, "Image Formats and File Formats, Palette and Truecolor".

One of the advantages of truecolor images is the ability to use variable transparency. This is described in the next two sections.

4.2.2. Understanding Variable Transparency (Alpha)

Colors in a truecolor image have four components: red, green, blue, and alpha. The alpha component corresponds to the transparency of a color. An alpha value of zero means the color is opaque, and an alpha value of 127 means the color is transparent, or clear.[1] In between values, from 1 to 126, correspond to various amounts of transparency.

Transparency is only meaningful when drawing objects on top of objects, or objects on top of the image background. An object drawn with an opaque color (alpha=0) will replace whatever was in the image before the object was drawn at that position. An object drawn with an transparent color (alpha=127) is invisible and does not affect the appearance of the image. An object drawn with a color that has an alpha value between 1 and 126 will be combined with whatever was in the image before the object was drawn using alpha blending.

The PHP Manual explains alpha blending like this: "In blending mode, the alpha channel component of the color supplied to all drawing functions determines how much of the underlying color should be allowed to shine through. As a result, gd automatically blends the existing color at that point with the drawing color, and stores the result in the image. The resulting pixel is opaque."[2]

Note

Note that the PHP Manual says the resulting pixel is opaque. This means that objects drawn with alpha above 0 are partially or completely transparent only relative to other objects in that same image. This does not result in an image with transparent portions which would show through to a browser or desktop background,

---

[1] PHPlot follows the GD Library convention here. Other systems use alpha=0 to mean transparent, and a maximum alpha value to mean opaque.
[2] From the PHP Reference Manual, imagealphablending

for example. (Read the PHP Manual page for imagesavealpha for more about this behavior and how to change it.) Use SetTransparentColor to make portions of an image transparent to web page or desktop backgrounds.

The following figure shows the effect of alpha blending when drawing lines. The left side shows the normal overlaying of lines, and the right side shows alpha-blended overlaying of lines with alpha = 60 (that is, 60/127 transparency). The effect of alpha blending can be seen where the data lines cross. Note: These plots use wide lines (3 pixels) and the portions of the images are magnified 2x to show detail.
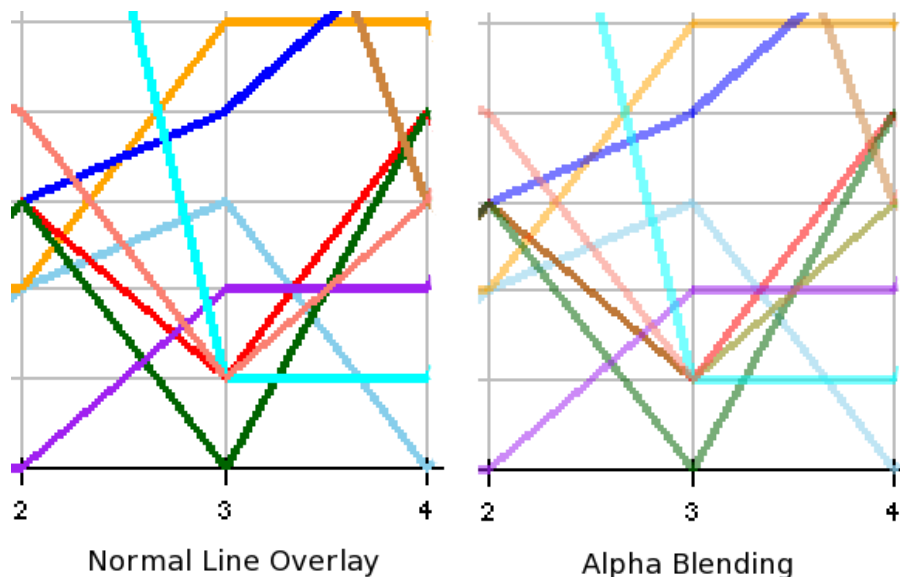
4.2.3. Using Variable Transparency (Alpha) in PHPlot

To use partially transparent colors (that is, colors with an alpha channel) with a PHPlot_truecolor object, you can specify an alpha value as part of a color specification, and you can specify a default alpha value for all data colors.

Use of alpha values with a color specification is described below, in Section 4.2.4, "Color Parameter Form Extensions". Here are some examples of using colors with an alpha specification.

This sets the color used for labels to red=192, green=192, blue=33, and alpha=50 (that is, 50/127 transparency).

```
$plot->SetTextColor(array(192, 192, 33, 50));
```

This sets the color used for tick marks to the color 'blue' from the color map, with alpha value 64 (64/127 transparency).

```
$plot->SetTickColor('blue:64');
```

This sets colors for the first three data sets to red, green, and blue with different alpha values. The three colors are represented using different formats for illustration purposes.

```
$plot->SetDataColors(array(
      array(255, 0, 0, 60), // Red with alpha=60
      '#00ff0050',          // Green with alpha=80 (0x50)
      'blue:70'));          // Blue with alpha=70
```

Instead of specifying the alpha value for each data set color, you can provide a default alpha value for all data colors using the third argument to This uses the colors specified in $my_color_array with a default alpha of 50. The default is applied to any color definition which does not already have an alpha value. SetDataColors.

```
$plot->SetDataColors($my_color_array, NULL, 50);
```

This can also be used to apply an alpha value to the default data colors. This retains the default data colors, but applies alpha = 50 (50/127 transparency) to all the colors. This is a quick way to get partially transparent data colors without re-specifying all the colors.

```
$plot->SetDataColors(NULL, NULL, 50);
```

4.2.4. Color Parameter Form Extensions

In addition to the forms specified in Section 3.5.1, "Color Parameter Forms", colors specifications can include an alpha value. Although this works with palette images as well as truecolor images, specifying alpha values with palette images provides limited value.

1. A color name, as defined by SetRGBArray or from a built-in color map if SetRGBArray was not called, followed by a colon and an alpha value as a decimal number, for example: 'red:60'. The alpha value is between 0 (opaque) and 127 (transparent). Note that colors in the color map can be defined with or without an alpha value. An alpha value appended to the color name overrides any specified in the color map. For example, if the color 'red2' is defined in the color map as array(255,0,0,80) - that is, red with 80/127 transparency - then 'red2' has alpha of 80, and 'red2:40' has alpha of 40.

2. Numeric color component values, in the form #rrggbbaa. Here rr is red, gg is green, and bb is blue, and each component value is represented as a 2-digit hexadecimal number between 00 and ff. Also aa is alpha, represented as a 2 digit hexadecimal number between 00 and 7f. For example, #00ff0010 is green with 16/127 transparency.

3. A PHP array of red, green, blue, and alpha color component values. Each value of red, green, and blue are in the range 0 to 255 inclusive, and the alpha component is in the range 0 to 127 inclusive. For example, array(0,255,0,16) is the same green with 16/127 transparency.

4.2.5. Image Formats and File Formats, Palette and Truecolor

PHPlot can produce JPEG, PNG, and GIF image files (and possibly others). You select the PHPlot output image file format with SetFileFormat.

PHPlot works with GD images before producing an image file. There are two types of GD images: truecolor and palette. Truecolor images represent pixels as 32 bit values, combining 8 bits each of red, green, and blue components with a 7 bit alpha (transparency) value. Palette images use a color table with at most 256 entries, and represent pixels as 8 bit indexes into the color table. The palette image color table entries have 32 bit values, with the same components as truecolor image pixel values. So palette images in GD can have at most 256 unique colors, but there is no limitation on the number of unique colors in truecolor images.

As long as you don't specify a background image when creating your plot object, truecolor images are created with the PHPlot_truecolor class, and palette images are created with the PHPlot class. If you specify a background image, the GD image created by PHPlot matches the type - truecolor or palette - of your background image file. More on background image files can be found in Section 4.2.7, "Background Images" below.

What happens when you output the GD image to an image file depends on the image file format you select.

JPEG image files are always truecolor. Whether you have a GD palette image or truecolor image, you will get a truecolor image file. Note: You are discouraged from using JPEG images with PHPlot, because they are not optimal for this type of graphical information due to use of lossy compression.

GIF image files are always palette type, limited to 256 colors. If you have a GD palette image, you will get a palette GIF image file with the colors you used in your plot. If you have a a GD truecolor image, GD will convert your image to palette format, reducing the number of colors to 256 if necessary. This may change the appearance of your plot. Note that some versions of the PHP manual for `imagecreatetruecolor()` incorrectly state that you cannot output a GIF file from a truecolor GD image.

PNG image files support truecolor images and palette images of various color depths. If you have a GD palette image, you will get a palette PNG image file. If you have a GD truecolor image, you will get a truecolor PNG image file. Note that by default, even though PNG truecolor image files support an alpha channel, GD eliminates the alpha channel when producing a PNG file. The visual effects of alpha blending are reproduced using opaque colors. GD apparently does this due to poor support in viewers for alpha channels. Refer to the PHP Manual page on `imagesavealpha()` for details.

The following figure shows the relationship between constructor, background image format, GD image type, and image file format.
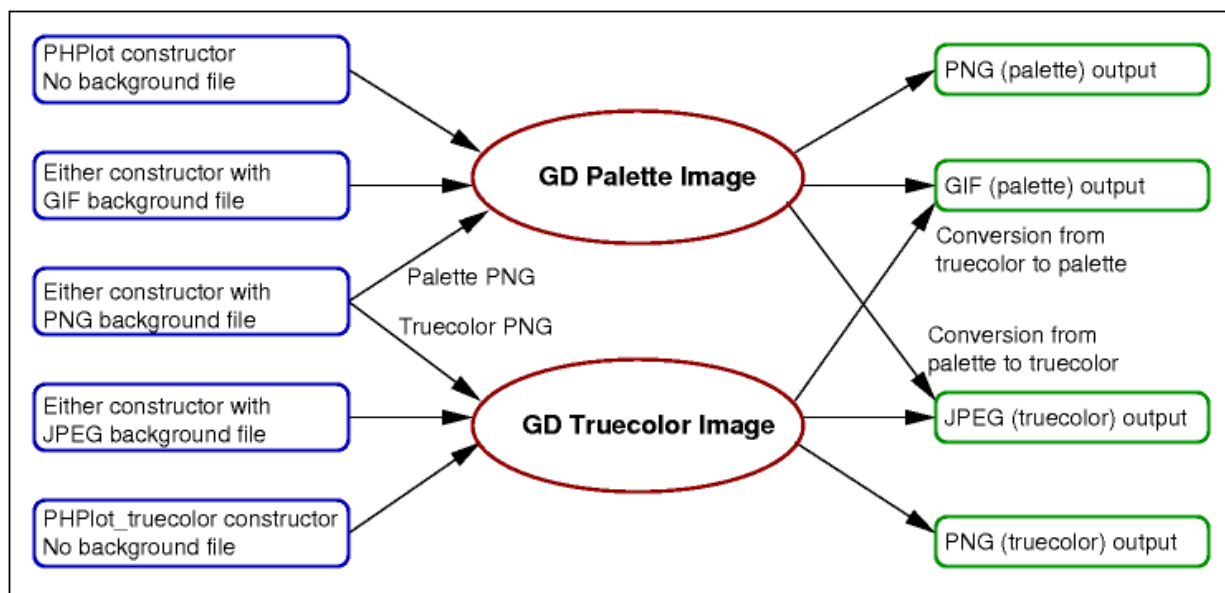
In the initial release of Truecolor support in PHPlot-5.1.1, alpha channel information was ignored when using a PHPlot object, and only used with a PHPlot_truecolor object. This was changed in PHPlot-5.1.2, and alpha channel information is used for both PHPlot and PHPlot_truecolor classes. However, alpha channel information is not always useful with palette images. More on this can be found in Section 4.2.9, "Palette Images and Advanced Color Features" below.

4.2.6. Truecolor Images and Plot Types

All PHPlot plot types work with truecolor images, but not all plot types work well with alpha blending of data colors.

Pie Charts
> Avoid using alpha blending with pie charts. The underlying GD routines do not fill the pie areas in a way that allows proper blending of colors. Flat pie charts (using SetShading(0)) are not too bad, showing some artifacts, but shaded or 3D-look pie charts are poorly rendered.

Bar Charts , Stacked Bar Charts

> Bars are drawn properly, but the 3D shading affects get blended, resulting in less than ideal appearance. Flat, outlined bars (using SetShading(0)) are fine with transparency, but when shading is on the 3D shadows overlap portions of the bars. With alpha blending, the overlaps take on new colors.

4.2.7. Background Images

When creating a PHPlot or PHPlot_truecolor object, you can provide an existing image filename to the constructor as the fourth argument, *$input_file*.

```
$plot = new PHPlot(800, 600, NULL, 'myimage.png');
```

This image file becomes the background for your plot. (The function SetInputFile also does this, but is deprecated for use except through the constructor.)

If you provide an input file to the constructor, the image associated with your PHPlot or PHPlot_truecolor object takes on the type of the input file: palette or truecolor. It does not matter which constructor you use when specifying an input file as background. (This was changed after the initial release of truecolor support. In PHPlot-5.1.1, you must use the PHPlot_truecolor constructor in order to use some truecolor features even when you use a truecolor background image file. Starting with PHPlot-5.1.2 you can use either constructor.)

Note

> The above applies only when an input file is specified to the PHPlot or PHPlot_truecolor constructor. It does not apply to an image background set with SetBgImage nor to a plot area background set with SetPlotAreaBgImage.

4.2.8. Additional Operations on Truecolor Images Using Callbacks

Advanced operations on truecolor PHPlot images are possible using PHPlot callbacks. See Section 4.3, "Callbacks" for more information about using callbacks. Here are some of the operations you can perform, and the corresponding GD functions. Refer to the GD section of the PHP Manual for more information on these functions.

Note

> Some of these functions are only available when PHP was built with the bundled version of the GD library.

4.2.8.1. imageantialias()

You can turn on anti-aliasing of truecolor images. This must be done before anything is drawn, so the pre-drawing callback `draw_setup` is used. Here is a partial example:

```
function pre_plot($img)
{
    imageantialias($img, True);
}
...
$plot = new PHPlot_truecolor(1024, 768);
$plot->SetCallback('draw_setup', 'pre_plot');
```

Note: There are limitations with anti-aliased images. You cannot use wide lines (SetLineWidths). Patterned lines do not work, so if you are displaying X or Y grid lines you must use SetDrawDashedGrid(False) to make these solid.

Also note that TrueType Font (TTF) text is always anti-aliased, even on palette images, regardless of the use of `imageantialias()`.

4.2.8.2. imagealphablending() and imagelayereffect()

These functions control the combining of partially transparent colors. They can be used via a `draw_setup` callback, in the same way as `imageantialias` in the example above. Note that alpha blending is on by default with all truecolor images.

4.2.8.3. imagegammacorrect()

You can have the GD library perform gamma adjustment on a truecolor image. This must be done after all drawing, so the post-drawing callback `draw_all` is used. Here is a partial example:

```
function post_plot($img)
{
    imagegammacorrect($img, 1.0, 0.5); // Input gamma=1, output gamma=.5
}


...
$plot = new PHPlot_truecolor(1024, 768);
$plot->SetCallback('draw_all', 'post_plot');
```

4.2.9. Palette Images and Advanced Color Features

You will have a GD palette image if you use the `PHPlot` constructor without a background image file, or if you use either the `PHPlot` or `PHPlot_truecolor` constructors with a background image file that is a palette image (GIF or some types of PNG). You can use alpha color specifications with palette GD images, but this is not recommended. The results are not well documented, but the following behavior has been observed:

- There is no alpha blending. Drawing operations simply replace existing pixels values with the new pixel values. (These are actually index values into the color table.)

- Alpha values are ignored when the image is output to a JPEG or GIF file. All colors are output as opaque.

- Alpha values are preserved in PNG image files. These will be palette, not truecolor, PNG images, with the color table containing the alpha values. You can therefore have palette PNG files with partial transparency, however not all viewers properly support this.

Nothing described in Section 4.2.8, "Additional Operations on Truecolor Images Using Callbacks" works with palette images, including gamma adjust and anti-aliasing (except that TrueType Font text is always anti-aliased.)

4.3. Callbacks

This section documents the callback feature in PHPlot.

Callbacks allow a programmer using PHPlot to insert their own functions into the graph drawing process. Callbacks are currently also used for development and testing of PHPlot.

Warning

All PHPlot class variables, and all methods/functions which are not documented in the "Reference" section of the PHPlot Reference Manual, are considered to be for internal use and are subject to be changed or removed

at any time. If you call internal functions, or access internal class variables, you greatly increase the risk of breaking your application with future PHPlot releases.

4.3.1. Callbacks Application Interface

Refer to these entries in the Function Reference:

- SetCallback - Register a callback function

- GetCallback - Return a currently registered callback function

- RemoveCallback - Unregister a callback function

Either a function name or an object and method can be registered as a callback with SetCallback. For more information about using callbacks with objects and methods, see the PHP manual under Types, Pseudo Types, Callback [http://www.php.net/manual/en/language.pseudo-types.php#language.types.callback] and the documentation for the PHP call_user_func [http://www.php.net/manual/en/function.call-user-func.php] function. Also refer to Section 4.3.4, "Object Methods as Callbacks" later in this manual. Whether calling a function or an object method as a callback, the same calling sequence is used.

```
function_name($img, $passthrough_arg, [other_args...])
```

$img
    The GD image resource for the plot image.

$passthrough_arg
    The third argument supplied to SetCallback ($arg) when the callback is established. This allows the programmer to pass information to the callback without using global variables. This can be any PHP type including array. To pass a reference, you should put it into an array and pass the array.

other_args...
    Zero or more additional arguments supplied by PHPlot to callbacks of this type. Refer to Section 4.3.3, "Available Callbacks" to see what callback reasons supply extra arguments.

For example, given this callback setup:

```
$plot->SetCallback('draw_graph', 'my_drawing_callback', $myvar);
```

Then PHPlot will call:

```
my_drawing_callback($img, $myvar_value, $plot_area);
```

Where $myvar_value is the value of $myvar at the time SetCallback was called. (The plot_area parameter is only supplied for the draw_graph callback in PHPlot-5.1.0 and later.)

Some callbacks are expected to return a value. This is documented in Section 4.3.3, "Available Callbacks". In all other cases, the return value from a callback function is ignored. (Callbacks which return a value were implemented in PHPlot-5.1.3.)

4.3.2. Callback Function Access

By default, the callback function has access only to the GD image resource as the $img argument, the pass-through argument provided when the callback function was registered, and additional arguments (if any) provided by PHPlot for the callback. It does not have access to the PHPlot class object instance, nor any of its contents.

If you need access to the internals of the PHPlot class instance from your callback, you have three options.

1. You can declare your PHPlot class instance variable as *global*.

2. You can pass the instance variable as the $arg when registering the callback. With PHP5, this will pass a reference to the object, which allows reading and changing variables. (PHP4 would pass a copy of the object at the time the callback function is being set up, which would probably not be useful. There are work-arounds, but PHP4 is no longer supported.)

3. You can use a class method which extends PHPlot. This is described in Section 4.3.4, "Object Methods as Callbacks".

As stated in the warning at the top of this section, any access to the class internals is risky and subject to break with any new update to PHPlot.

4.3.3. Available Callbacks

This section defines the currently available callback names. A callback name is also called a *reason*.

Most of the callbacks currently available are drawing callbacks, activated during the graph drawing process started by DrawGraph. By convention, a drawing callback occurs right after the event which it names. For example, the **draw_titles** callback will be called after drawing the plot titles.

Debug callbacks are for use when developing and debugging PHPlot itself. Needless to say, their use for other purposes is discouraged.

The following table lists the available callback reasons.

Callback Name:	Calling Point:	Extra Parameters:	Notes:
data_color	Every time a color is needed for a data element.	$row, $col, $extra	The callback is expected to return an integer color index into the data colors array. This is for custom color selection. For more information, see Section 4.4, "Custom Data Color Selection".
draw_setup	After all setup, before drawing anything.	(None)	Anything drawn here will be covered by the background.
draw_image_background	After drawing the image backgrounds and border.	(None)	
draw_plotarea_background	After drawing the plot area background.	plot_area	plot_area parameter was added in PHPlot-5.1.0
draw_titles	After drawing the plot title, X and Y titles.	(None)	Called even if no titles were set.
draw_axes	After drawing the X and Y axis and grid lines.	(None)	Not called for pie charts.
draw_graph	After drawing the body of the graph.	plot_area	plot_area parameter was added in PHPlot-5.1.0

Callback Name:	Calling Point:	Extra Parameters:	Notes:
draw_border	After drawing the plot area border.	(None)	Not called for pie charts.
draw_legend	After drawing the legend, if legend is enabled.	(None)	Not called if no legend was set.
draw_all	After all drawing is complete.	plot_area	Added in PHPlot-5.1.0
debug_textbox	Just before drawing any text.	$px, $py, $bbox_width, $bbox_height	Provides access to the orthogonal bounding box position and size for the text string.
debug_scale	Called at end of many scale calculation functions.	Function name, then an array of variable name => value	For displaying intermediate values in margin and scale calculations.

Notes:

Several of the drawing callbacks include `plot_area` as an extra parameter. This is an array of 4 values that define the plot area within the image, in GD pixel coordinates, as left_x, top_y, right_x, and bottom_y. For more information, see Chapter 7, *PHPlot Plot Layout*.

See Section 4.3.5, "Using Callbacks to Annotate Plots" for information on using the drawing callbacks to annotate your plot.

4.3.4. Object Methods as Callbacks

The callback function argument to SetCallback can be an array of two elements: a class variable and a method. This can be used with any class, but here we are interested in using an extension of the PHPlot class. Consider the following setup:

```
class my_PHPlot extends PHPlot
{
  function my_PHPlot($width=600, $height=400, $outfile=NULL, $infile=NULL)
  {
    $this->PHPlot($width, $height, $outfile, $infile);
  }

  function callback($img, $arg)
  {
    fwrite(STDERR, "callback in object\n");
    fwrite(STDERR, "Plot area: ({$this->plot_area[0]}, {$this->plot_area[1]}) :");
    fwrite(STDERR, " ({$this->plot_area[2]}, {$this->plot_area[2]})\n");
  }
}
```

We define a class which extends PHPlot, and a method 'callback' which displays the plot area using the internal PHPlot class variable plot_area. (Note we are using a PHP4-style constructor, which also works with PHP5. You can use the PHP5 constructor method instead.)

We will then create an instance of the extended class, and set a callback.

```
$plot = new my_PHPlot(400,300);
```

```
$plot->SetCallback('draw_titles', array($plot, 'callback'));
```

This is for PHP5. For PHP4, you need to use a reference to the $plot instance. Note that PHP4 is no longer supported.

When the draw_titles callback is triggered, it will call the 'callback' method inside our extended class. Because this is an extension of the PHPlot class, it has access to all the member variables via $this.

4.3.5. Using Callbacks to Annotate Plots

This section contains information about using PHPlot callbacks to annotate a plot with text and graphics. This is an advanced topic, and requires some knowledge of both PHPlot and the PHP GD extension.

Warning

The information in this section uses features which are recent additions to PHPlot, and in some cases uses PHPlot internal variables and functions. As a result, these methods are less likely to work with older releases, and more at risk to change or break in future releases.

This section will first provide general information and advice about annotating plots using callbacks. After, portions of the script from Section 5.22, "Example - Annotating a Plot Using a Callback" will be explained in more detail.

The emphasis here is on using callbacks, but annotation is also possible without callbacks. You can use SetPrintImage(False) to disable automatic output of your image. Then, when DrawGraph returns, you can annotate your plot using GD functions on the `img` member variable of your PHPlot object. Use of callbacks is preferred, however, because it makes your script somewhat less dependent on PHPlot internals (such as the `img` variable).

4.3.5.1. Setting the callback

Use SetCallback to establish a drawing callback. You can find a list of callbacks in Section 4.3.3, "Available Callbacks". The various callbacks with names starting 'draw_' are called at different points in the drawing process. Drawn objects will cover items drawn at an earlier stage. For example, if you draw a line from a 'draw_titles' callback (which is called after the plot titles are drawn, but before the graph is drawn), the line would be 'behind' and possibly covered by the plotted data.

Note that PHPlot does very little except save parameter values until you call DrawGraph. For that reason, you should use GD functions for annotation only from a drawing callback (that is, a callback with a name starting with 'draw_'). The drawing callbacks are called after PHPlot calculations and image resource setup, at which point everything is ready for drawing. In addition, you should not use PHPlot functions which control plot appearance from your drawing callback. These would either have no affect, because it is too late, or produce unexpected results.

4.3.5.2. Coordinates

When drawing with GD, you will use the Device Coordinate system. The coordinates in this system are pixels, with the origin in the upper left corner of your image. Y advances down and X advances to the right.

If you want to make annotations relative to specific values in your plot data, you need to translate those values from World Coordinates to device coordinates. Use the PHPlot function GetDeviceXY to perform this translation. You will need access to your PHPlot object from inside your callback function in order to use this (or any other PHPlot method function). You can make it global, or designate it as the passthrough argument to SetCallback.

If your annotations will fall outside the plot area (for example, in an area you reserved for annotation using SetPlotAreaPixels or SetMarginsPixels, then you need not be concerned with coordinate translation. Of course, you can also add annotations at fixed pixel coordinates inside the plot area, however these may overlay (if done from a draw_graph or later callback) or underlay (if done before the draw_graph callback) the plotted data.

4.3.5.3. Colors

Every GD drawing function you use will require a color value argument. You are recommended to allocate your own colors in your callback using the GD function `imagecolorresolve()`. This function will always return a color index, by either re-using an existing color in the image's color map, or by allocating a new color. Using imagecolorresolve() rather than trying to access the PHPlot internal variables for color indexes will protect your script from breaking if the way PHPlot manages its internal colors ever changes.

4.3.5.4. Text

Text can be added to your plot using GD functions which include `imagestring`, for build-in simple fonts, and `imagettftext` for TrueType font text. To use these functions, you need device coordinates, as described above.

You can also add text to your plot using the PHPlot function `DrawText`. This is documented only for internal use by PHPlot, so there is a risk of future incompatibility. But this function provides support for controlling the text justification, and works better with multi-line text.

4.3.5.5. Example

This example creates a bar chart and adds annotation. The goal is to draw an ellipse and add text to the highest and lowest bars in a bar chart. Refer to Section 5.22, "Example - Annotating a Plot Using a Callback" for the complete script and output from this example.

The script starts with the usual PHPlot object creation and setup.

```
$plot = new PHPlot(800, 600);
$plot->SetTitle('Monthly Widget Sales');
...
```

(For the complete script, see the example referenced above.)

Before calling DrawGraph, establish the drawing callback. This uses the `draw_all` callback, which gets called when all drawing is complete in DrawGraph. (Note: If using PHPlot-5.0.7 or earlier, use 'draw_graph' instead, as 'draw_all' was not yet available.) The name of our callback function is `annotate_plot`, and we are passing the PHPlot object ($plot) as a pass-through parameter. You can use a global or class callback instead - see Section 4.3.1, "Callbacks Application Interface" for more on these options.

```
$plot->SetCallback('draw_all', 'annotate_plot', $plot);
```

Here is the declaration of our callback function. The `$img` parameter is provided by PHPlot itself, and is the GD resource for our image. The `$plot` parameter is the pass-through argument we provided above when establishing the callback. Some callbacks make other parameters available. In fact, 'draw_all' provides the plot area coordinates as an additional parameter, but we don't need that here so we do not have to include that in the function declaration.

```
function annotate_plot($img, $plot)
{
```

As stated above, you should allocate your own colors, rather than trying to get into PHPlot's internals for color values. Here we allocate two colors and assign the color indexes to local variables:

```
$red = imagecolorresolve($img, 255, 0, 0);
$green = imagecolorresolve($img, 0, 216, 0);
```

Next, we want to draw graphics centered on two points in our data. The points were calculated as best_index (X), best_sales (Y), worst_index (X), and worst_sales (Y). In order to draw at these locations, we need to translate the values from World Coordinates to Device Coordinates. This is done using the PHPlot function GetDeviceXY.

```
list($best_x, $best_y) = $plot->GetDeviceXY($best_index, $best_sales);
list($worst_x, $worst_y) = $plot->GetDeviceXY($worst_index, $worst_sales);
```

Now we are ready to draw some ellipses, centered on our two data points. The values 50 and 20 are the width and height, in pixels.

```
imageellipse($img, $best_x, $best_y, 50, 20, $green);
imageellipse($img, $worst_x, $worst_y, 50, 20, $red);
```

As stated above, we have two options for text, and the example uses each method. We can draw text using the GD functions, but we have to do a little more work to position the text. Here the text is approximately centered horizontally and above the data point. (Note ImageString by default uses the upper left corner of the text string for positioning.)

```
$font = '3';
$fh = imagefontheight($font);
$fw = imagefontwidth($font);
imagestring($img, $font, $best_x-$fw*4, $best_y-$fh-10, 'Good Job!', $green);
```

Or, we can use the PHPlot internal function DrawText. With a PHPlot version 5.1.0 and later, we omit the font specification and it will default to the generic font, which can be set with SetFont('generic', ...)

```
$plot->DrawText('', 0, $worst_x, $worst_y-10, $red, 'Bad News!', 'center', 'bottom');
```

4.4. Custom Data Color Selection

This section describes customizing the selection of data colors using a PHPlot callback. The data color callback was added in PHPlot-5.1.3.

4.4.1. Standard Behavior of Data Color Selection

Before explaining how to customize data color selection, here is a review of how data color selection works by default.

Think of your data array as having rows and columns. The rows represent values of the independent variable (usually X), and the columns contain one or more values of the dependent variable (usually Y) for that value of the independent variable. For this discussion, ignore any additional entries in the data array, such as labels and X values. The set of values from a column in your data array is also referred to as a data set.

The standard behavior of PHPlot is to select a data color from the data colors array using the column index for the data point. The selected color will be used to draw a point marker, line segment, bar, etc. This was explained in Section 3.5.3, "Plotting Colors".

For example, if you have a data array with 12 rows and 3 columns for a bar chart, you are drawing 12 groups of 3 bars. Within each bar group, the first bar will be drawn with the first color in the data colors array (the color with index 0), the second bar will use the second color from the data colors array, and the third bar will use the third color. You can see this in Example 5.4, "Bar Chart", where the first three colors in the data colors array are SkyBlue, green, and orange.

There are two other color arrays: the error bar colors and data border colors. Error bar colors are used in error plots to indicate the positive and negative error range, and data border colors are used to outline bars in bar charts when 3D shading is off. The same index (but not necessarily the same color) is used to select the color for any of the three

elements which are used in a plot. For example, the first data set in a points plot with error bars will use data color index 0 for the point markers, and error bar color index 0 for the error bars. The second bar in each group in an unshaded bar chart will use the second data color to fill the bar and the second data border color to outline it.

You can set the colors in the three color arrays with SetDataColors, SetErrorBarColors, and SetDataBorderColors. PHPlot will pad all these arrays to the number of columns in your data array, by duplicating the earlier values. (For example, if you have 5 data sets and define 3 colors red, green, and blue, PHPlot will pad this to be a 5 color array red, green, blue, red, green.) It will not truncate the arrays. This means you can define more data colors than there are data columns. These additional colors will not be used with the standard color selection method, but can be used with custom data color selection.

4.4.2. Custom Data Color Selection

If you need more control over data colors, you can use the PHPlot callback called `data_color`. (See Section 4.3, "Callbacks" for general information about callbacks.) Some of the things you can do with custom data color selection are:

• A bar chart with each bar having a different color.

• A linepoints plot with different colors for the line segments and the point markers.

• A bar chart where the bar color depends on the value of that data point.

> ## Note
>
> Custom data color selection is not available for plot types `pie`, `area`, or `stackedarea`. These three plot types already provide full control over the data color selection, with no need for the callback function, because each color in the color array is only used once.

To customize the use of data colors, you will define a function that accepts as arguments the data array row and column index numbers (0-based integers), and returns the color array index. Register this function with PHPlot as a callback, and your function will be called whenever PHPlot needs to select a data color.

Note that your callback will return an array index, not a color value. For example, if it returns 0, the first color in the data colors array will be used, and the first color in the error bar colors array (if error bars are being drawn), and the first color in the data border colors array (if data borders are being drawn). You will most likely need to set up the data colors array (and possibly the error bar colors array and data border colors array too) in order to get the results you want.

A function to act as a data color callback might look like this:

```
function pickcolor($img, $passthrough, $row, $col, $extra = 0)
{
  $color_index = ...;

  return $color_index;
}
```

The first two arguments are common to all callbacks: the PHPlot image resource, and your passthrough argument (if any - see below). (You generally will not need to access the image resource from the data colors callback, but it is provided to all callbacks.) The second and third arguments specify which data value is being plotted. The $row corresponds to the independent variable (usually X), and $col corresponds to the data set - plot line, bar within a bar group, etc. Both $row and $col are zero based integers indexes.

Your callback is expected to return a color array index for this data point. This will be an integer greater than or equal to zero, where zero indicates the first color in the colors array should be used. Your returned index should be within the bounds of the color array being referenced, however PHPlot will use the value you return modulo the size of the

array. For example, the default PHPlot data colors array has 16 colors. If your callback returns the value 20, the 5th color in the array will be used (because 20 % 16 = 4, and index 4 is the 5th value in the array).

The $extra argument to your callback is for extra information you may need to determine the color to use. Currently, this is only used for 'linepoints' plots and 'linepoints' error plots. These plots are drawn in two stages: points and lines. In case you want different colors for the points and lines, use the $extra argument. It will have the value 1 when PHPlot is requesting the color of the point marker (shape), and the value will be 0 when requesting the color of the line segment. Note that the error bars of a linepoints error plot are drawn with the color index returned for the points (but using the error bars colors, not the data colors).

You do not need to specify the $extra argument in your callback function declaration if you do not need it. But if you do specify it, you must make it an optional argument with the value zero, because PHPlot does not always supply the value.

The above function would be established as a data color callback for a PHPlot object $plot like this:

```
$plot->SetCallback('data_color', 'pickcolor', $passthru_arg);
```

The first argument is the callback name, or 'reason': data_color. The second argument is the name of your callback function. An object and method can be used here instead - see Section 4.3.4, "Object Methods as Callbacks". The third argument is an optional pass-through value that will be sent to your callback function each time it is called.

4.4.3. Custom Data Color Selection Examples

For examples of using a data color callback, see Section 5.25, "Example - Creative Use of the Data Color Callback" and Section 5.26, "Example - Custom Bar Colors Using the Data Color Callback".

4.5. Tuning Parameters

This section documents some PHPlot class member variables that can be used to adjust the appearance of plots. You should rarely find it necessary to change these, and PHPlot does not provide "Set" functions for them.

The class member variables listed in Section 10.1, "List of Member Variables" are generally reserved for use only by the class implementation itself. But there are some adjustments you can make to the appearance of a plot only by changing member variables. This section documents some PHPlot class member variables that alter a plot appearance, but which do not have any defined class functions for you to use to set the values.

For example, if you want PHPlot to draw the X/Y grid above (after) the plot, rather than behind it, you would do the following:

```
$plot = new PHPlot(800, 600);
...
$plot->grid_at_foreground = TRUE; // Draw grid after plot
```

4.5.1. Tuning Bar Charts

These variables affect plot types bars and stackedbars. They are used to control the width of the bars. (For horizontal plots, the "width" of the bars is actually the height.)

bar_extra_space
    Controls the amount of extra space within each group of bars. Default is 0.5, meaning 1/2 of the width of one bar is left as a gap, within the space allocated to the group (see group_frac_width). Increasing this makes each group of bars shrink together. Decreasing this makes the group of bars expand within the allocated space.

group_frac_width

    Controls the amount of available space used by each bar group. Default is 0.7, meaning the group of bars fills 70% of the available space (but that includes the empty space due to `bar_extra_space`). Increasing this makes the group of bars wider.

bar_width_adjust

    Controls the width of each bar. Default is 1.0. Decreasing this makes individual bars narrower, leaving gaps between the bars in a group. This must be greater than 0. If it is greater than 1, the bars will overlap.

If bar_extra_space=0, group_frac_width=1, and bar_width_adjust=1 then all the bars touch (within each group, and adjacent groups).

4.5.2. Tuning OHLC Charts

These variables affect plot types [ohlc](#), [candlesticks](#), and [candlesticks2](#), For candlesticks plots, they adjust the calculation of the width of the candlestick body. For basic OHLC plots, they adjust the calculation of the length of the tick marks which represent opening and closing prices. (All of these were added in PHPlot-5.3.0.)

ohlc_max_width

    This is one half the maximum width in pixels of the candlestick body or OHLC tick mark. The default is 8.

ohlc_min_width

    This is one half the minimum width in pixels of the candlestick body or OHLC tick mark. The default is 2.

ohlc_frac_width

    This is the fractional amount of the available space (plot width area divided by number of points) to use for half the width of the candlestick bodies or OHLC tick marks. The default is 0.3. This needs to be less than 0.5 or there will be overlap between adjacent candlesticks.

PHPlot calculates a value to use for one half the width of the candlestick bodies, or for the OHLC open/close tick mark lengths, as follows:

```
half_width = max(ohlc_min_width, min(ohlc_max_width, ohlc_frac_width * avail_area)
Where avail_area = plot_area_width / number_data_points
```

4.5.3. Tuning the Legend

This variable adjusts the appearance of the legend.

legend_colorbox_width

    This is an adjustment factor for the width of the color boxes in the legend. With the default value 1.0, the color boxes are as wide as one character in the font used in the legend (width of "E" for TrueType fonts). A value of 2.0 makes the color boxes twice as wide, and 0.5 makes them half the character width. (This was added in PHPlot-5.3.0.)

4.5.4. Label Tuning

These variables affect the display of labels.

data_value_label_angle

    This sets the angle, in degrees, for the position of data value labels near the data points they label. Together with `data_value_label_distance`, it determines the position of the reference point for the label. (This does not apply to data value labels for bars or stackedbars plots, as the label position is fixed for these plot types.) The default if unset is 90 degrees, which places the label above the data point. PHPlot automatically selects which text

alignment to use, based on the angle. For example, with the default 90 degree angle, the label will be horizontally centered, vertically bottom aligned. If the angle is 0 degrees, the alignment is horizontally left, vertically centered.

`data_value_label_distance`
> This sets the distance, in pixels, for the position of data value labels near the data points they label. Together with `data_value_label_angle`, it determines the position of the reference point for the label. (This does not apply to data value labels for bars or stackedbars plots, as the label position is fixed for these plot types.) The default if unset is 5 pixels.

4.5.5. Miscellaneous Tuning

These variables affect other aspects of the appearance of a plot.

`grid_at_foreground`
> Controls the order in which certain plot elements are drawn. The default is FALSE, meaning the X axis, Y axis, and grid lines are drawn before the main part of the plot. If TRUE, the X axis, Y axis, and grid lines are drawn after the main part of the plot, which results in the grid lines overlaying the plotted data.

`locale_override`
> Set this to TRUE (or any non-empty value) to prevent PHPlot from loading information about the locale from the operating system. You must do this if you want to override the locale using `setlocale()` from your PHP code, perhaps because your platform does not allow setting the locale from environment variables. See [SetNumberFormat](#) for more information.

`safe_margin`
> This is the amount of space that PHPlot leaves between elements that should not touch. The default is 5 pixels. Changing this is not recommended. The effect is similar to changing the cellpadding on an HTML table.

4.6. Multiple Plots Per Image

This section contains information about producing more than one plot on an image.

Using PHPlot, you can produce more than one plot on a single image. These can be *tiled plots* - separate plots manually positioned within the image, or *overlay plots*. Tiled plots are used when you want to display more than one plot on a single image, for example side-by-side. Overlay plots are used when you want to show more than one type or range of data representation on a single plot. For example, an overlay plot could be used to show two data sets with different Y scales, or to overlay a bar chart with a line plot. You can also combine tiled and overlay plots in a single image.

An example of two tiled plots on an image can be found in [Section 5.18, "Example - Two Plots on One Image"](#). An example of an overlay plot can be found in [Section 5.34, "Example - Overlaying Plots"](#)

4.6.1. Overview of Multiple Plots

When producing multiple plots on an image, a single PHPlot object is used. The overall steps to be followed are:

1. Create a `PHPlot` or `PHPlot_truecolor` object (referred to here as `$plot`).

2. Use `$plot->SetPrintImage(False)` to disable automatic output of the image after a plot is created.

3. Prepare the first plot, including setting the data array, plot type, and any other applicable settings.

4. Use `$plot->DrawGraph()` when complete. This creates the plot, but does not produce any output.

5. Repeat the previous two steps to prepare each additional plot, completing it with `$plot->DrawGraph()`.

6. When all the plots are complete, use `$plot->PrintImage()` to output the completed image.

The sections which follow contain additional information you will need to produce multiple plots on a single image.

4.6.2. Plot Settings with Multiple Plots

In general, PHPlot applies settings made for one plot as defaults for the next plot, when using the same PHPlot class instance. There are some special cases, however, which are discussed in the sections below.

4.6.2.1. Global Settings

Certain plot elements apply to the image as a whole, not to individual plots. PHPlot will draw these at most once per image. (That is, the element will be drawn only the first time `DrawGraph()` is called after the element has been set up.)

- Main title ([SetTitle](#))

- Image background color ([SetBackgroundColor](#)) or image background file ([SetBgImage](#))

- Image border ([SetImageBorderColor](#) and [SetImageBorderType](#))

For example, the first plot on an image that has a main title will result in the main title being drawn. If any subsequent plot (using the same PHPlot instance) also sets a main title, that will be ignored.

4.6.2.2. Data Scaling

Whether you use [SetPlotAreaWorld](#) to set the plot area data range, or you let PHPlot calculate the plot area data range, that range applies to all subsequent plots unless overridden. Even if you set a new data array, the calculated or pre-set data range from the previous plot applies. Without being told otherwise, PHPlot will not re-examine the data array to recalculate the data range. This allows you to re-use an automatically calculated date range, if you want.

If instead you want PHPlot to automatically calculate the data range for additional plots, call `SetPlotAreaWorld()` (with no arguments), or `SetPlotAreaWorld(NULL, NULL, NULL, NULL)`. Either of these forms causes PHPlot to forget about a specified or calculated data range, and it will compute a new range.

Of course, you can also use [SetPlotAreaWorld](#) with parameter values, to manually set all or part of the data range for each plot. Any parameters you do not set (or specify as NULL) will be calculated based on the data array for the current plot. That is, PHPlot will forget about the previous data range once you call [SetPlotAreaWorld](#), regardless of how many non-NULL parameters you use.

When overlaying plots, you will often want all the plots to use the same data scale, so the values can be read off of the axis. Another option is to have two separate Y scales, with one represented on the left side and one on the right side. (See [Section 5.34, "Example - Overlaying Plots"](#) for an example of overlaying plots with two different Y scales.) In some cases it may make sense to overlay plots with different scales and no separate axis, for example when using data value labels, or when the important information is the trend or shape shown by the graph rather than the actual values.

4.6.2.3. Plot Area

You can specify a plot area (window) with [SetPlotAreaPixels](#) or [SetMarginsPixels,](#) or you can let PHPlot calculate a plot area. Whether you set the plot area yourself, or you let PHPlot calculate it, those settings apply to all subsequent plots unless overridden.

This means that if you are doing side-by-side (tiled) plots on an image, you must use [SetPlotAreaPixels](#) or [SetMarginsPixels](#) with each plot, to set the area of the image to be used for that plot. Remember that the plot area does not include the axis labels, tick marks, or titles, so you must leave enough room between and around plots for these.

If you are doing overlay plots, you can let PHPlot calculate the plot area for the first plot, or you can specify the area with SetPlotAreaPixels or SetMarginsPixels. You need not use these for subsequent plots; PHPlot will continue to use the same window, overlaying the additional plots.

However, if you allow PHPlot to calculate the plot area, it will only use information in the first plot to determine the margins. This will not work well if subsequent plots require more margin space. For example, if the first plot has a Y axis title and tick labels only on the left side, and the second overlay plot has a Y axis title and tick labels on the right side, automatic plot area calculation will only leave enough margin space on the left side. As a result, the right side Y axis title and tick labels may fall off the image edge. To avoid this, use either SetPlotAreaPixels or SetMarginsPixels to specify large enough margins.

4.6.2.4. Tick Increment

Tick increments are recalculated for each plot, based on the data range, unless set with SetXTickIncrement or SetYTickIncrement. 'Data range' here refers to that set with SetPlotAreaWorld, or automatically calculated. This differs from the way PHPlot handles the data range, which is not recalculated by default after the first plot.

For overlay plots, if you want to use the same tick increments, you should either set the desired tick increment (doing this for the first plot is sufficient), or make sure the data ranges match.

4.6.2.5. Grid Lines

PHPlot defaults to drawing the dependent variable grid (usually Y), and the grid lines will be drawn at tick positions. As stated above, the tick positions by default will be recalculated for each plot, using the calculated or explicitly set data range.

For overlay plots, you generally do not want to have more than one set of grid lines in each of X and Y, or the results will be confusing. Even if your plot overlays have the same tick increments, avoid having the grid lines drawn more than once. Otherwise, the grid lines for the second plot will overlay the plotted data from the first plot. You can turn off the grid lines with SetDrawXGrid(False) and SetDrawYGrid(False).

4.6.2.6. Legend Positioning

Legend position with multiple plots works differently depending on whether the position is defaulted, set with SetPlotAreaPixels, or set with SetPlotAreaWorld.

- If the legend position is defaulted, a legend will be drawn at the upper right corner of each plot. The same legend will be drawn in each position, unless the contents are changed with SetLegend().

- If the legend position is specified using device coordinates with SetLegendPixels(), the legend will be drawn at those coordinates as measured on the image, once per plot, at the same location. This repeated over-drawing is usually harmless, but if you want to have it drawn only once, either use SetLegend() only before the last plot, or use either SetLegend(NULL) or SetLegend(array()) to cancel the legend after the first plot.

- If the legend position is specified using world coordinates with SetLegendWorld(), the legend will be drawn at the specified world coordinates for each plot. This assumes the specified world coordinates are within the plot area for each plot. As with the default positioning case, the same legend will be drawn for each plot, unless the contents are changed with SetLegend().

4.6.2.7. Plot Area Background

If you set a plot area background color with SetPlotBgColor and SetDrawPlotAreaBackground, or if you set a plot area background image with SetPlotAreaBgImage, this will be applied to each plot until disabled. This works well for side-by-side (tiled) plots, as each will get the same background by default.

If you are overlaying multiple plots on an image, setting a plot area background color or image for one plot will result in that background hiding previous plots. Therefore, you need to set up the background for the first plot, then turn it off it for the second plot. If you previously set a plot area background image, you can disable it for subsequent plots with `SetPlotAreaBgImage(NULL)`. If you previously set and enabled a plot area background color, you can disable it for subsequent plots with `SetDrawPlotAreaBackground(FALSE)`.

4.6.2.8. Axis Positioning

You can position the X and Y axis manually with [SetXAxisPosition](#) and [SetYAxisPosition](#), or you can let PHPlot calculate the axis positions for you. Whether you set the positions yourself, or let PHPlot calculate them for you, those positions apply to subsequent plots unless overridden. Even if you set a new data array, PHPlot will not recalculate the axis positions unless told to.

If you want PHPlot to automatically re-calculate the X axis position for a subsequent plot, use `SetXAxisPosition()` (with no arguments), or `SetXAxisPosition('')`. To restore automatic Y axis position calculation, use `SetYAxisPosition()` or `SetYAxisPosition('')`.

4.6.3. Summary - Tiled Multiple Plots

Here are some guidelines for tiling multiple plots:

- Unless all plots will use the same X and Y data ranges, use [SetPlotAreaWorld](#) with each plot. Call the function with no arguments to have PHPlot automatically calculate the data range for the plot, or supply arguments to explicitly set a data range.

- Use [SetPlotAreaPixels](#) to set the area within the image for each plot. Remember to leave room for axis labels and titles.

- If you want a legend for each plot, use [SetLegendPixels](#) or [SetLegendWorld](#) to position it. Or let the position default to the upper right corner of each plot.

- You can only have one main title for the entire image.

4.6.4. Summary - Overlay Multiple Plots

Here are some guidelines for overlaying multiple plots:

- All plots will use the same data scaling by default, whether automatically calculated by PHPlot or set with [SetPlotAreaWorld](#). Use [SetPlotAreaWorld](#) if you want different data scaling for subsequent plots.

- You can let PHPlot calculate the plot window by default, but it will not account for additional margin space needed by plots after the first. Instead, you can use [SetPlotAreaPixels](#) or [SetMarginsPixels](#) to set a specific plot area to use for all plots.

- Set the tick increments you want for each plot, especially if the data ranges differ. You can have two sets of tick marks and labels if you position them on the opposite sides of the plot area.

- Draw grid lines, if you want them, only for the first plot, and turn them off for the second plot.

- If you want a single legend, either set it up for the last plot, or set it up for any plot and cancel it for the next plot. If you want multiple legends, one per overlay, position them manually with [SetLegendWorld](#) or [SetLegendPixels](#).

- If you want a plot area background, you must set it for the first plot and cancel it for the second plot, or it will hide the plots.

4.6.5. Multiple Plots - History

A number of fixes were made in PHPlot-5.3.1 that affect multiple plots per image. If you are creating multiple plot images using PHPlot-5.3.0 or earlier, you should upgrade to the latest release. If you are unable to upgrade, you may need to work around the following issues:

- Color allocation: In PHPlot-5.2.0 and PHPlot-5.3.0, the data color array (whether defaulted or set with SetDataColors) was truncated to the number of colors required for a plot. This means that the additional colors were not available for subsequent plots, so the data colors would repeat. For example, if plot #1 used 3 colors for 3 data sets, and plot #2 had 5 data sets, only 3 colors were available and the first two colors would be reused for the 4th and 5th data sets. To work around this, you can reload the data colors before each subsequent plot. To reload the default data colors, use `$plot->SetDataColors(False)`. Another work-around is to define a custom data color callback, which turns off the color slot optimization.

- Legend positioning using SetLegendWorld was not correctly applied to subsequent plots in an image through PHPlot-5.3.0. If you have multiple side-by-side plots and you want the legend in the same world coordinate position in each plot, you still need to use `SetLegendWorld(..., ...)` when creating each plot.

- Through PHPlot-5.3.0, there was no way to reset the X axis position or Y axis position to the default of automatic positioning. That is, `SetXAxisPosition()`, `SetXAxisPosition('')`, `SetYAxisPosition()`, and `SetYAxisPosition('')` did not work. There is no work-around to get automatic positioning of the axis lines.

- Through PHPlot-5.3.0, several functions had more restricted usage when resetting to defaults.

  - Use `SetLegendPixels(NULL, NULL)` rather than `SetLegendPixels()`.

  - Use `SetNumXTicks('')` and `SetNumYTicks('')` rather than `SetNumXTicks()` and `SetNumYTicks()`.

  - Use `SetLegend(array())` rather than `SetLegend(NULL)`.

Chapter 5. PHPlot Examples

This chapter contains examples of plots produced with PHPlot.

Each of the following PHPlot examples shows an image, followed by the PHP script which produced that image. Each script is self-contained (needing only PHPlot), so you can copy it from this manual and run it with PHP to produce the image. Note that some of the scripts may require the latest version of PHPlot.

Note

The PHP CLI (command line interface), used to generate the examples here, never outputs HTTP headers. So it isn't necessary to use SetIsInline to suppress headers when using the CLI. This is a useful method you can use to debug and test your own PHPlot scripts without having to modify them for stand-alone use. Also, by using the CLI instead of a web server and browser, you can more readily see any error messages. Run your PHPlot scripts with the PHP CLI like this (using the ImageMagick [http://www.imagemagick.org/] display program to view the results):

```
$  php myscript.php > output.png
$  display output.png
```

ImageMagick is available for several operating systems. There are many other image viewers for Linux and Linux-like systems, including qiv and gqview.

5.1. Example - Line Plot

This is a simple line plot with a single data set. Data type 'data-data' is used to include the X values (the years) in the data points.

Example 5.1. Line Plot
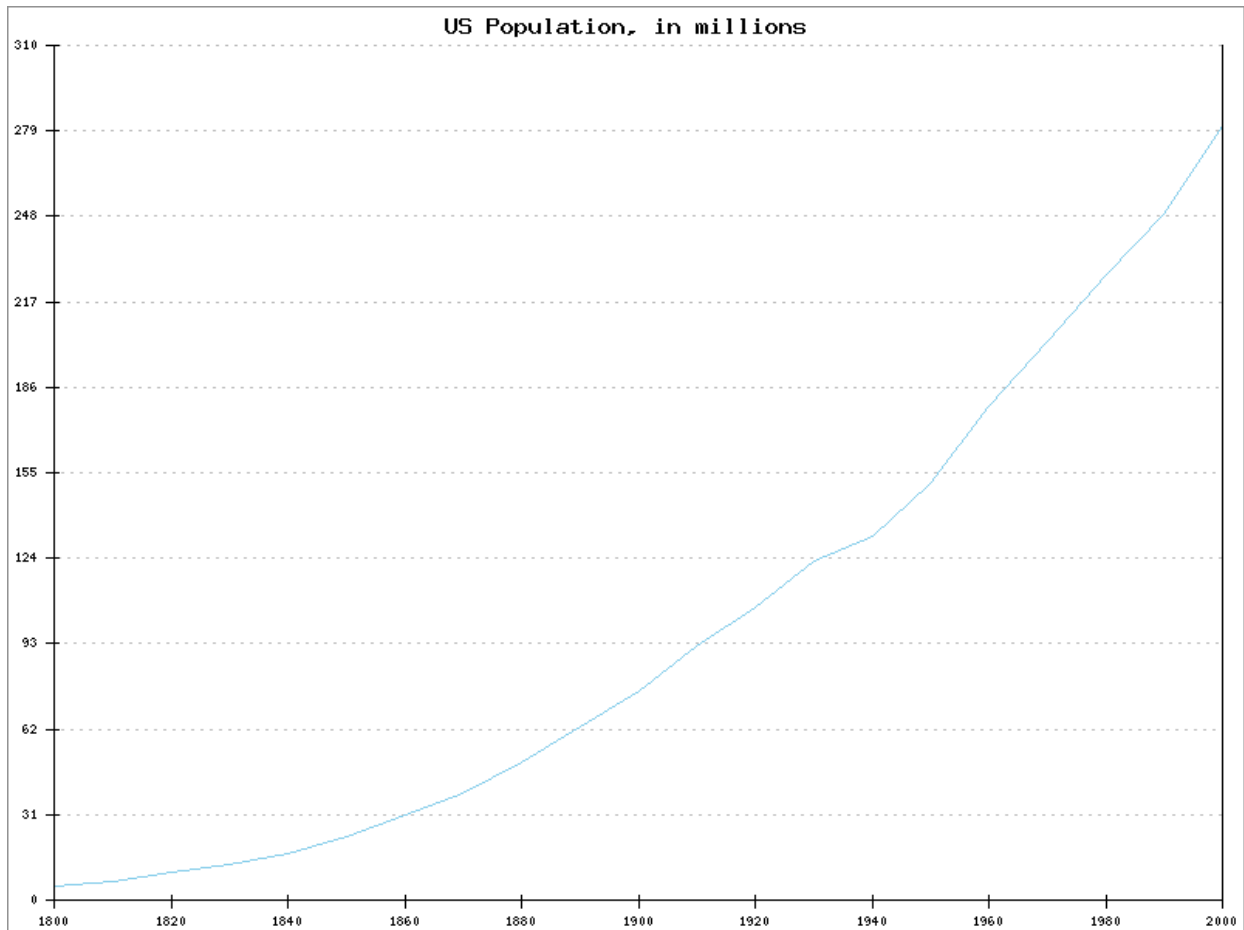

```php
<?php
# PHPlot Example: Simple line graph
require_once 'phplot.php';

$data = array(
  array('', 1800,    5), array('', 1810,    7), array('', 1820,   10),
  array('', 1830,   13), array('', 1840,   17), array('', 1850,   23),
  array('', 1860,   31), array('', 1870,   39), array('', 1880,   50),
  array('', 1890,   63), array('', 1900,   76), array('', 1910,   92),
  array('', 1920,  106), array('', 1930,  123), array('', 1940,  132),
  array('', 1950,  151), array('', 1960,  179), array('', 1970,  203),
  array('', 1980,  227), array('', 1990,  249), array('', 2000,  281),
);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');
```

```
$plot->SetPlotType('lines');
$plot->SetDataType('data-data');
$plot->SetDataValues($data);

# Main plot title:
$plot->SetTitle('US Population, in millions');

# Make sure Y axis starts at 0:
$plot->SetPlotAreaWorld(NULL, 0, NULL, NULL);

$plot->DrawGraph();
```

5.2. Example - Line Plot: Functions

This is a line plot showing the graph of sin(x) and cos(x). This uses quite a few of the PHPlot style control functions to tune the appearance of the plot.

Example 5.2. Line Plot: Functions
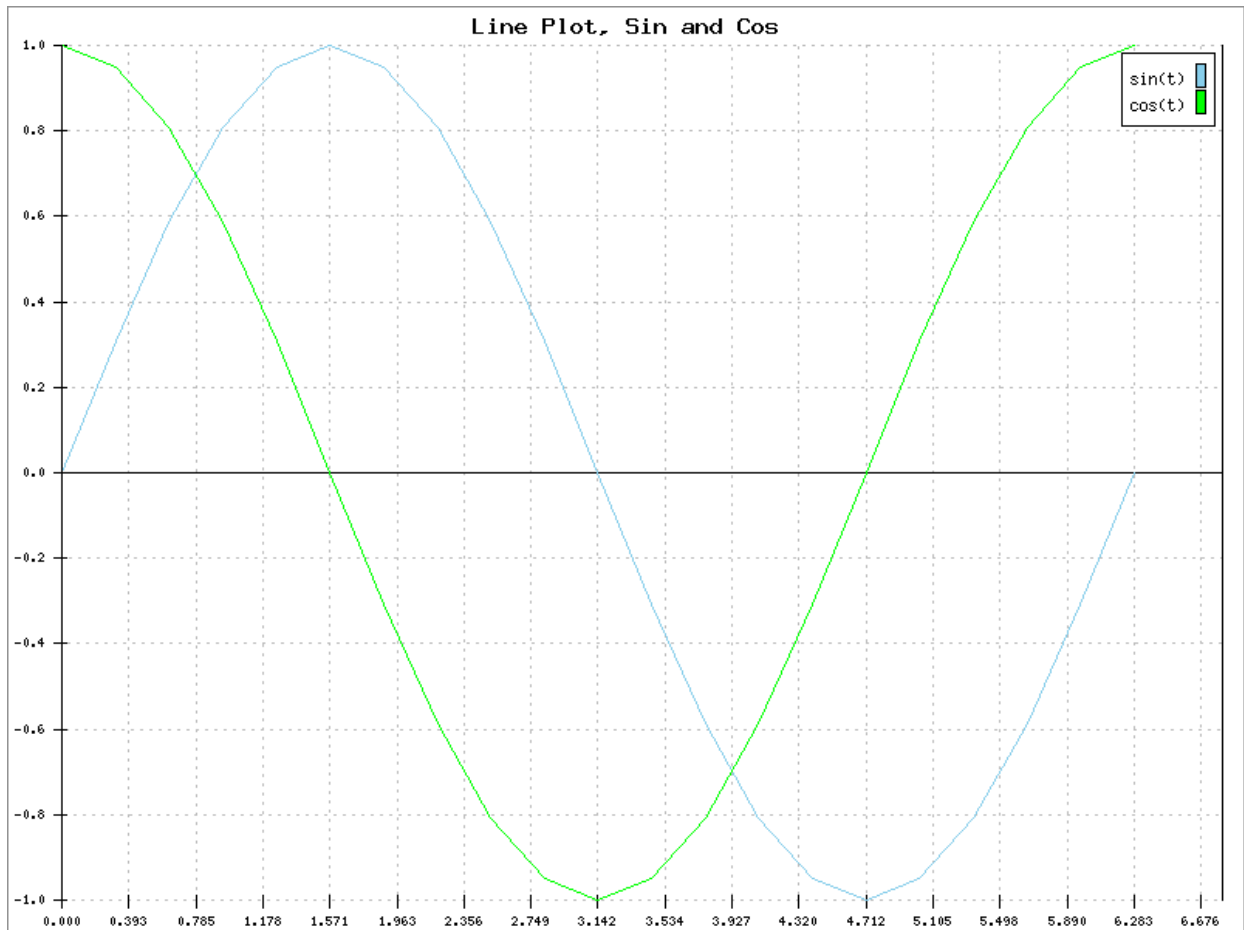

```php
<?php
# PHPlot Example: Line graph, 2 lines
require_once 'phplot.php';

# Generate data for:
#    Y1 = sin(x)
#    Y2 = cos(x)
$end = M_PI * 2.0;
$delta = $end / 20.0;
$data = array();
for ($x = 0; $x <= $end; $x += $delta)
  $data[] = array('', $x, sin($x), cos($x));

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');
```

```
$plot->SetPlotType('lines');
$plot->SetDataType('data-data');
$plot->SetDataValues($data);

# Main plot title:
$plot->SetTitle('Line Plot, Sin and Cos');

# Make a legend for the 2 functions:
$plot->SetLegend(array('sin(t)', 'cos(t)'));

# Select a plot area and force ticks to nice values:
$plot->SetPlotAreaWorld(0, -1, 6.8, 1);

# Even though the data labels are empty, with numeric formatting they
# will be output as zeros unless we turn them off:
$plot->SetXDataLabelPos('none');

$plot->SetXTickIncrement(M_PI / 8.0);
$plot->SetXLabelType('data');
$plot->SetPrecisionX(3);

$plot->SetYTickIncrement(0.2);
$plot->SetYLabelType('data');
$plot->SetPrecisionY(1);

# Draw both grids:
$plot->SetDrawXGrid(True);
$plot->SetDrawYGrid(True);

$plot->DrawGraph();
```

5.3. Example - Area Plot

In the area plot, PHPlot fills the area from each data set down to the next data set, or to the X axis for the last data set. For this example, the data was prepared such that the data sets are cumulative percentages. (See also Example 5.21, "Stacked Area Plot" which produces a similar plot using a different data representation.)

Example 5.3. Area Plot
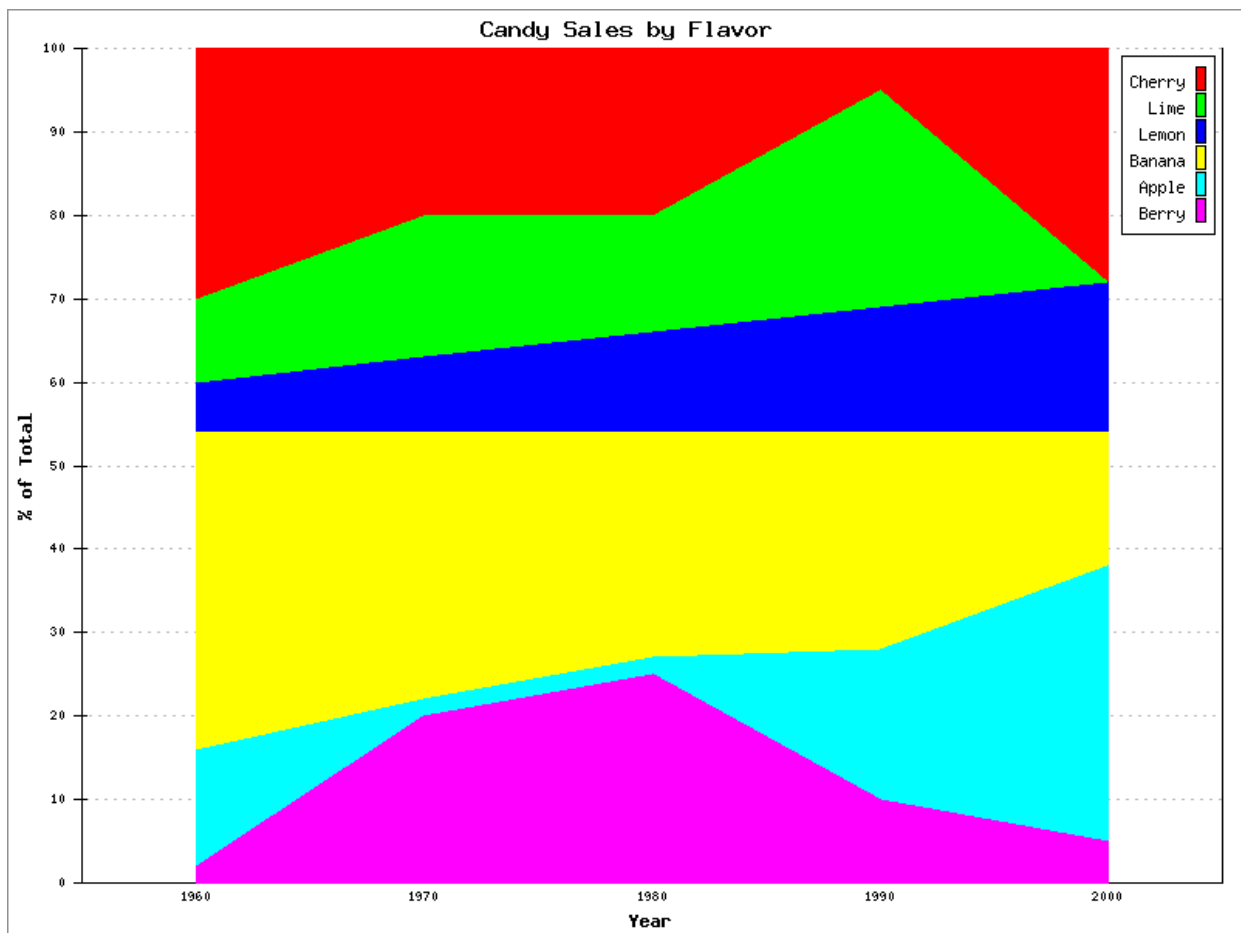

```php
<?php
# PHPlot Example: Area chart, 6 areas.
require_once 'phplot.php';

$data = array(
  array('1960', 100, 70, 60, 54, 16,  2),
  array('1970', 100, 80, 63, 54, 22, 20),
  array('1980', 100, 80, 66, 54, 27, 25),
  array('1990', 100, 95, 69, 54, 28, 10),
  array('2000', 100, 72, 72, 54, 38,  5),
);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');
```

```
$plot->SetPlotType('area');
$plot->SetDataType('text-data');
$plot->SetDataValues($data);

# Main plot title:
$plot->SetTitle('Candy Sales by Flavor');

# Set Y data limits, tick increment, and titles:
$plot->SetPlotAreaWorld(NULL, 0, NULL, 100);
$plot->SetYTickIncrement(10);
$plot->SetYTitle('% of Total');
$plot->SetXTitle('Year');

# Colors are significant to this data:
$plot->SetDataColors(array('red', 'green', 'blue', 'yellow', 'cyan', 'magenta'));
$plot->SetLegend(array('Cherry', 'Lime', 'Lemon', 'Banana', 'Apple', 'Berry'));

# Turn off X tick labels and ticks because they don't apply here:
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');

$plot->DrawGraph();
```

5.4. Example - Bar Chart

This is a bar chart with three data sets plotted. The data type is 'text-data', so the X values are implicit. But the X values are not relevant because the data labels (month names) are used instead, and the X tick marks and labels are turned off. This plot uses the default shading of bars.

Example 5.4. Bar Chart
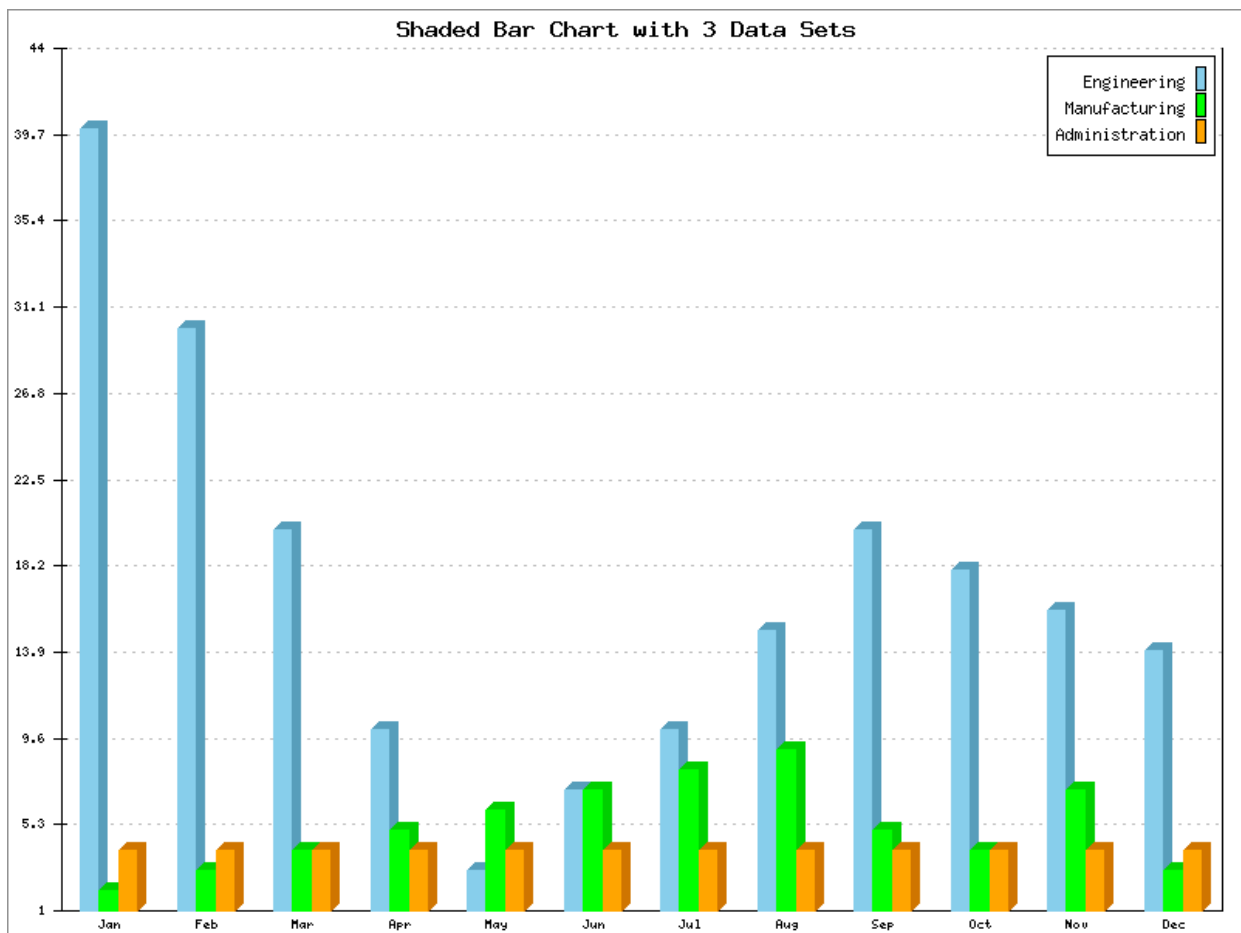

```php
<?php
# PHPlot Example: Bar chart, 3 data sets, shaded
require_once 'phplot.php';

$data = array(
  array('Jan', 40, 2, 4), array('Feb', 30, 3, 4), array('Mar', 20, 4, 4),
  array('Apr', 10, 5, 4), array('May',  3, 6, 4), array('Jun',  7, 7, 4),
  array('Jul', 10, 8, 4), array('Aug', 15, 9, 4), array('Sep', 20, 5, 4),
  array('Oct', 18, 4, 4), array('Nov', 16, 7, 4), array('Dec', 14, 3, 4),
);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');

$plot->SetPlotType('bars');
```

```
$plot->SetDataType('text-data');
$plot->SetDataValues($data);

# Main plot title:
$plot->SetTitle('Shaded Bar Chart with 3 Data Sets');

# Make a legend for the 3 data sets plotted:
$plot->SetLegend(array('Engineering', 'Manufacturing', 'Administration'));

# Turn off X tick labels and ticks because they don't apply here:
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');

$plot->DrawGraph();
```

5.5. Example - Unshaded Bar Chart

This is the same example as Example 5.4, "Bar Chart" except shading has been turned off. Instead we get flat rectangles with borders for the bars.

Example 5.5. Bar Chart - Unshaded
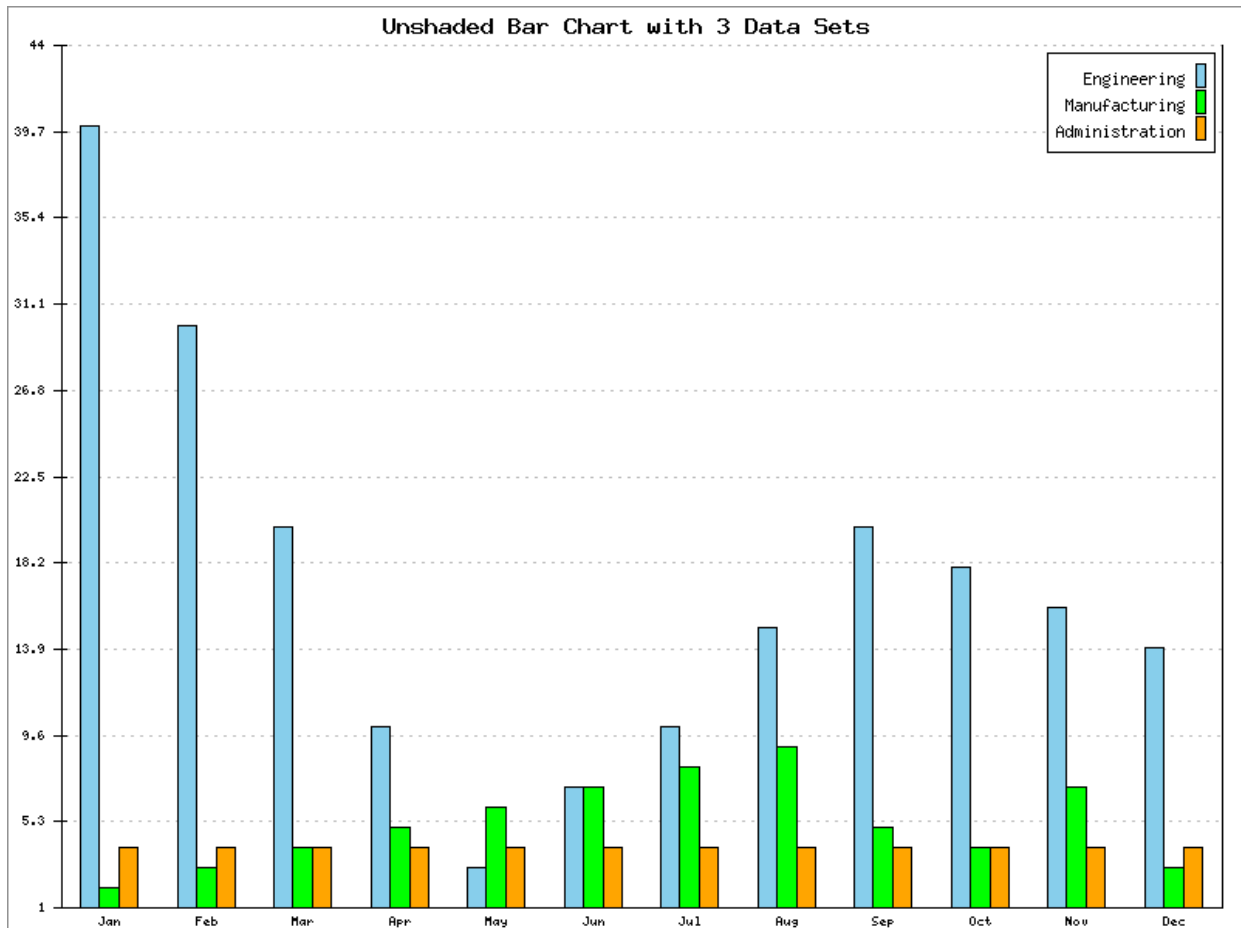

```php
<?php
# PHPlot Example: Bar chart, 3 data sets, unshaded
require_once 'phplot.php';

$data = array(
  array('Jan', 40, 2, 4), array('Feb', 30, 3, 4), array('Mar', 20, 4, 4),
  array('Apr', 10, 5, 4), array('May',  3, 6, 4), array('Jun',  7, 7, 4),
  array('Jul', 10, 8, 4), array('Aug', 15, 9, 4), array('Sep', 20, 5, 4),
  array('Oct', 18, 4, 4), array('Nov', 16, 7, 4), array('Dec', 14, 3, 4),
);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');

$plot->SetPlotType('bars');
$plot->SetDataType('text-data');
```

```
$plot->SetDataValues($data);

# Main plot title:
$plot->SetTitle('Unshaded Bar Chart with 3 Data Sets');

# No 3-D shading of the bars:
$plot->SetShading(0);

# Make a legend for the 3 data sets plotted:
$plot->SetLegend(array('Engineering', 'Manufacturing', 'Administration'));

# Turn off X tick labels and ticks because they don't apply here:
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');

$plot->DrawGraph();
```

5.6. Example - Bar Chart, Label Options

This is a bar chart showing data per year. Because the Y values are so large, we enable numeric formatting of Y tick labels, with precision 0. This results in a comma separator between thousands. This example also shows how to force the Y tick marks to start at zero and use a nice whole number for the tick interval.

Example 5.6. Bar Chart - Label Options
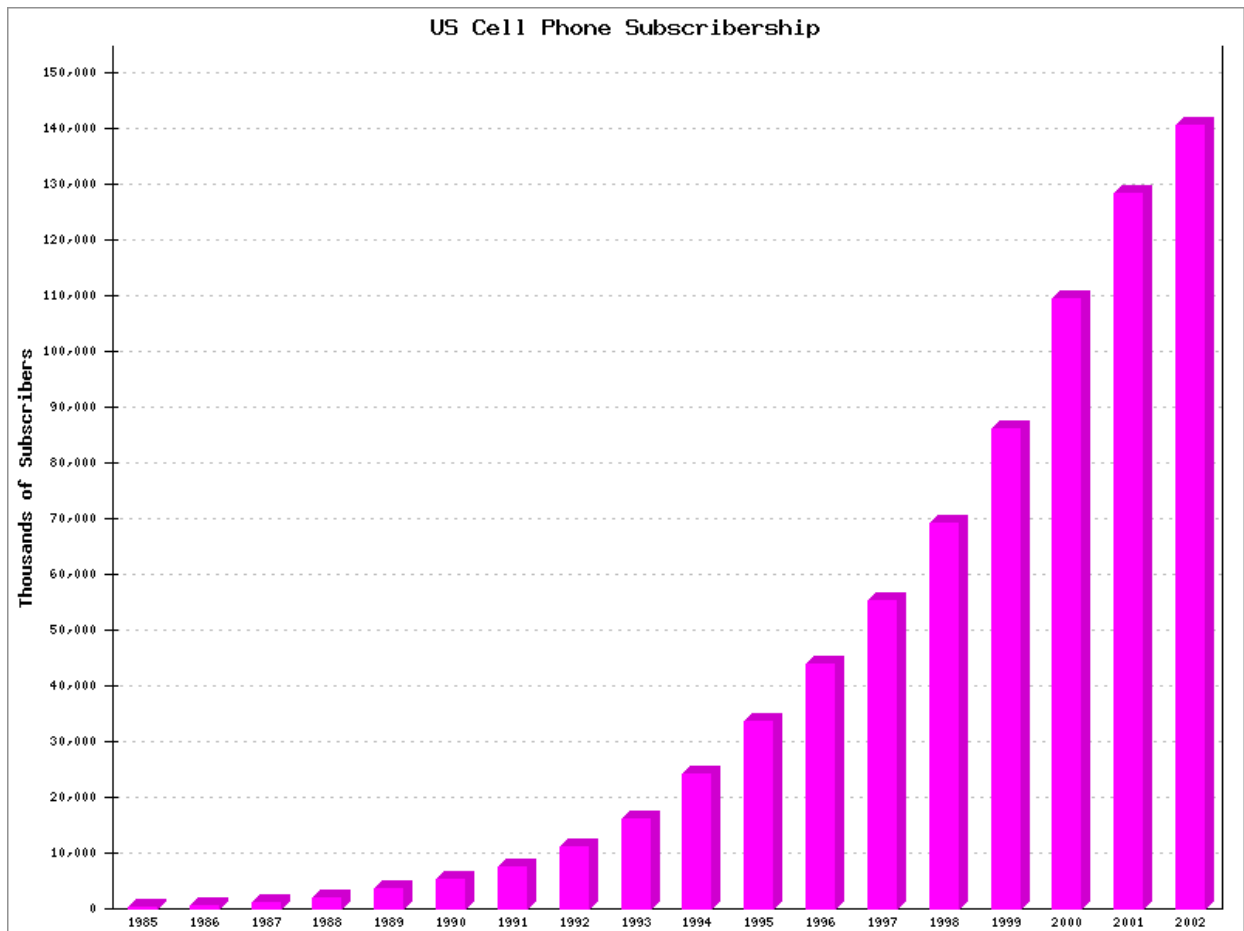

```php
<?php
# PHPlot Example: Bar chart, annual data
require_once 'phplot.php';

$data = array(
  array('1985', 340),    array('1986', 682),    array('1987', 1231),
  array('1988', 2069),   array('1989', 3509),   array('1990', 5283),
  array('1991', 7557),   array('1992', 11033),  array('1993', 16009),
  array('1994', 24134),  array('1995', 33768),  array('1996', 44043),
  array('1997', 55312),  array('1998', 69209),  array('1999', 86047),
  array('2000', 109478), array('2001', 128375), array('2002', 140767),
);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');
```

```
$plot->SetPlotType('bars');
$plot->SetDataType('text-data');
$plot->SetDataValues($data);

# Let's use a new color for these bars:
$plot->SetDataColors('magenta');

# Force bottom to Y=0 and set reasonable tick interval:
$plot->SetPlotAreaWorld(NULL, 0, NULL, NULL);
$plot->SetYTickIncrement(10000);
# Format the Y tick labels as numerics to get thousands separators:
$plot->SetYLabelType('data');
$plot->SetPrecisionY(0);

# Main plot title:
$plot->SetTitle('US Cell Phone Subscribership');
# Y Axis title:
$plot->SetYTitle('Thousands of Subscribers');

# Turn off X tick labels and ticks because they don't apply here:
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');

$plot->DrawGraph();
```

5.7. Example - Line/Point Plot, Point Shapes

This is a 'linepoints' plot (Lines/Points) showing all the point shapes available for 'point' and 'linepoints' plots. In this example, we also use a legend to display a text string for each data set, and change the data colors to get a different color for each point shape. Note that the point shape sizes have been increased to 10 in this example, to make them easier to identify.

Example 5.7. Line/Point Plot, Point Shapes
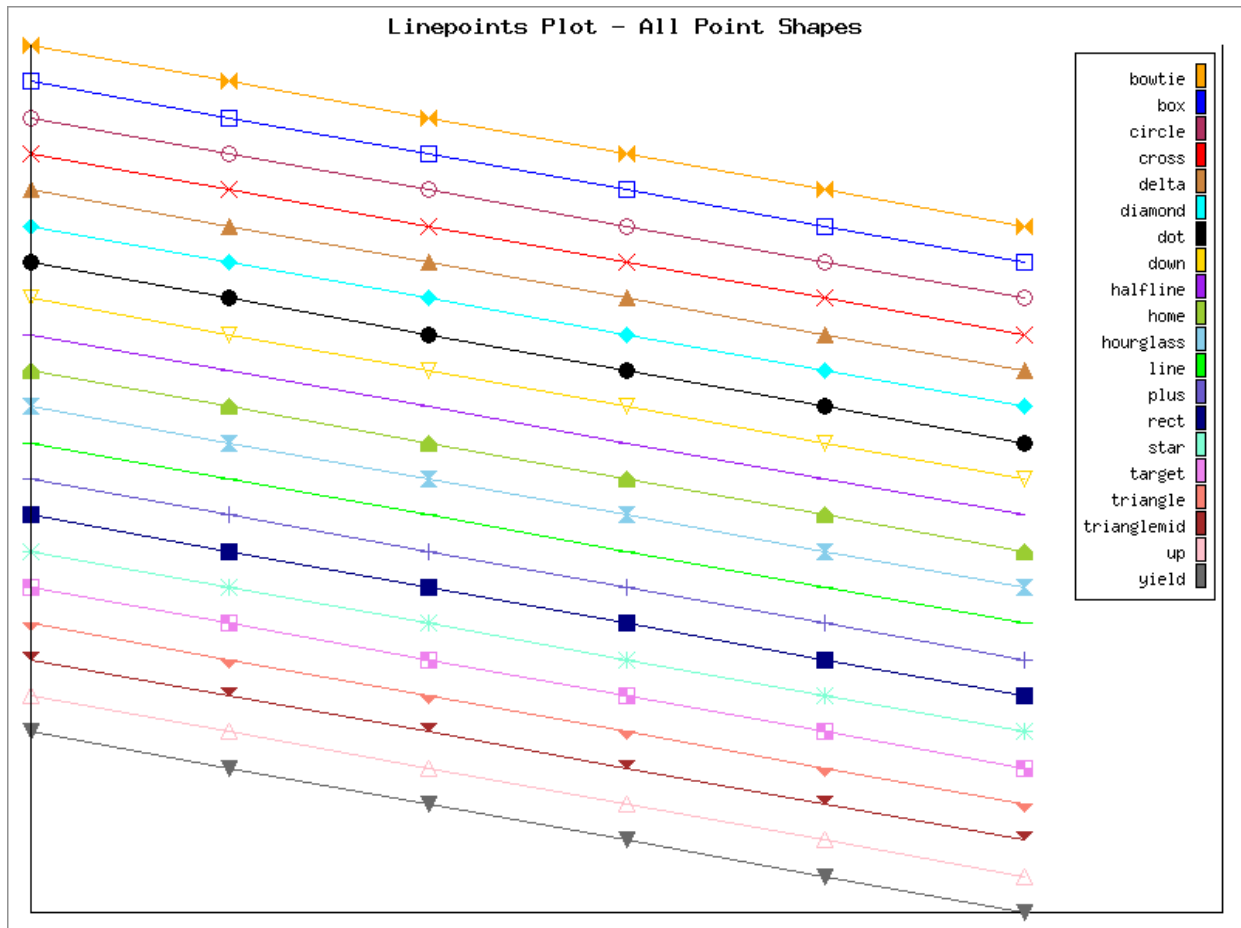

```
<?php
# PHPlot Example: Line-Point plot showing all the point shapes
require_once 'phplot.php';

# This array is used for both the point shapes and legend:
$shapes = array('bowtie', 'box', 'circle', 'cross', 'delta',
    'diamond', 'dot', 'down', 'halfline', 'home',
    'hourglass', 'line', 'plus', 'rect', 'star',
    'target', 'triangle', 'trianglemid', 'up', 'yield');

# Number of shapes defines the number of lines to draw:
$n_shapes = count($shapes);

# Make offset diagonal lines, one for each shape:
```

```
$ppl = 6; # Number of points per line.
$data = array();
for ($i = 0; $i < $ppl; $i++) {
    $subdata = array('', $i);
    $offset = $n_shapes + $ppl - $i - 2;
    for ($j = 0; $j < $n_shapes; $j++) $subdata[] = $offset - $j;
    $data[] = $subdata;
}

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');

$plot->SetPlotType('linepoints');
$plot->SetDataType('data-data');
$plot->SetDataValues($data);

# Main plot title:
$plot->SetTitle('Linepoints Plot - All Point Shapes');

# Increase X range to make room for the legend.
$plot->SetPlotAreaWorld(0, 0, $ppl, $n_shapes + $ppl - 2);
# Turn off tick labels and ticks - not used for this plot.
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');
$plot->SetYTickLabelPos('none');
$plot->SetYTickPos('none');

# Need some different colors;
$plot->SetDataColors(array('orange', 'blue', 'maroon', 'red', 'peru',
    'cyan', 'black', 'gold', 'purple', 'YellowGreen',
    'SkyBlue', 'green', 'SlateBlue', 'navy', 'aquamarine1',
    'violet', 'salmon', 'brown', 'pink', 'DimGrey'));

# Show all shapes:
$plot->SetPointShapes($shapes);

# Make the points bigger so we can see them:
$plot->SetPointSizes(10);

# Make the lines all be solid:
$plot->SetLineStyles('solid');

# Also show that as the legend:
$plot->SetLegend($shapes);

# Draw no grids:
$plot->SetDrawXGrid(False);
$plot->SetDrawYGrid(False);

$plot->DrawGraph();
```

5.8. Example - Pie Chart, text-data-single

This is a pie chart with the data array type 'text-data-single'. This data type is only used with pie charts. Each record in the data array simply contains a label (which is not used by PHPlot) and a slice size. In this example, we use the label to identify the data for our own reference, and then build a legend from those data labels along with the data values. This is useful because PHPlot only labels the slices with percentage values.

Example 5.8. Pie Chart, text-data-single
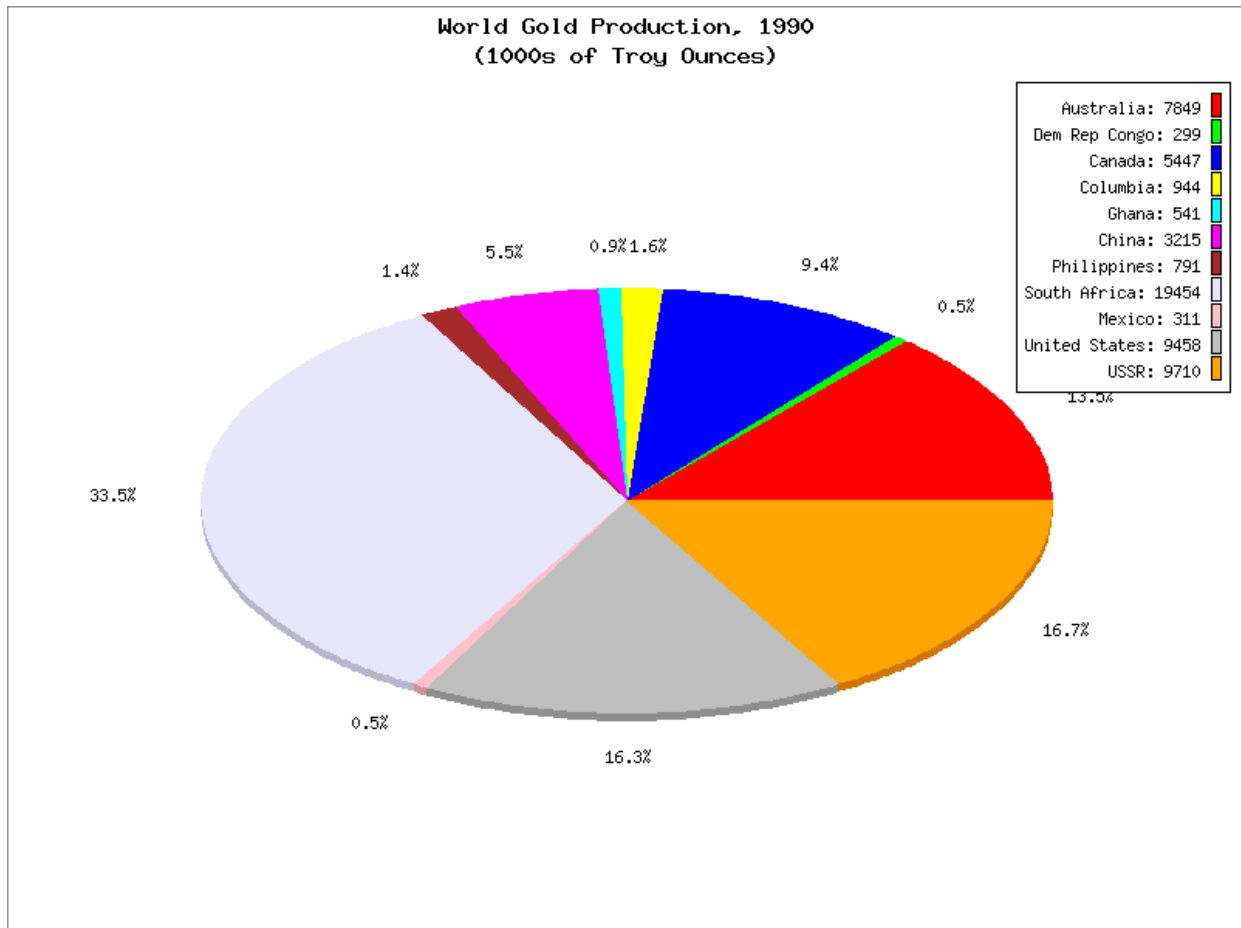

```php
<?php
# PHPlot Example: Pie/text-data-single
require_once 'phplot.php';

# The data labels aren't used directly by PHPlot. They are here for our
# reference, and we copy them to the legend below.
$data = array(
  array('Australia', 7849),
  array('Dem Rep Congo', 299),
  array('Canada', 5447),
  array('Columbia', 944),
  array('Ghana', 541),
  array('China', 3215),
  array('Philippines', 791),
```

```
  array('South Africa', 19454),
  array('Mexico', 311),
  array('United States', 9458),
  array('USSR', 9710),
);

$plot = new PHPlot(800,600);
$plot->SetImageBorderType('plain');

$plot->SetPlotType('pie');
$plot->SetDataType('text-data-single');
$plot->SetDataValues($data);

# Set enough different colors;
$plot->SetDataColors(array('red', 'green', 'blue', 'yellow', 'cyan',
                           'magenta', 'brown', 'lavender', 'pink',
                           'gray', 'orange'));

# Main plot title:
$plot->SetTitle("World Gold Production, 1990\n(1000s of Troy Ounces)");

# Build a legend from our data array.
# Each call to SetLegend makes one line as "label: value".
foreach ($data as $row)
  $plot->SetLegend(implode(': ', $row));

$plot->DrawGraph();
```

5.9. Example - Pie Chart, text-data

This is a simple pie chart showing the data type 'text-data'. When you use this data type with pie charts, the first entry in each record (a label) is ignored; the sum of all the second entries equals the relative size of the first slice, the sum of the third entries is the second slice, etc. So this pie has 4 slices, of relative size 250, 200, 350, and 200.

Data type 'data-data' is similar, except the first two entries in each record (label and X value for other plot types) is ignored.

Example 5.9. Pie Chart, text-data
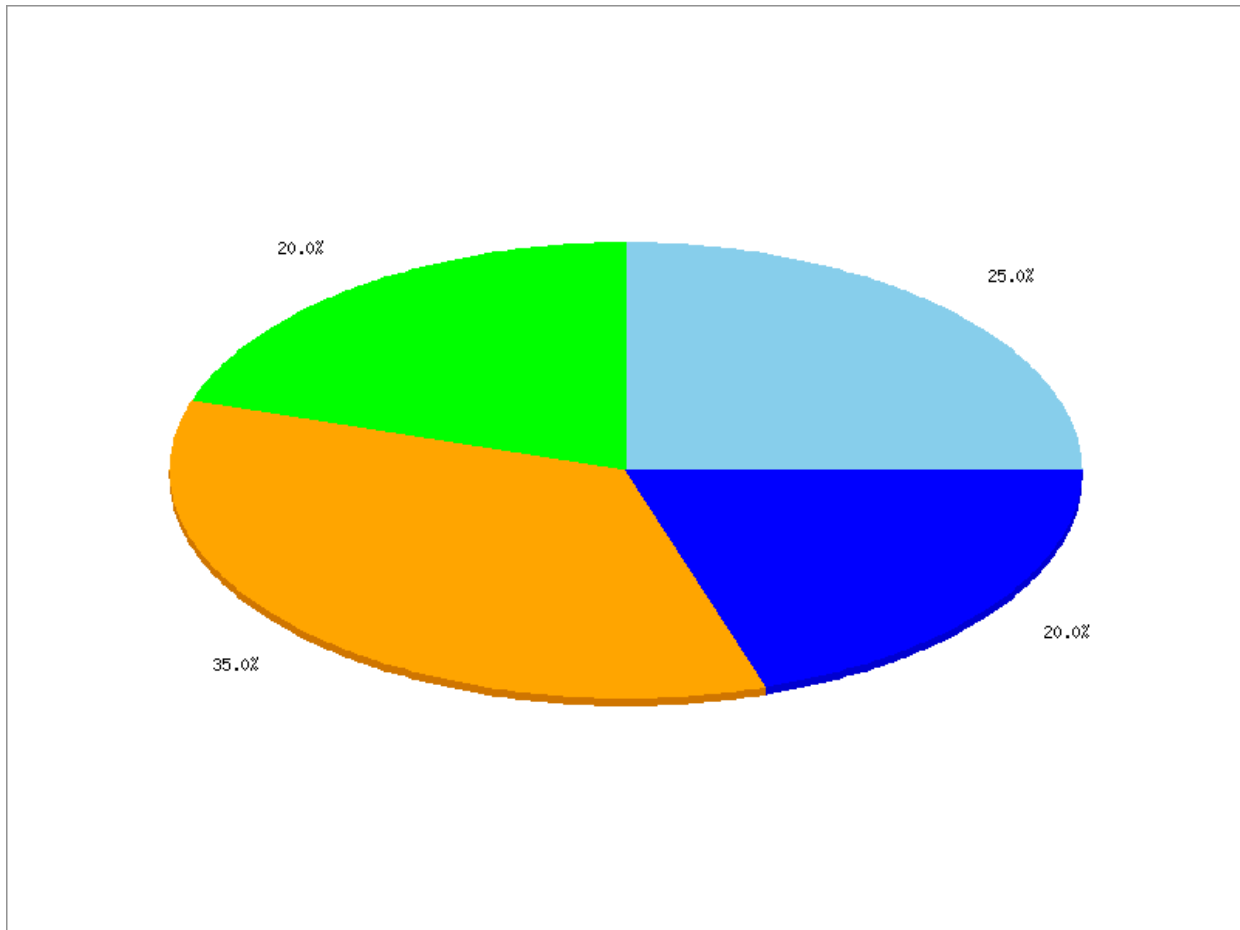

```php
<?php
# PHPlot Example: Pie/text-data
require_once 'phplot.php';

$data = array(
  array('', 100, 100, 200, 100),
  array('', 150, 100, 150, 100),
);

$plot = new PHPlot(800,600);
$plot->SetImageBorderType('plain');
$plot->SetDataType('text-data');
```

```
$plot->SetDataValues($data);
$plot->SetPlotType('pie');
$plot->DrawGraph();
```

5.10. Example - Pie Chart, flat with options

For this pie chart, we turned off shading with SetShading to get a flat pie instead of a 3-D look. We also overrode the default colors with our own array using SetDataColors, and used the same color names to make a legend with SetLegend. Finally, we moved the labels in towards the center with SetLabelScalePosition.

Example 5.10. Pie Chart, flat with options
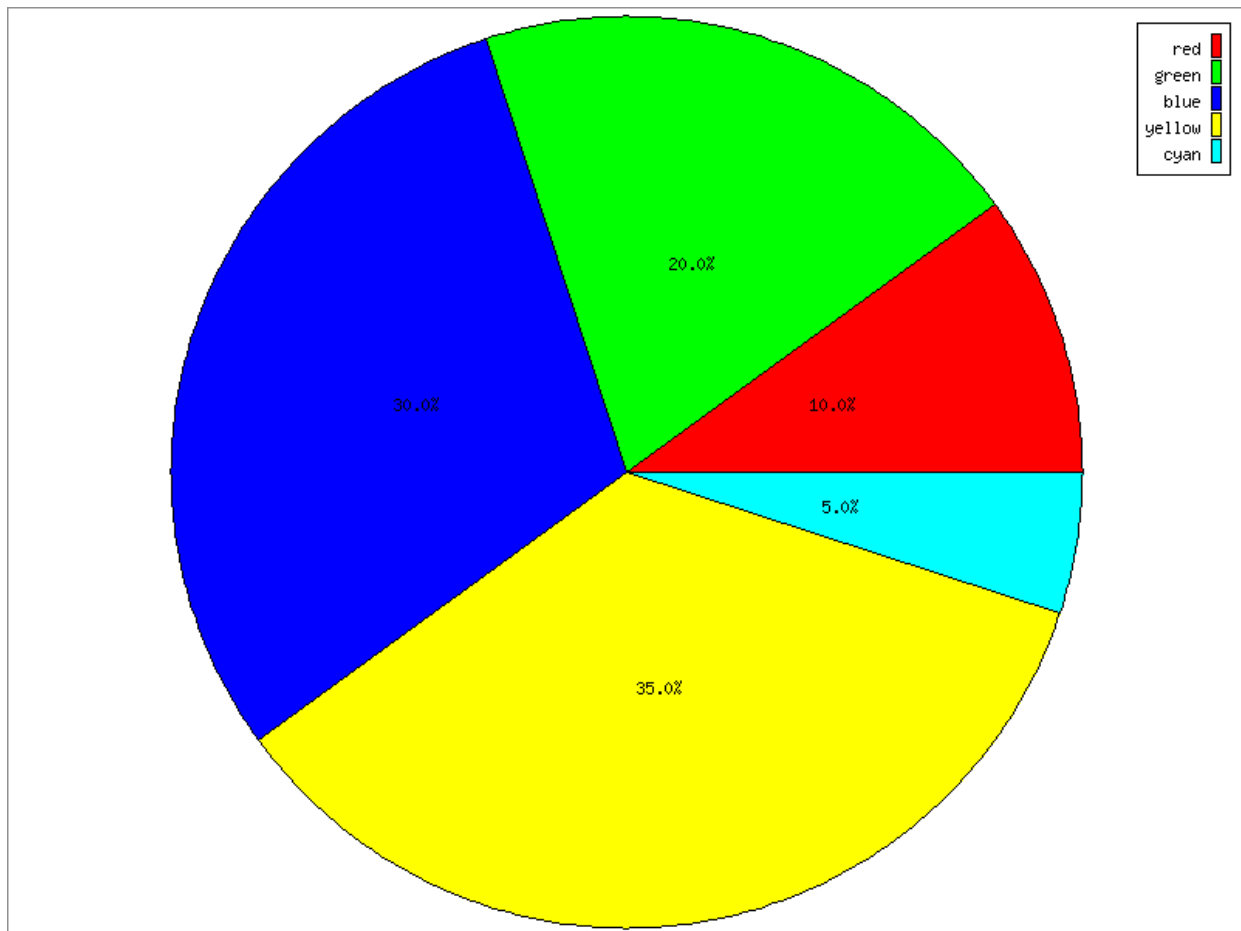

```php
<?php
# PHPlot Example:  Flat Pie with options
require_once 'phplot.php';

$data = array(
  array('', 10),
  array('', 20),
  array('', 30),
  array('', 35),
  array('',  5),
);

$plot = new PHPlot(800,600);
$plot->SetImageBorderType('plain');
$plot->SetDataType('text-data-single');
```

```
$plot->SetDataValues($data);
$plot->SetPlotType('pie');

$colors = array('red', 'green', 'blue', 'yellow', 'cyan');
$plot->SetDataColors($colors);
$plot->SetLegend($colors);
$plot->SetShading(0);
$plot->SetLabelScalePosition(0.2);

$plot->DrawGraph();
```

5.11. Example - Points Plot with Error Bars

This is a point plot with error bars (as indicated by data type 'data-data-error'). Each point is specified as X value, Y value, Y error in the positive direction, and Y error in the negative direction.

Example 5.11. Points Plot with Error Bars
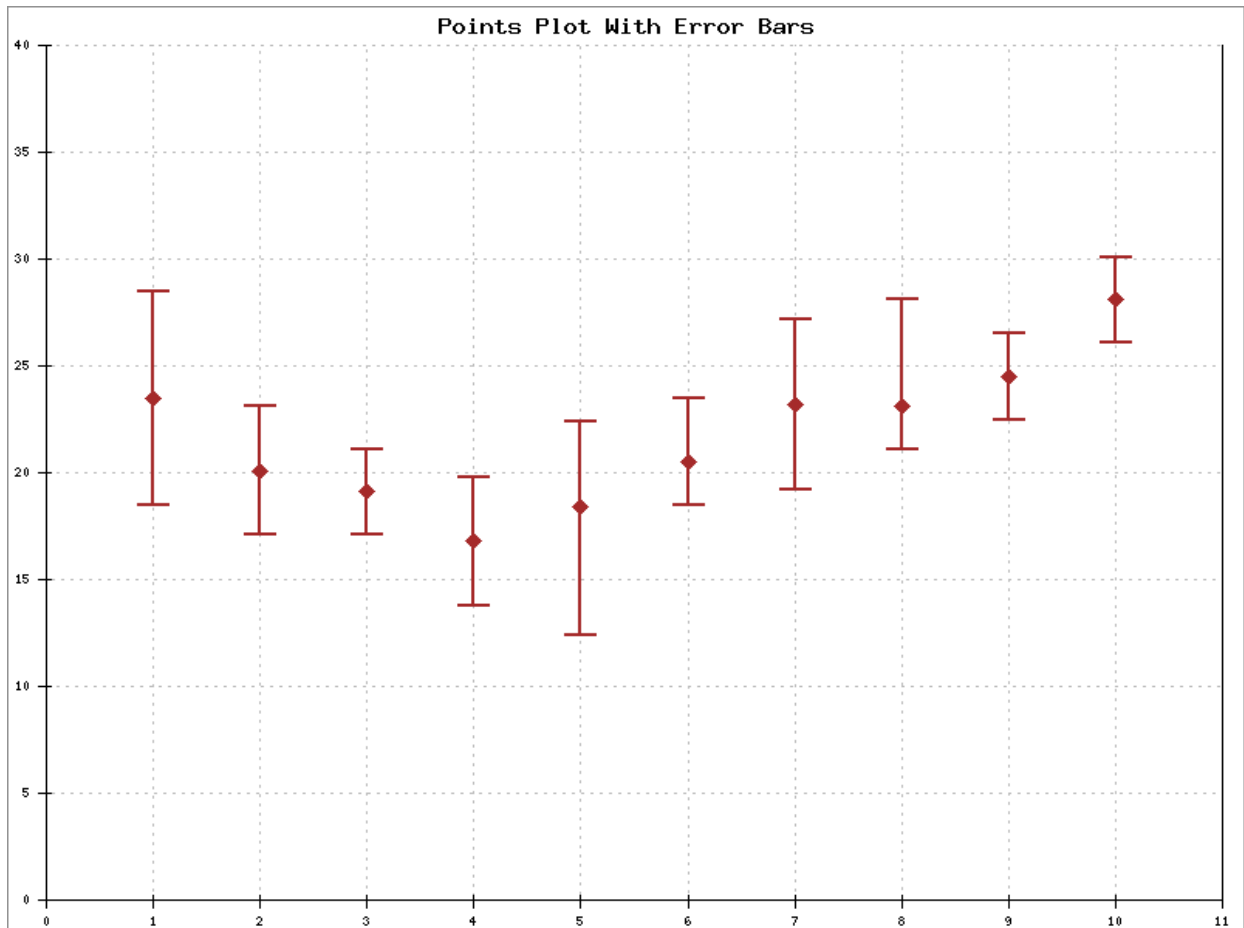

```
<?php
# PHPlot Example: Point chart with error bars
require_once 'phplot.php';

$data = array(
  array('', 1,  23.5, 5, 5), array('', 2,  20.1, 3, 3),
  array('', 3,  19.1, 2, 2), array('', 4,  16.8, 3, 3),
  array('', 5,  18.4, 4, 6), array('', 6,  20.5, 3, 2),
  array('', 7,  23.2, 4, 4), array('', 8,  23.1, 5, 2),
  array('', 9,  24.5, 2, 2), array('', 10, 28.1, 2, 2),
);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');

$plot->SetPlotType('points');
```

```
$plot->SetDataType('data-data-error');
$plot->SetDataValues($data);

# Main plot title:
$plot->SetTitle('Points Plot With Error Bars');

# Set data range and tick increments to get nice even numbers:
$plot->SetPlotAreaWorld(0, 0, 11, 40);
$plot->SetXTickIncrement(1);
$plot->SetYTickIncrement(5);

# Draw both grids:
$plot->SetDrawXGrid(True);
$plot->SetDrawYGrid(True);  # Is default

# Set some options for error bars:
$plot->SetErrorBarShape('tee');  # Is default
$plot->SetErrorBarSize(10);
$plot->SetErrorBarLineWidth(2);

# Use a darker color for the plot:
$plot->SetDataColors('brown');
$plot->SetErrorBarColors('brown');

# Make the points bigger so we can see them:
$plot->SetPointSizes(10);

$plot->DrawGraph();
```

5.12. Example - Points Plot / Scatterplot

This is a rather contrived example of using a 'points' plot to make a scatterplot. The data array is a set of X/Y points. With 'points' plots, the data can be in any order and duplicate X values are allowed. The points here are generated from R = 0.5 * Theta.

For this example, the X and Y axes and tick marks were moved to 0,0, labels turned off, and plot borders enabled for all four sides.

Example 5.12. Points Plot / Scatterplot
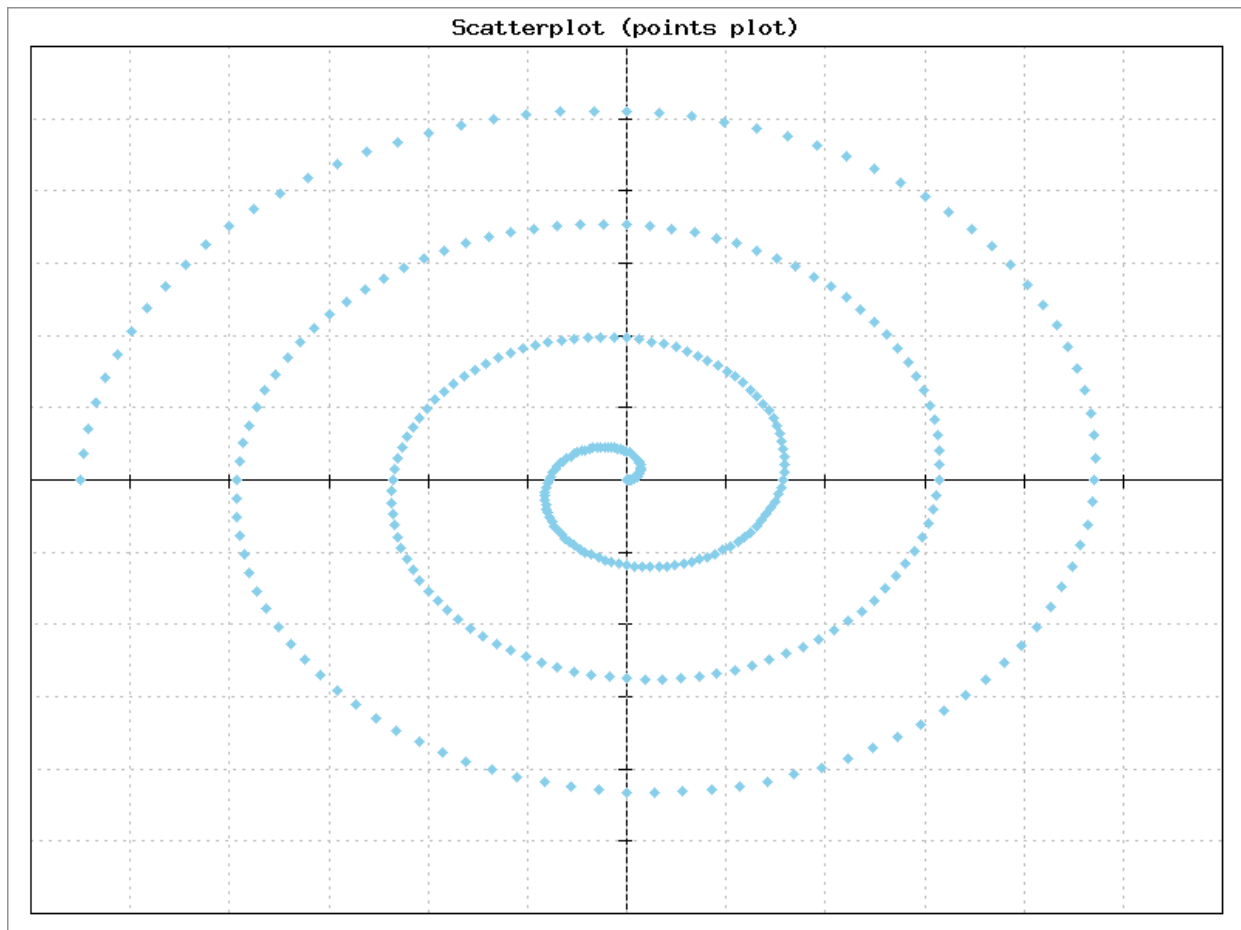

```
<?php
# PHPlot Example: Point plot - scatter plot
require_once 'phplot.php';

$data = array();
$a = 0.5;
$d_theta = M_PI/48.0;
for ($theta = M_PI * 7; $theta >= 0; $theta -= $d_theta)
  $data[] = array('', $a * $theta * cos($theta), $a * $theta * sin($theta));

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');
```

```
$plot->SetPlotType('points');
$plot->SetDataType('data-data');
$plot->SetDataValues($data);

# Main plot title:
$plot->SetTitle('Scatterplot (points plot)');

# Need to set area and ticks to get reasonable choices.
$plot->SetPlotAreaWorld(-12, -12, 12, 12);
$plot->SetXTickIncrement(2);
$plot->SetYTickIncrement(2);

# Move axes and ticks to 0,0, but turn off tick labels:
$plot->SetXAxisPosition(0); # Is default
$plot->SetYAxisPosition(0);
$plot->SetXTickPos('xaxis');
$plot->SetXTickLabelPos('none');
$plot->SetYTickPos('yaxis');
$plot->SetYTickLabelPos('none');

# Turn on 4 sided borders, now that axes are inside:
$plot->SetPlotBorderType('full');

# Draw both grids:
$plot->SetDrawXGrid(True);
$plot->SetDrawYGrid(True);  # Is default

$plot->DrawGraph();
```

5.13. Example - Squared Plot

This is a squared line plot, which is similar to a line plot but the points are connected with steps.

Example 5.13. Squared Plot
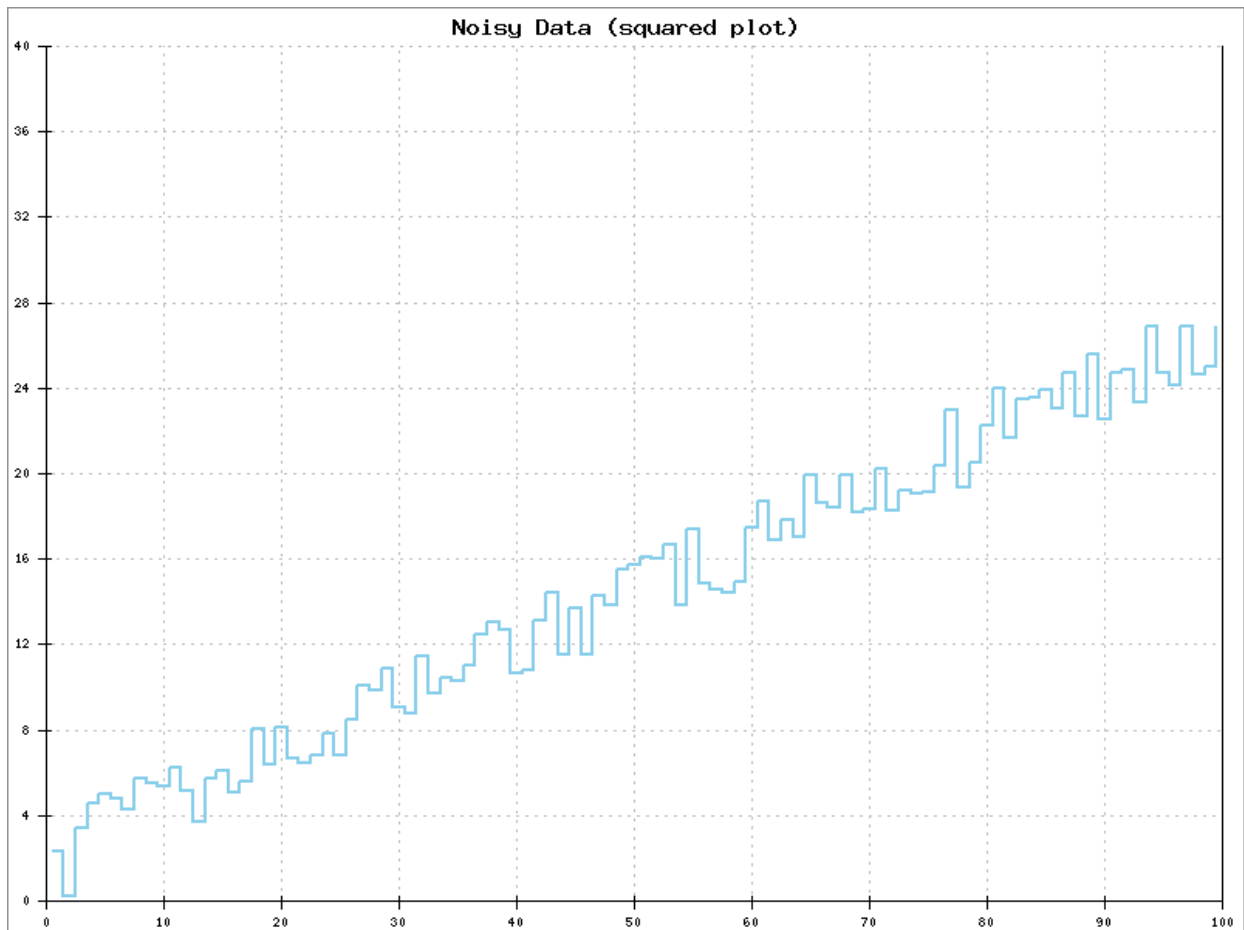

```php
<?php
# PHPlot Example: squared plot
require_once 'phplot.php';

# To get repeatable results with 'random' data:
mt_srand(1);

# Make some noisy data:
$data = array();
for ($i = 0; $i < 100; $i++)
  $data[] = array('', $i / 4.0 + 2.0 + mt_rand(-20, 20) / 10.0);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');

$plot->SetPlotType('squared');
$plot->SetDataType('text-data');
```

```
$plot->SetDataValues($data);

$plot->SetTitle('Noisy Data (squared plot)');

# Make the lines a bit wider:
$plot->SetLineWidths(2);

# Turn on the X grid (Y grid is on by default):
$plot->SetDrawXGrid(True);

# Use exactly this data range:
$plot->SetPlotAreaWorld(0, 0, 100, 40);

$plot->DrawGraph();
```

5.14. Example - Stacked Bars, Shaded

This is a stacked bar chart, with the default 3-D shaded look. (Compare with Example 5.15, "Stacked Bars, Unshaded" which has no shading.) The only valid data types for stacked bars are 'text-data' (for vertical plots), and 'text-data-yx' (for horizontal plots).

Example 5.14. Stacked Bars, Shaded
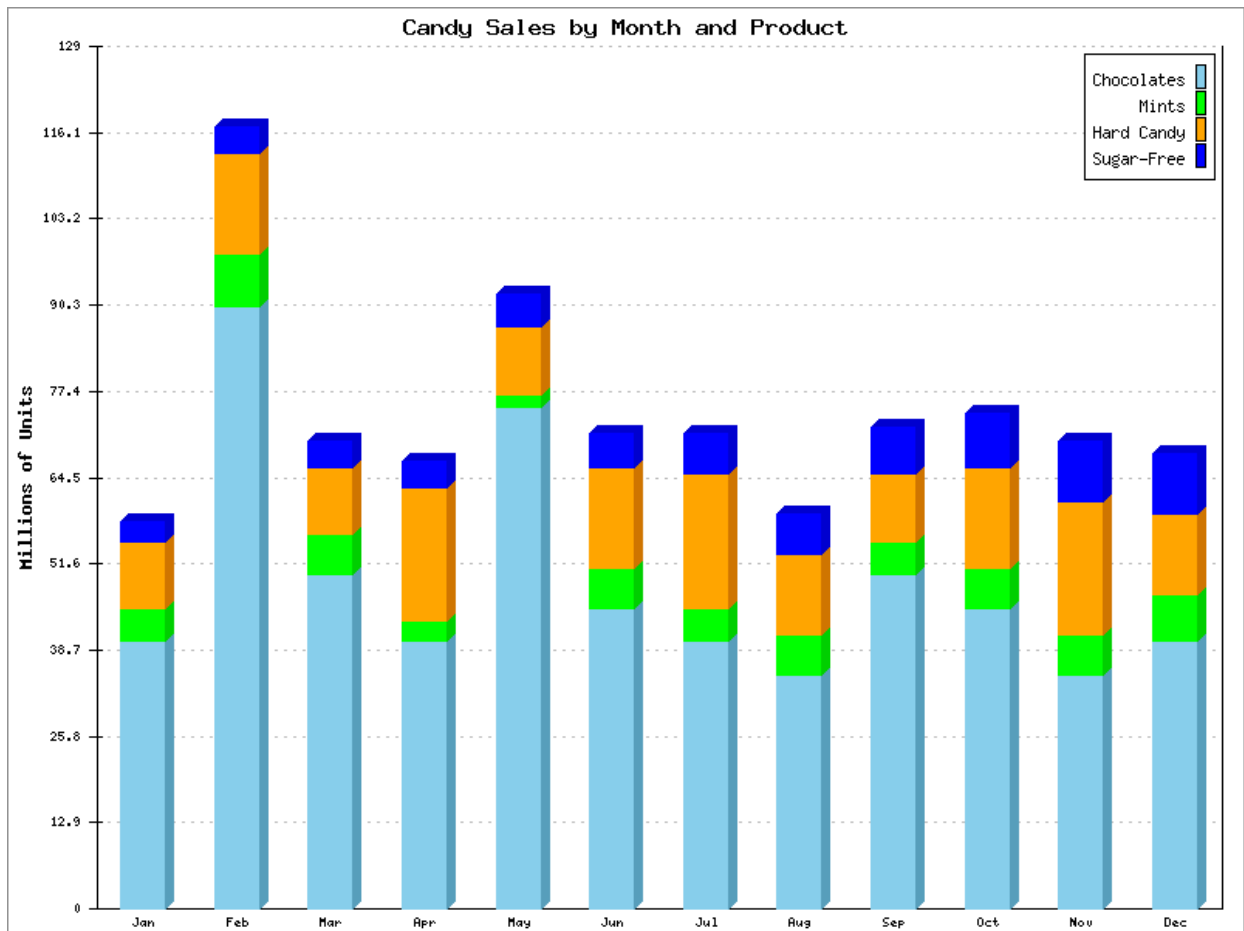

```php
<?php
# PHPlot Example: Stacked Bars, shaded
require_once 'phplot.php';

$data = array(
  array('Jan', 40, 5, 10, 3), array('Feb', 90, 8, 15, 4),
  array('Mar', 50, 6, 10, 4), array('Apr', 40, 3, 20, 4),
  array('May', 75, 2, 10, 5), array('Jun', 45, 6, 15, 5),
  array('Jul', 40, 5, 20, 6), array('Aug', 35, 6, 12, 6),
  array('Sep', 50, 5, 10, 7), array('Oct', 45, 6, 15, 8),
  array('Nov', 35, 6, 20, 9), array('Dec', 40, 7, 12, 9),
);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');
```

```
$plot->SetPlotType('stackedbars');
$plot->SetDataType('text-data');
$plot->SetDataValues($data);

$plot->SetTitle('Candy Sales by Month and Product');
$plot->SetYTitle('Millions of Units');
$plot->SetLegend(array('Chocolates', 'Mints', 'Hard Candy', 'Sugar-Free'));

$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');

$plot->DrawGraph();
```

5.15. Example - Stacked Bars, Unshaded

This is the same as Example 5.14, "Stacked Bars, Shaded" except we have turned off shading and now get flat stacked bars with borders.

Example 5.15. Stacked Bars, Unshaded
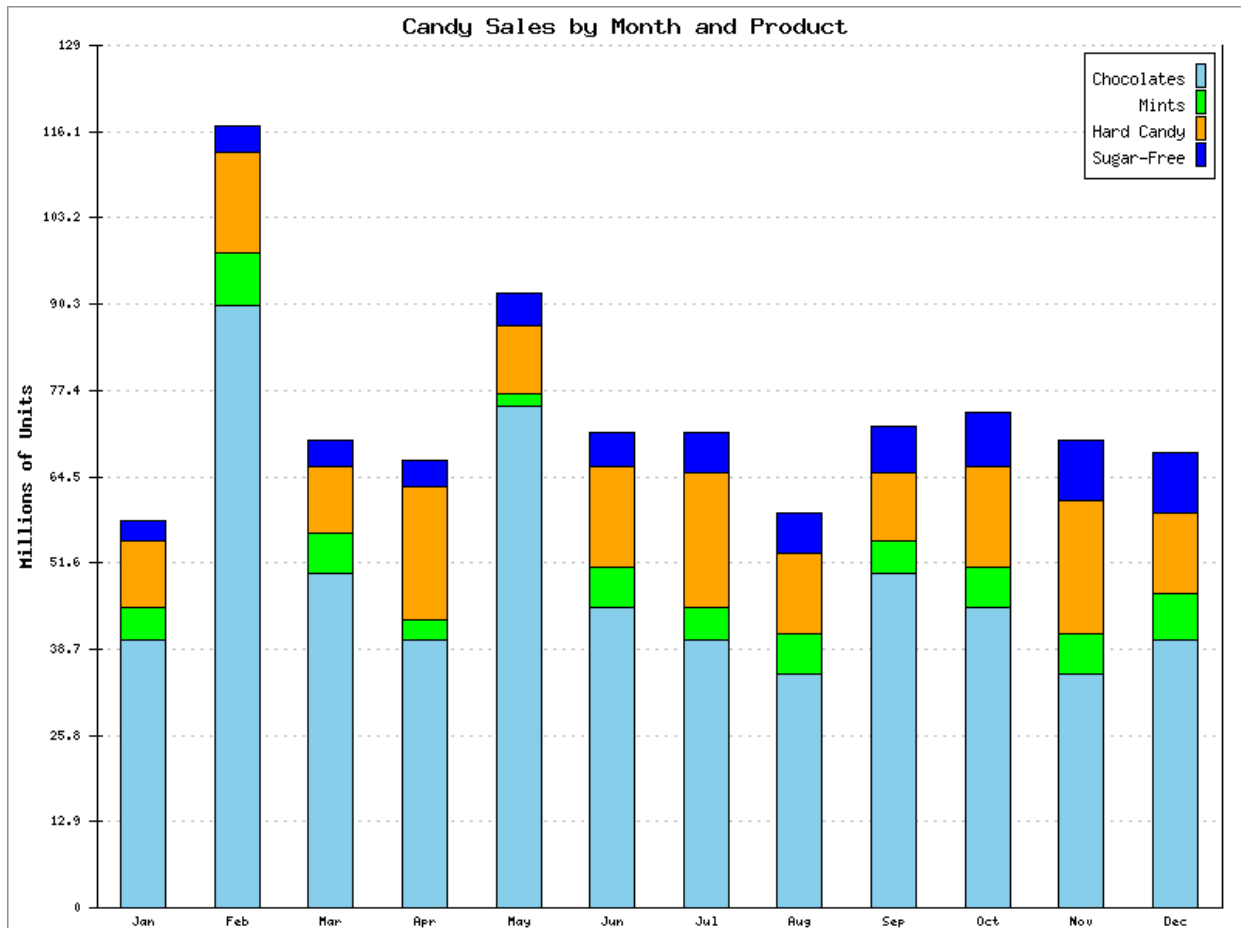

```
<?php
# PHPlot Example: Stacked Bars, unshaded
require_once 'phplot.php';

$data = array(
  array('Jan', 40, 5, 10, 3), array('Feb', 90, 8, 15, 4),
  array('Mar', 50, 6, 10, 4), array('Apr', 40, 3, 20, 4),
  array('May', 75, 2, 10, 5), array('Jun', 45, 6, 15, 5),
  array('Jul', 40, 5, 20, 6), array('Aug', 35, 6, 12, 6),
  array('Sep', 50, 5, 10, 7), array('Oct', 45, 6, 15, 8),
  array('Nov', 35, 6, 20, 9), array('Dec', 40, 7, 12, 9),
);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');
```

```
$plot->SetPlotType('stackedbars');
$plot->SetDataType('text-data');
$plot->SetDataValues($data);

$plot->SetTitle('Candy Sales by Month and Product');
$plot->SetYTitle('Millions of Units');

# No shading:
$plot->SetShading(0);

$plot->SetLegend(array('Chocolates', 'Mints', 'Hard Candy', 'Sugar-Free'));

$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');

$plot->DrawGraph();
```

5.16. Example - Thin Bar Line Plot

This is a Thin Bar Line Plot (thinbarline).

Example 5.16. Thin Bar Line
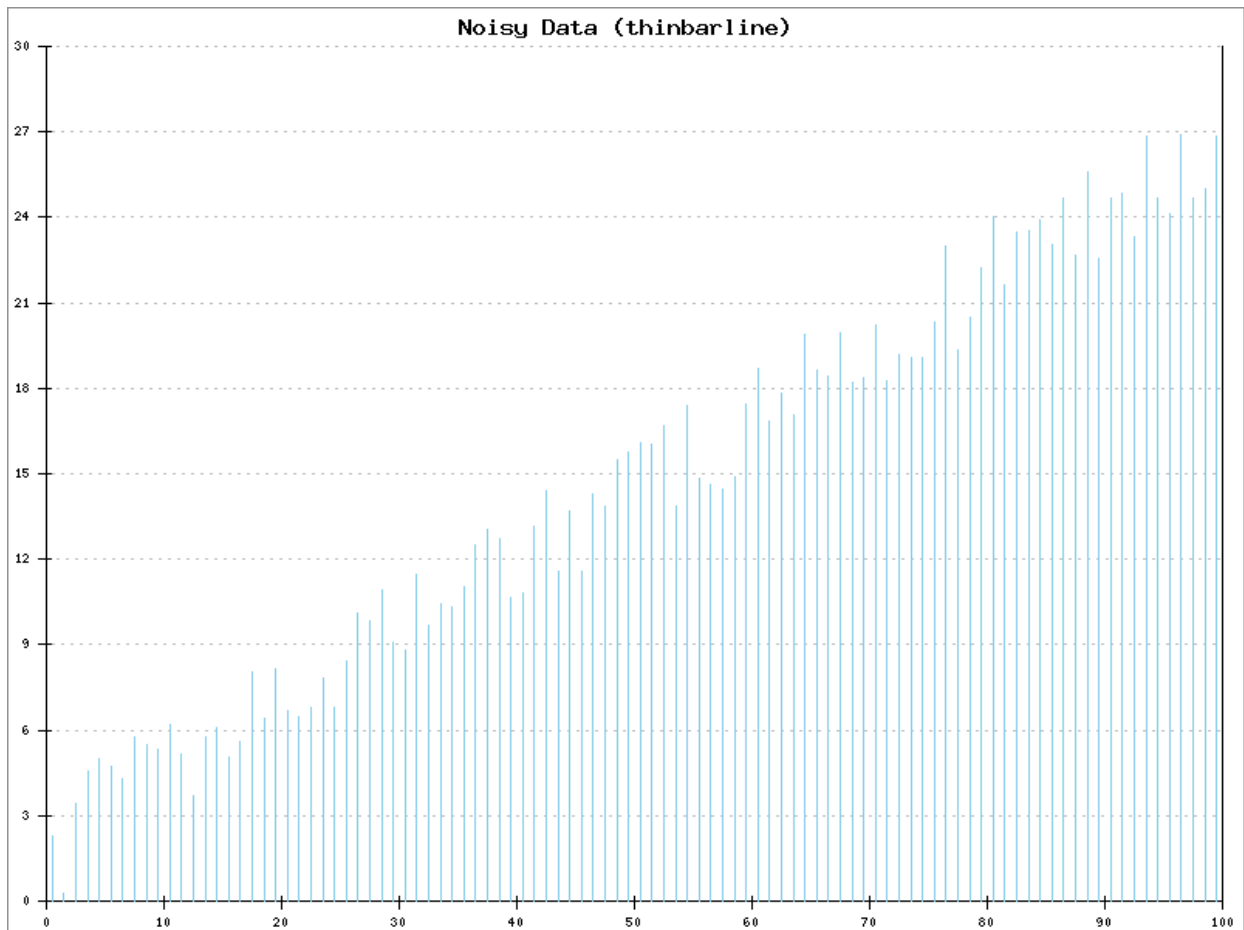

```php
<?php
# PHPlot Example: thinbarline plot
require_once 'phplot.php';

# To get repeatable results with 'random' data:
mt_srand(1);

# Make some noisy data:
$data = array();
for ($i = 0; $i < 100; $i++)
  $data[] = array('', $i / 4.0 + 2.0 + mt_rand(-20, 20) / 10.0);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');

$plot->SetPlotType('thinbarline');
$plot->SetDataType('text-data');
```

```
$plot->SetDataValues($data);

# Main plot title:
$plot->SetTitle('Noisy Data (thinbarline)');

$plot->DrawGraph();
```

5.17. Example - Thin Bar Line Plot, Wider Lines

This is the same as the previous plot type (Example 5.16, "Thin Bar Line") except the lines are wider. This now looks more like a bar chart.

Example 5.17. Thin Bar Line Plot, Wider Lines
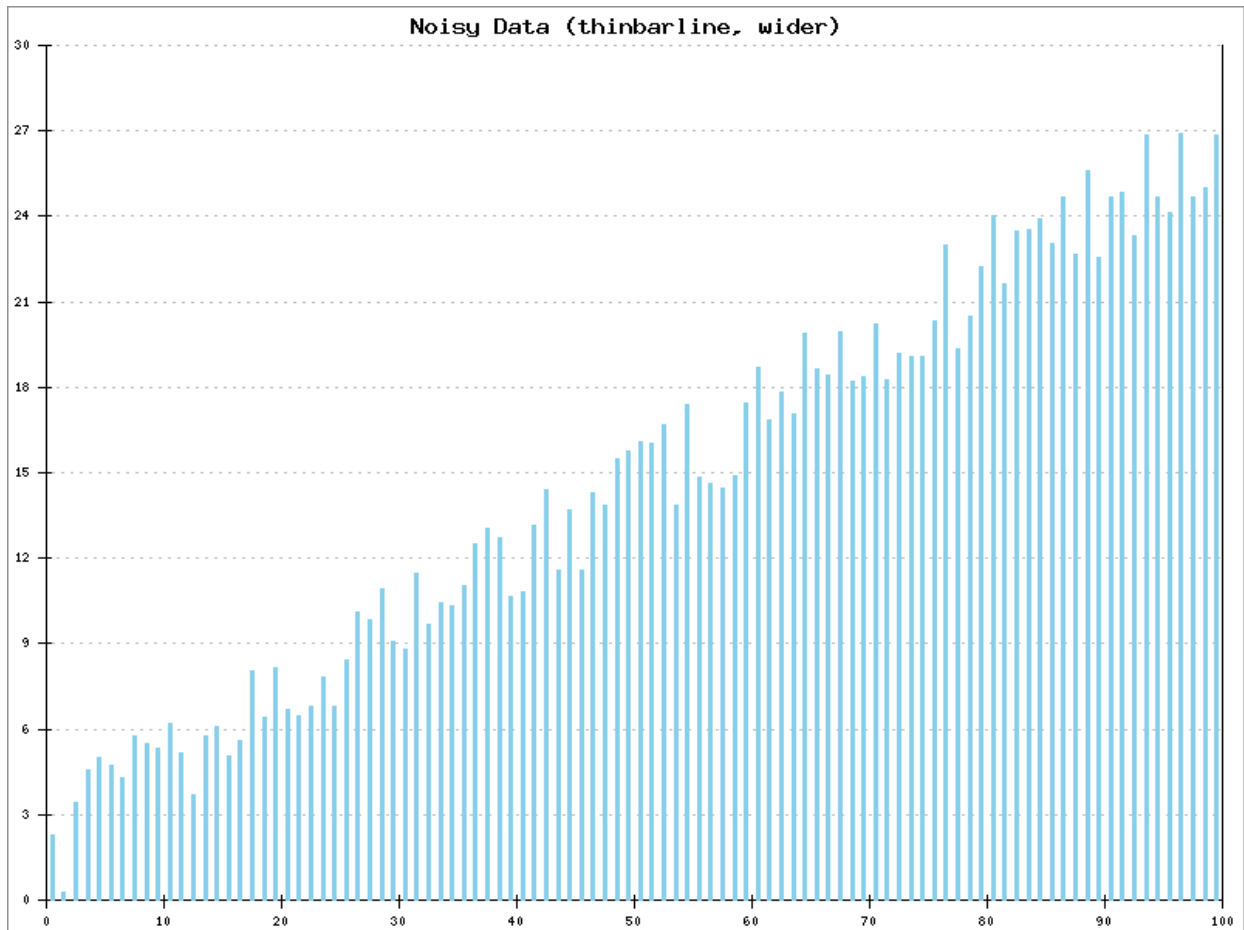

```php
<?php
# PHPlot Example: thinbarline plot, wider
require_once 'phplot.php';

# To get repeatable results with 'random' data:
mt_srand(1);

# Make some noisy data:
$data = array();
for ($i = 0; $i < 100; $i++)
  $data[] = array('', $i / 4.0 + 2.0 + mt_rand(-20, 20) / 10.0);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');

$plot->SetPlotType('thinbarline');
```

```
$plot->SetDataType('text-data');
$plot->SetDataValues($data);

# Main plot title:
$plot->SetTitle('Noisy Data (thinbarline, wider)');

# Make the lines wider:
$plot->SetLineWidths(3);

$plot->DrawGraph();
```

5.18. Example - Two Plots on One Image

This example shows multiple plots tiled on a single image. To place multiple plots on an image, first disable automatic output with SetPrintImage(False). Then define each plot area using SetPlotAreaPixels and create the plot. Finish each plot with DrawGraph. At the end, PrintImage outputs the image containing all of the plots.

See Section 4.6, "Multiple Plots Per Image" for more information.

Example 5.18. Two Plots on One Image
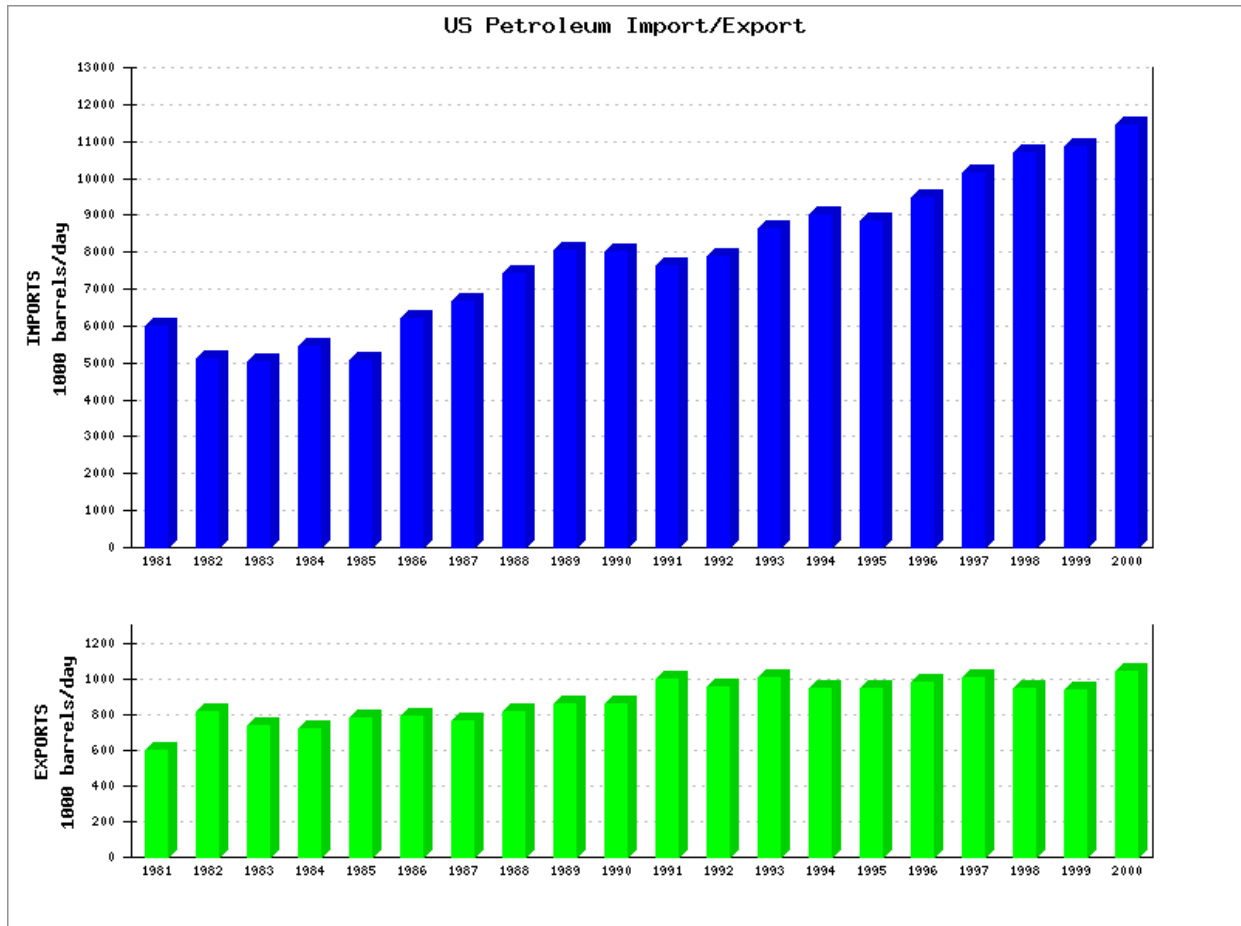

```php
<?php
# PHPlot Example: Two plots on one image
require_once 'phplot.php';

$data1 = array(         # Data array for top plot: Imports
  array('1981', 5996),  array('1982', 5113),  array('1983', 5051),
  array('1984', 5437),  array('1985', 5067),  array('1986', 6224),
  array('1987', 6678),  array('1988', 7402),  array('1989', 8061),
  array('1990', 8018),  array('1991', 7627),  array('1992', 7888),
  array('1993', 8620),  array('1994', 8996),  array('1995', 8835),
  array('1996', 9478),  array('1997', 10162), array('1998', 10708),
  array('1999', 10852), array('2000', 11459),
);
```

```
$data2 = array(          # Data array for bottom plot: Exports
  array('1981', 595),  array('1982', 815),  array('1983', 739),
  array('1984', 722),  array('1985', 781),  array('1986', 785),
  array('1987', 764),  array('1988', 815),  array('1989', 859),
  array('1990', 857),  array('1991', 1001), array('1992', 950),
  array('1993', 1003), array('1994', 942),  array('1995', 949),
  array('1996', 981),  array('1997', 1003), array('1998', 945),
  array('1999', 940),  array('2000', 1040),
);

$plot = new PHPlot(800,600);
$plot->SetImageBorderType('plain');

# Disable auto-output:
$plot->SetPrintImage(0);

# There is only one title: it is outside both plot areas.
$plot->SetTitle('US Petroleum Import/Export');

# Set up area for first plot:
$plot->SetPlotAreaPixels(80, 40, 740, 350);

# Do the first plot:
$plot->SetDataType('text-data');
$plot->SetDataValues($data1);
$plot->SetPlotAreaWorld(NULL, 0, NULL, 13000);
$plot->SetDataColors(array('blue'));
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');
$plot->SetYTickIncrement(1000);
$plot->SetYTitle("IMPORTS\n1000 barrels/day");

$plot->SetPlotType('bars');
$plot->DrawGraph();

# Set up area for second plot:
$plot->SetPlotAreaPixels(80, 400, 740, 550);

# Do the second plot:
$plot->SetDataType('text-data');
$plot->SetDataValues($data2);
$plot->SetPlotAreaWorld(NULL, 0, NULL, 1300);
$plot->SetDataColors(array('green'));
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');
$plot->SetYTickIncrement(200);
$plot->SetYTitle("EXPORTS\n1000 barrels/day");

$plot->SetPlotType('bars');
$plot->DrawGraph();

# Output the image now:
$plot->PrintImage();
```

5.19. Example - Bar Chart with Data Value Labels

This is a bar chart with data value labels. Data value labels can be used as an alternative to (or along with) Y tick labels, but only with bar and stackedbar charts. (Bar chart data value labels were added to PHPlot-5.0rc3.)

Example 5.19. Bar Chart with Data Value Labels
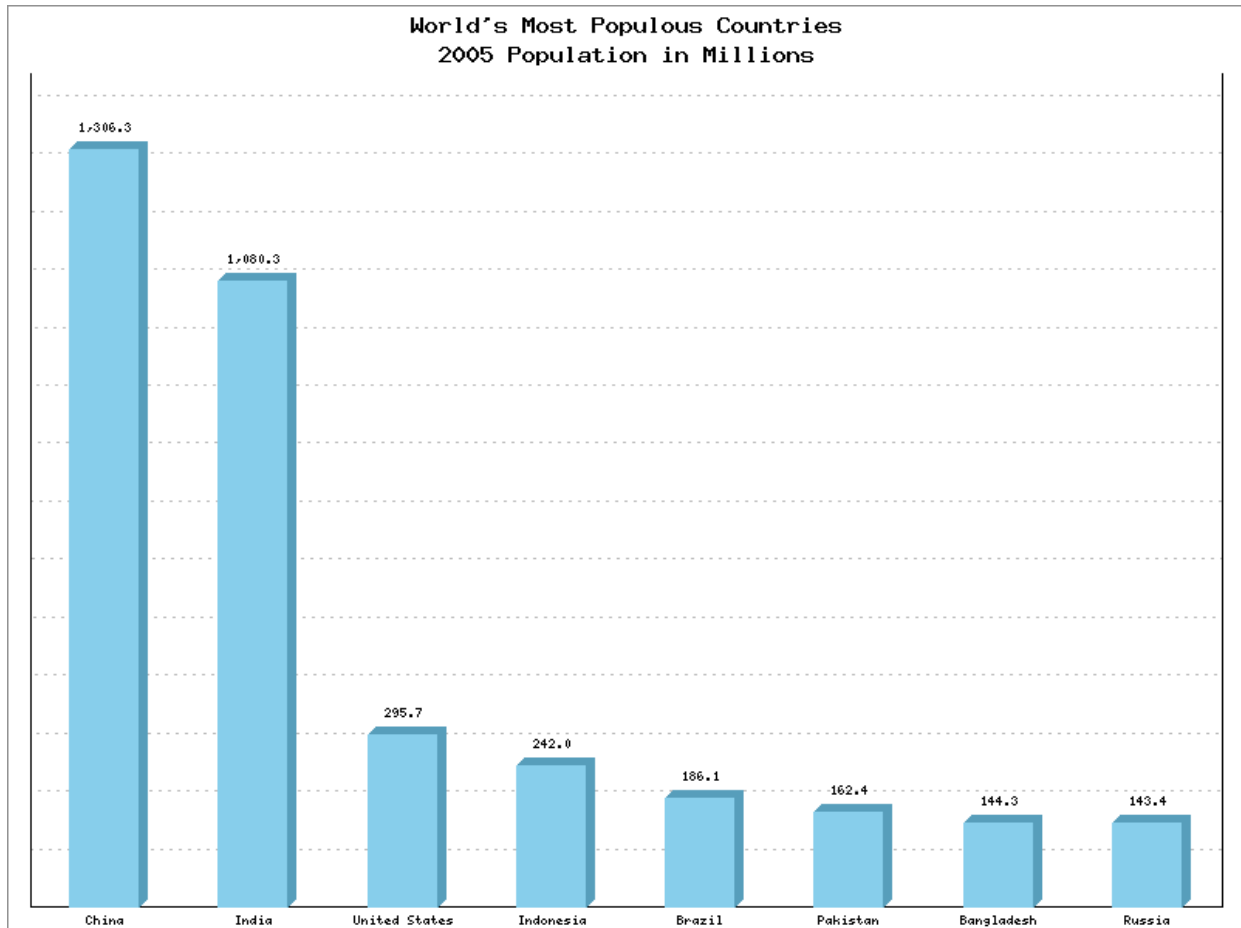

```php
<?php
# PHPlot Example: Bar chart, with data labels
require_once 'phplot.php';

$data = array(
  array('China', 1306.31),            array('India', 1080.26),
  array('United States',  295.73),    array('Indonesia', 241.97),
  array('Brazil', 186.11),            array('Pakistan', 162.42),
  array('Bangladesh', 144.32),        array('Russia', 143.42),
);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');
$plot->SetPlotType('bars');
```

```
$plot->SetDataType('text-data');
$plot->SetDataValues($data);
$plot->SetTitle("World's Most Populous Countries\n2005 Population in Millions");

# Turn off X tick labels and ticks because they don't apply here:
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');

# Make sure Y=0 is displayed:
$plot->SetPlotAreaWorld(NULL, 0);
# Y Tick marks are off, but Y Tick Increment also controls the Y grid lines:
$plot->SetYTickIncrement(100);

# Turn on Y data labels:
$plot->SetYDataLabelPos('plotin');

# With Y data labels, we don't need Y ticks or their labels, so turn them off.
$plot->SetYTickLabelPos('none');
$plot->SetYTickPos('none');

# Format the Y Data Labels as numbers with 1 decimal place.
# Note that this automatically calls SetYLabelType('data').
$plot->SetPrecisionY(1);

$plot->DrawGraph();
```

5.20. Example - Stacked Bars with Y Data Value Labels

This is the same as Example 5.15, "Stacked Bars, Unshaded" except that Y data value labels are turned on. Note this feature was added in PHPlot-5.1.1.

Example 5.20. Stacked Bars with Y Data Value Labels
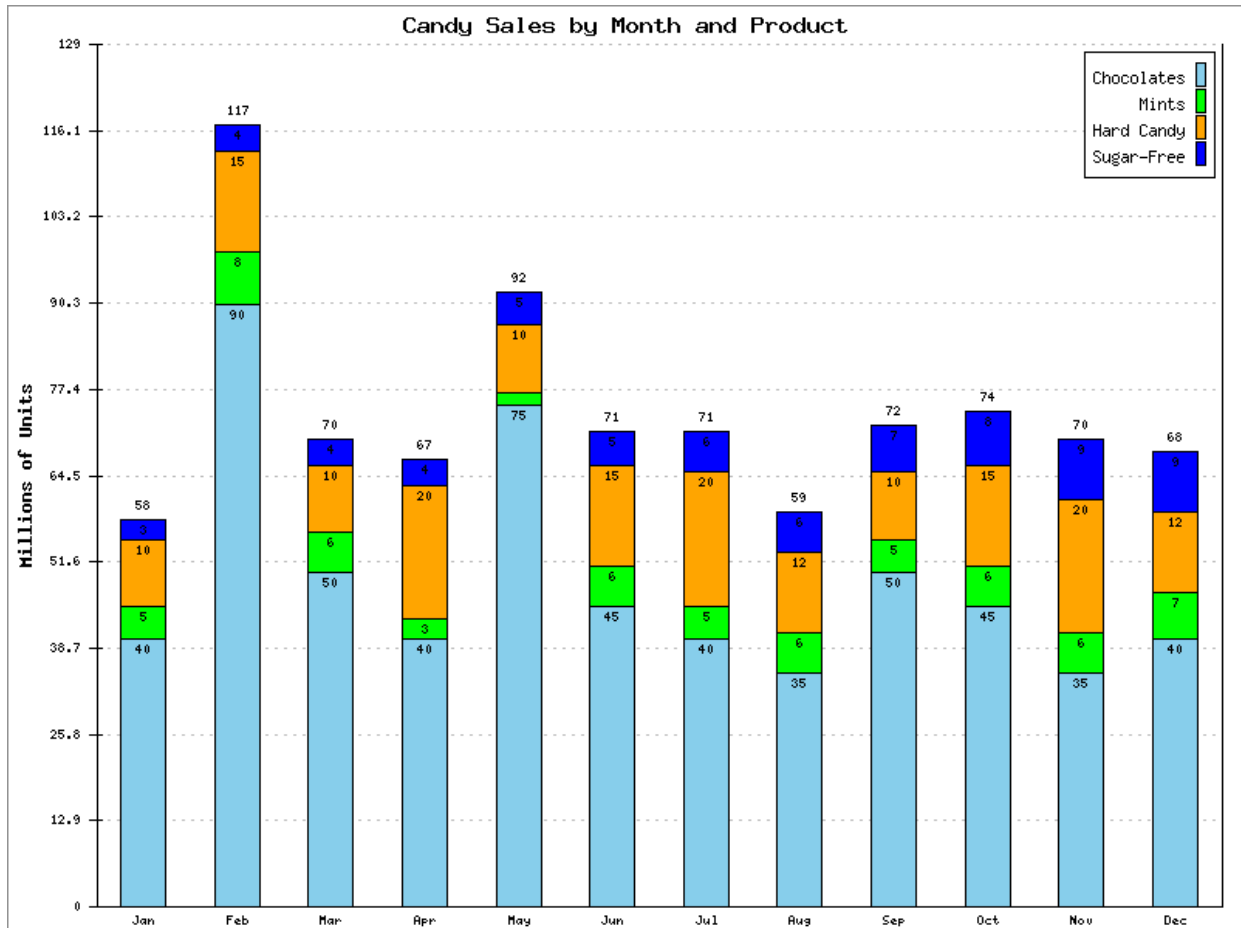

```php
<?php
# PHPlot Example: Stacked Bars, unshaded, with Y data labels
require_once 'phplot.php';

$data = array(
  array('Jan', 40, 5, 10, 3), array('Feb', 90, 8, 15, 4),
  array('Mar', 50, 6, 10, 4), array('Apr', 40, 3, 20, 4),
  array('May', 75, 2, 10, 5), array('Jun', 45, 6, 15, 5),
  array('Jul', 40, 5, 20, 6), array('Aug', 35, 6, 12, 6),
  array('Sep', 50, 5, 10, 7), array('Oct', 45, 6, 15, 8),
  array('Nov', 35, 6, 20, 9), array('Dec', 40, 7, 12, 9),
);

$plot = new PHPlot(800, 600);
```

```
$plot->SetImageBorderType('plain');

$plot->SetPlotType('stackedbars');
$plot->SetDataType('text-data');
$plot->SetDataValues($data);

$plot->SetTitle('Candy Sales by Month and Product');
$plot->SetYTitle('Millions of Units');

# No shading:
$plot->SetShading(0);

$plot->SetLegend(array('Chocolates', 'Mints', 'Hard Candy', 'Sugar-Free'));

$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');

# Turn on Y Data Labels: Both total and segment labels:
$plot->SetYDataLabelPos('plotstack');

$plot->DrawGraph();
```

5.21. Example - Stacked Area Plot

The stacked area plot is similar in appearance to the area plot, and this example is the same as in Section 5.3, "Example - Area Plot" except for the plot type and data values. In the stacked area plot, PHPlot accumulates the Y values at each X position, similar to the stacked bar plot, and fills the area between the resulting values. For example, in 1960, 30% of the candy sales were cherry, 10% were lime, and 6% were lemon. In the stacked area plot, this is represented simply as (30, 10, 6, ...), whereas in the area plot example we had to sum the values to get (100, 70, 60, 54, ...).

Note this plot type was added in PHPlot-5.1.1.

Example 5.21. Stacked Area Plot
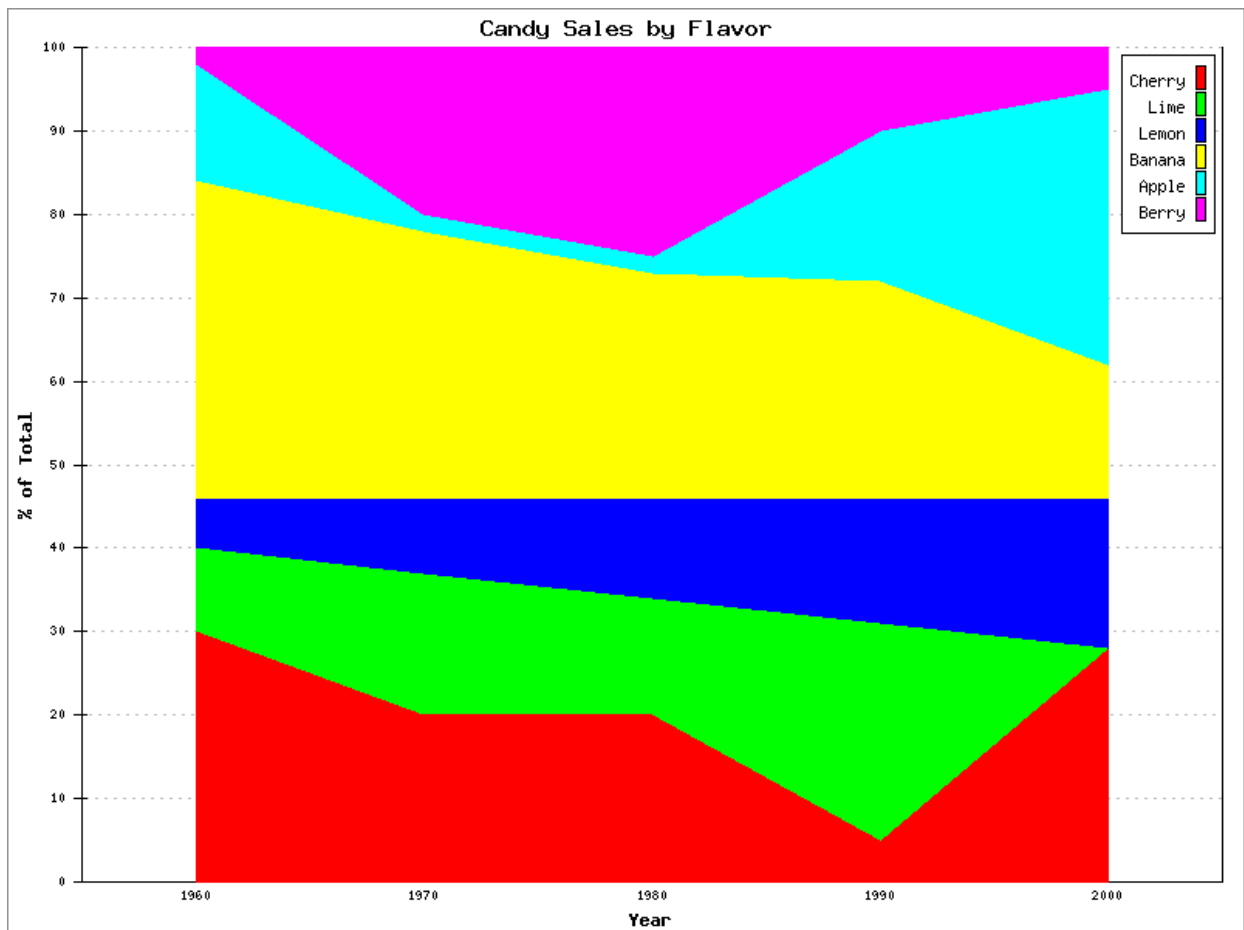

```php
<?php
# PHPlot Example: Stacked Area chart
require_once 'phplot.php';

$data = array(
  array('1960', 30, 10,  6, 38, 14,  2),
  array('1970', 20, 17,  9, 32,  2, 20),
  array('1980', 20, 14, 12, 27,  2, 25),
  array('1990',  5, 26, 15, 26, 18, 10),
  array('2000', 28,  0, 18, 16, 33,  5),
);
```

```
$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain');

$plot->SetPlotType('stackedarea');
$plot->SetDataType('text-data');
$plot->SetDataValues($data);

# Main plot title:
$plot->SetTitle('Candy Sales by Flavor');

# Set Y data limits, tick increment, and titles:
$plot->SetPlotAreaWorld(NULL, 0, NULL, 100);
$plot->SetYTickIncrement(10);
$plot->SetYTitle('% of Total');
$plot->SetXTitle('Year');

# Colors are significant to this data:
$plot->SetDataColors(array('red', 'green', 'blue', 'yellow', 'cyan', 'magenta'));
$plot->SetLegend(array('Cherry', 'Lime', 'Lemon', 'Banana', 'Apple', 'Berry'));

# Turn off X tick labels and ticks because they don't apply here:
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');

$plot->DrawGraph();
```

5.22. Example - Annotating a Plot Using a Callback

This is an advanced example that uses a drawing callback to add annotations to a plot. More information on this topic can be found in Section 4.3.5, "Using Callbacks to Annotate Plots", where this example is described in detail.

Example 5.22. Annotated Plot
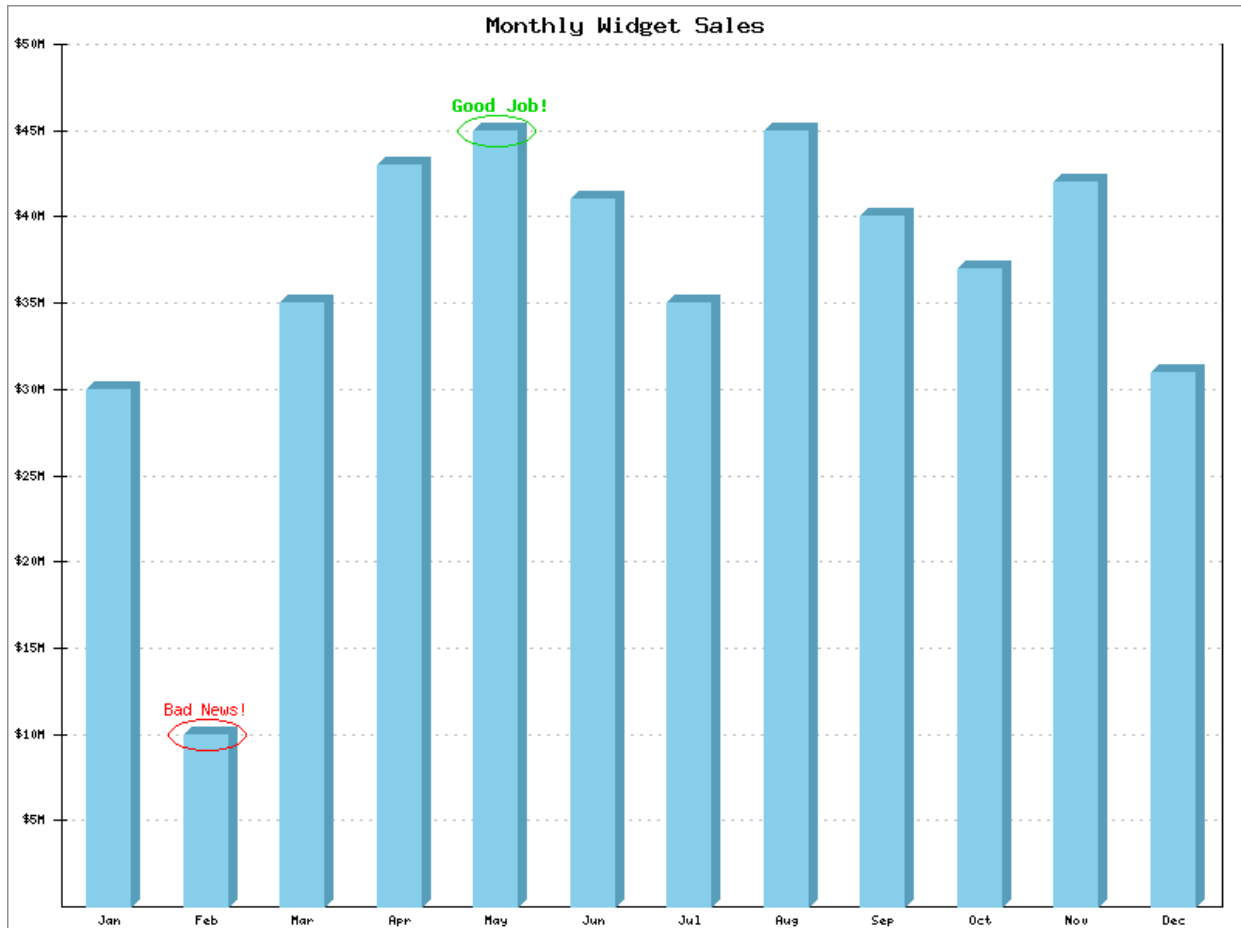

```php
<?php
# PHPlot Example: Annotating a plot using callbacks
# Note: This example is coded for PHPlot > 5.0.7
require_once 'phplot.php';

# Get the Sales data. In real life, this would most likely come from
# a database or external file. For this example, we will use 'random'
# data, but with a fixed seed for repeatable results.
function get_data()
{
    mt_srand(1);
    $data = array();
    # Build an array with 12 arrays of (month_name, value):
    for ($month = 1; $month <= 12; $month++)
```

```
        $data[] = array(strftime('%b', mktime(12, 0, 0, $month, 1)),
                        5 + mt_rand(5, 40));
    return $data;
}


# Find the best and worst sales data.
# Gets the Y value (sales data) and X value. For PHPlot text-data data,
# the X values are assigned as 0.5, 1.5, 2.5, etc.
# The data array is in 'text-data' format: array of array(label, Y)...
function get_best_worst($data,
    &$best_index, &$best_sales, &$worst_index, &$worst_sales)
{
  $best_sales = NULL;
  $worst_sales = NULL;
  foreach ($data as $x => $point) {
      if (!isset($best_sales) || $point[1] > $best_sales) {
          $best_sales = $point[1];
          $best_index = $x + 0.5;
      }
      if (!isset($worst_sales) || $point[1] < $worst_sales) {
          $worst_sales = $point[1];
          $worst_index = $x + 0.5;
      }
  }
}


# Plot annotation callback.
# The pass-through argument is the PHPlot object.
function annotate_plot($img, $plot)
{
    global $best_index, $best_sales, $worst_index, $worst_sales;

    # Allocate our own colors, rather than poking into the PHPlot object:
    $red = imagecolorresolve($img, 255, 0, 0);
    $green = imagecolorresolve($img, 0, 216, 0);

    # Get the pixel coordinates of the data points for the best and worst:
    list($best_x, $best_y) = $plot->GetDeviceXY($best_index, $best_sales);
    list($worst_x, $worst_y) = $plot->GetDeviceXY($worst_index, $worst_sales);

    # Draw ellipses centered on those two points:
    imageellipse($img, $best_x, $best_y, 50, 20, $green);
    imageellipse($img, $worst_x, $worst_y, 50, 20, $red);

    # Place some text above the points:
    $font = '3';
    $fh = imagefontheight($font);
    $fw = imagefontwidth($font);
    imagestring($img, $font, $best_x-$fw*4, $best_y-$fh-10,
                'Good Job!', $green);

    # We can also use the PHPlot internal function for text.
    # It does the center/bottom alignment calculations for us.
    # Specify the font argument as NULL or '' to use the generic one.
    $plot->DrawText('', 0, $worst_x, $worst_y-10, $red,
                'Bad News!', 'center', 'bottom');
}


# Begin main processing:
```

```
# Fill the data array:
$data = get_data();

# Find the best and worst months:
get_best_worst($data, $best_index, $best_sales, $worst_index, $worst_sales);

# Create the PHPlot object, set title, plot type, data array type, and data:
$plot = new PHPlot(800, 600);
$plot->SetTitle('Monthly Widget Sales');
$plot->SetPlotType('bars');
$plot->SetDataType('text-data');
$plot->SetDataValues($data);
# Borders are needed for the manual:
$plot->SetImageBorderType('plain');

# Select X data labels (not tick labels):
$plot->SetXTickPos('none');
$plot->SetXTickLabelPos('none');
$plot->SetXDataLabelPos('plotdown');

# Format Y labels as "$nM" with no decimals, steps of 5:
$plot->SetYLabelType('data', 0, '$', 'M');
$plot->SetYTickIncrement(5.0);

# Force the bottom of the plot to be at Y=0, and omit
# the bottom "$0M" tick label because it looks odd:
$plot->SetPlotAreaWorld(NULL, 0);
$plot->SetSkipBottomTick(True);

# Establish the drawing callback to do the annotation:
$plot->SetCallback('draw_all', 'annotate_plot', $plot);

# Draw the graph:
$plot->DrawGraph();
```

5.23. Example - Complete Web Form with Plot

This section shows a complete mini-application which uses PHPlot to display a graph based on user input through a web form. The purpose of this example is to illustrate form handling and parameter passing from a form-handling script to an image-generating script.

Here a screen-shot of the application, as seen from a web browser. (The bottom section with the graph will only be shown after the form is submitted.)

Example 5.23. Screen Shot of Web Form with Plot
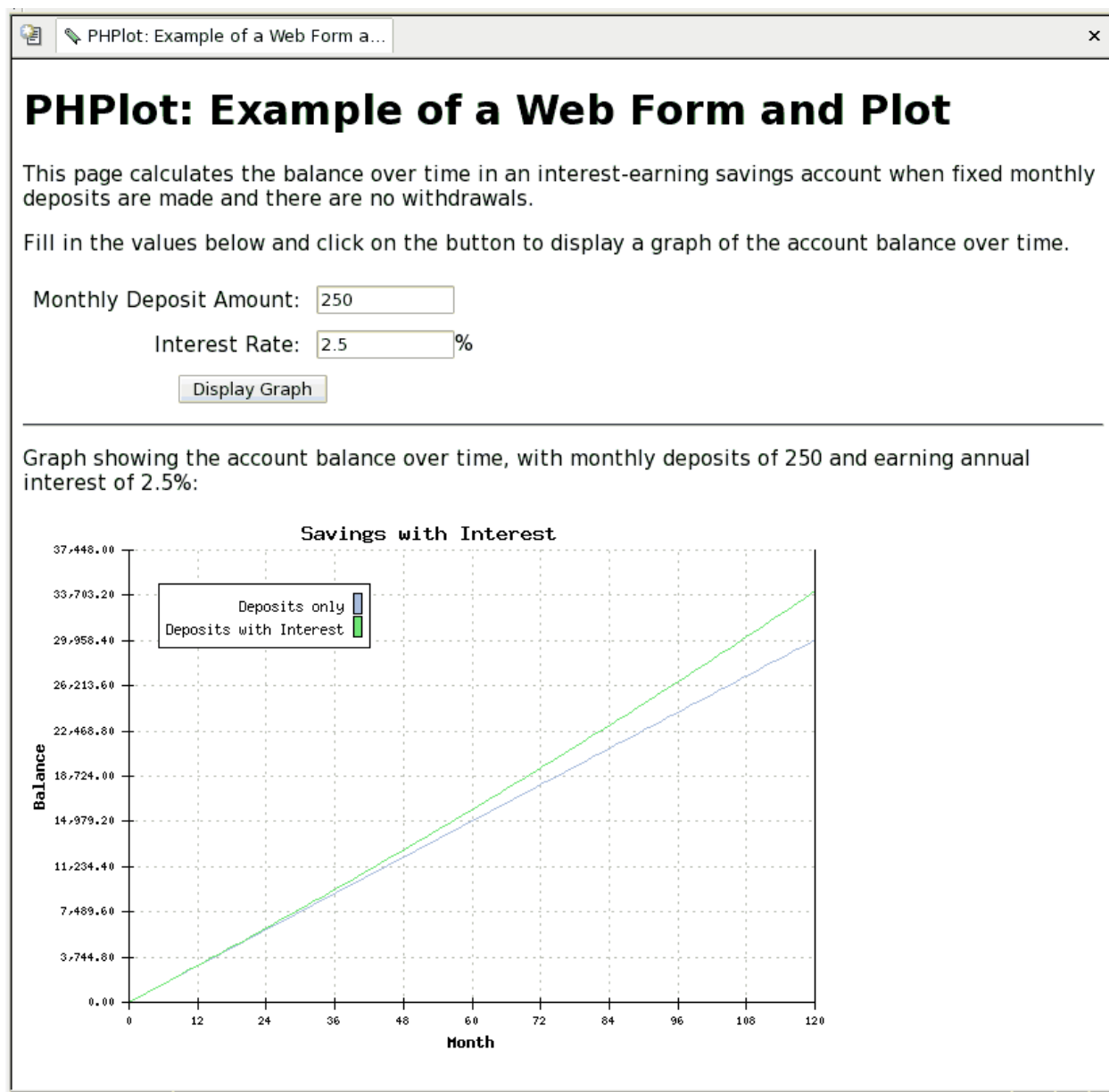
Note

Unlike the other examples in this chapter, the web form example consists of two scripts, and only works with a web server. The two scripts are shown in their entirety, but are broken up into blocks, with comments preceding each block, for presentation purposes.

5.23.1. Web Form Main Script

This section presents the main script `webform.php` which displays the web form and handles form submission. This script does not use PHPlot. When first accessed from a browser (with no parameters), it displays only the form and descriptive text. When the form is submitted, the same script runs again. This time, the script receives form parameters, and displays the graph in addition to the form. To display the graph, the script generates an image (img) tag which references the second script (which is described in the next section). That second script actually generates the plot image.

The script begins with a descriptive comment, and then defines constants for the name of the other script, the image size, and the parameter defaults.

```php
<?php
/*  PHPlot web form example

  Parameter names and parameter array keys:
    'deposit' = Amount deposited per month.
    'intrate' = Interest rate as a percentage (e.g. 4 means 4% or 0.04)
*/


# Name of script which generates the actual plot:
define('GRAPH_SCRIPT', 'webform_img.php');
# Image size. It isn't really necessary that this script know this image
# size, but it improves page rendering.
define('GRAPH_WIDTH', 600);
define('GRAPH_HEIGHT', 400);

# Default values for the form parameters:
$param = array('deposit' => 100.00, 'intrate' => 4.0);
```

Function `build_url()` is a general-purpose function used to generate a URL to a script with parameters. The parameters are in a PHP associative array. The return value is a relative or complete URL which might look like this: `webform_img.php?deposit=100&intrate=4.0&h=400&w=600`.

```php
# Build a URL with escaped parameters:
#    $url - The part of the URL up through the script name
#    $param - Associative array of parameter names and values
# Returns a URL with parameters. Note this must be HTML-escaped if it is
# used e.g. as an href value. (The & between parameters is not pre-escaped.)
function build_url($url, $param)
{
    $sep = '?';  // Separator between URL script name and first parameter
    foreach ($param as $name => $value) {
        $url .= $sep . urlencode($name) . '=' . urlencode($value);
        $sep = '&';   // Separator between subsequent parameters
    }
    return $url;
}
```

The function `begin_page()` creates the HTML at the top of the page. In a real application, this might include a page header.

```
# Output the start of the HTML page:
function begin_page($title)
{
    echo <<<END
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>$title</title>
</head>
<body>
<h1>$title</h1>

END;
}
```

The function `end_page()` creates the HTML at the end of the page. In a real application, this might include a page footer.

```
# Output the bottom of the HTML page:
function end_page()
{
    echo <<<END
</body>
</html>

END;
}
```

The function `show_descriptive_text()` produces HTML text which describes the form. This will go above the form on the web page.

```
# Output text which describes the form.
function show_descriptive_text()
{
    echo <<<END
<p>
This page calculates the balance over time in an interest-earning savings
account when fixed monthly deposits are made and there are no withdrawals.
</p>
<p>
Fill in the values below and click on the button to display a
graph of the account balance over time.
</p>

END;
}
```

The function `show_form()` outputs the HTML form. This includes entry boxes for the two parameters and a submit button. The form action URL is this script itself, so we use the SCRIPT_NAME value to self-reference the script.

```
# Output the web form.
# The form resubmits to this same script for processing.
# The $param array contains default values for the form.
# The values have already been validated as containing numbers and
# do not need escaping for HTML.
function show_form($param)
{
```

```
    $action = htmlspecialchars($_SERVER['SCRIPT_NAME']);

    echo <<<END
<form name="f1" id="f1" method="get" action="$action">
<table cellpadding="5" summary="Entry form for balance calculation">
<tr>
  <td align="right"><label for="deposit">Monthly Deposit Amount:</label></td>
  <td><input type="text" size="10" name="deposit" id="deposit"
      value="{$param['deposit']}">
</tr>
<tr>
  <td align="right"><label for="intrate">Interest Rate:</label></td>
  <td><input type="text" size="10" name="intrate" id="intrate"
      value="{$param['intrate']}">%
</tr>
<tr>
  <td colspan="2" align="center"><input type="submit" value="Display Graph"></td>
</tr>
</table>
</form>

END;
}
```

The function `check_form_params()` performs the important task of validating the parameters received from a form submission. Each parameter is checked for presence and syntax, then converted to the appropriate PHP type. This function is also used to determine if a plot should be displayed. A plot is displayed only if valid form parameters were received.

```
# Check for parameters supplied to this web page.
# If there are valid parameters, store them in the array argument and
# return True.
# If there are no parameters, or the parameters are not valid, return False.
function check_form_params(&$param)
{
    $valid = True;

    if (empty($_GET['deposit']) || !is_numeric($_GET['deposit'])
            || ($deposit = floatval($_GET['deposit'])) < 0)
        $valid = False;

    if (empty($_GET['intrate']) || !is_numeric($_GET['intrate'])
            || ($intrate = floatval($_GET['intrate'])) < 0 || $intrate > 100)
        $valid = False;

    if ($valid) $param = compact('deposit', 'intrate');
    return $valid;
}
```

The function `show_graph()` produces the HTML which will will invoke the second script to produce the graph. This is an image (img) tag which references the second script, including the parameters the script needs to generate the plot. This is one of several ways to pass parameters from the form handling script and the image generating script. The other way is using session variables. Using URL parameters is simpler, especially when there are only a few parameters. Note the HTML also specifies the width and height of the plot image. This is not necessary, however it helps the browser lay out the page without waiting for the image script to complete.

```
# Display a graph.
# The param array contains the validated values: deposit and intrate.
```

```
# This function creates the portion of the page that contains the
# graph, but the actual graph is generated by the $GRAPH_SCRIPT script.
function show_graph($param)
{
    # Include the width and height as parameters:
    $param['w'] = GRAPH_WIDTH;
    $param['h'] = GRAPH_HEIGHT;
    # URL to the graphing script, with parameters, escaped for HTML:
    $img_url = htmlspecialchars(build_url(GRAPH_SCRIPT, $param));

    echo <<<END
<hr>
<p>
Graph showing the account balance over time, with monthly deposits of
{$param['deposit']} and earning annual interest of {$param['intrate']}%:

<p><img src="$img_url" width="{$param['w']}" height="{$param['h']}"
    alt="Account balance graph.">

END;
}
```

Finally, with all the functions defined, the main code is just a few lines.

```
# This is the main processing code.
begin_page("PHPlot: Example of a Web Form and Plot");
$params_supplied = check_form_params($param);
show_descriptive_text();
show_form($param);
if ($params_supplied) show_graph($param);
end_page();
```

5.23.2. Web Form Image Script

This section presents the second script `webform_img.php`, which generates the plot using PHPlot. The URL to this script, along with its parameters, is embedded in the web page produced by the main script in Section 5.23.1, "Web Form Main Script". When the user's browser asks the web server for the image, this second script runs and generates the plot.

The script begins with a descriptive comment and then includes the PHPlot source.

```
<?php
/*  PHPlot web form example - image generation

    This draws the plot image for webform.php
    It expects the following parameters:
        'deposit' = Amount deposited per month. Must be >= 0.
        'intrate' = Interest rate as a percentage (e.g. 4 means 4% or 0.04)
        'w', 'h' = image width and height. (Must be between 100 and 5000)
*/
require_once 'phplot.php';
```

Function `check_form_params()` validates the parameters supplied to the script. Two parameters are required (intrate and deposit), and two are optional (h and w). Even though the main script validated the parameters it passes to this script, it is still necessary for the script to do its own validation. That is because any accessible script can be called from any other web page, or directly from a browser, with arbitrary parameters. (Error handling details can be found below.)

```
# Check for parameters supplied to this web page.
# Parameters must be checked here, even though the calling script checked them,
# because there is nothing stopping someone from calling this script
# directly with arbitrary parameters.
# Parameter values are stored in the param[] array (valid or not).
# If the parameters are valid, return True, else return False.
function check_form_params(&$param)
{
    $valid = True;
    $depost = 0;
    $intrate = 0;

    if (empty($_GET['deposit']) || !is_numeric($_GET['deposit'])
            || ($deposit = floatval($_GET['deposit'])) < 0)
        $valid = False;

    if (empty($_GET['intrate']) || !is_numeric($_GET['intrate'])
            || ($intrate = floatval($_GET['intrate'])) < 0 || $intrate > 100)
        $valid = False;

    # If width and height are missing or invalid, just use something reasonable.
    if (empty($_GET['w']) || !is_numeric($_GET['w'])
            || ($w = intval($_GET['w'])) < 100 || $w > 5000)
        $w = 1024;
    if (empty($_GET['h']) || !is_numeric($_GET['h'])
            || ($h = intval($_GET['h'])) < 100 || $h > 5000)
        $h = 768;

    $param = compact('deposit', 'intrate', 'h', 'w');
    return $valid;
}
```

Function `calculate_data()` computes the data for the plot. This uses the parameters supplied to the script, and populates a data array suitable for PHPlot. Because the script uses the data-data format, each row in the array consists of a label (unused), X value (this is the month number), and 2 Y values (account balance without interest, and account balance with interest).

```
# Calculate the data for the plot:
# This is only called if the parameters are valid.
# The calculation is simple. Each month, two points are calculated: the
# cumulative deposits (balance without interest), and balance with interest.
# At time 0 the balance is 0. At the start of each month, 1/12th of
# the annual interest rate is applied to the balance, and then the deposit
# is added, and that is reported as the balance.
# We calculate for a fixed amount of 120 months (10 years).
function calculate_data($param, &$data)
{
    $deposit = $param['deposit'];
    $monthly_intrate = 1.0 + $param['intrate'] / 100.0 / 12.0;
    $balance_without_interest = 0;
    $balance = 0;
    $data = array(array('', 0, 0, 0)); // Starting point
    for ($month = 1; $month <= 120; $month++) {
        $balance_without_interest += $deposit;
        $balance = $balance * $monthly_intrate + $deposit;
        $data[] = array('', $month, $balance_without_interest, $balance);
    }
}
```

Function `draw_graph()` uses PHPlot to actually produce the graph. This function is similar to the other code examples in this chapter. A PHPlot object is created, set up, and then told to draw the plot. If the script parameters are not valid, however, an attempt is made to draw the plot without a data array. This results in an error, which PHPlot handles by creating an image file with an error message. This method of error handling is used because the script cannot return a textual error message since it is referenced from a web page via an image (img) tag. An alternative to this error handling is to have the script return an HTTP error code such as error 500 (server error).

```
# Draw the graph:
function draw_graph($valid_params, $param, $data)
{
    extract($param);

    $plot = new PHPlot($w, $h);
    $plot->SetTitle('Savings with Interest');
    $plot->SetDataType('data-data');
    # Don't set data values if parameters were not valid. This will result
    # in PHPlot making an image with an error message.
    if ($valid_params) {
        $plot->SetDataValues($data);
    }
    $plot->SetLegend(array('Deposits only', 'Deposits with Interest'));
    $plot->SetLegendPixels(100, 50); // Move legend to upper left
    $plot->SetXTitle('Month');
    $plot->SetXTickIncrement(12);
    $plot->SetYTitle('Balance');
    $plot->SetYLabelType('data', 2);
    $plot->SetDrawXGrid(True);
    $plot->SetPlotType('lines');
    $plot->DrawGraph();
}
```

Lastly, the main code for the image drawing script simply uses the above functions.

```
# This is our main processing code.
$valid_params = check_form_params($param);
if ($valid_params) calculate_data($param, $data);
draw_graph($valid_params, $param, $data);
```

5.24. Example - Using Truecolor To Make a Histogram

This example creates a [Truecolor](#) plot containing a histogram of a photograph, then overlays the histogram on a scaled-down copy of the photograph. The histogram is partly transparent so you can still see the photograph below. Refer to [Section 4.2, "Truecolor Images"](#) for more information on using truecolor PHPlot images.

Here are some notes on the code example:

- The main functions are `get_histogram` and `plot_histogram`. Parameters controlling the histogram and its placement on the image are in an array passed to `plot_histogram`. For the purpose of this demo, the array `$param` is used, and there is no provision to change the parameters or the photograph filename.

- This isn't a 'true' histogram, because the Y values are automatically scaled by PHPlot so they fill the available height. This could be called a 'relative histogram', with the heights indicating the relative count of pixels in the image with that value.

- The histogram is created by converting each pixel's R, G, B color values to a grayscale value between 0 and 255, and counting the number of times each value appears in the image.

- The photograph image is scaled by PHPlot to fit into the background of the plot image using [SetBgImage](#). The histogram is then drawn into an area restricted using [SetPlotAreaPixels](#), leaving most of the background image unobscured. All labels and tick marks are turned off. The plot data colors are set to be partly transparent using the default alpha argument to [SetDataColors](#).

Note

This is a demonstration only. Processing individual pixels in nested loops this way using PHP code is not recommended, because it is very slow. A small image file (800x600) might be processed in a few seconds, but a larger file such as a 12 megapixel photograph could take 30 seconds, for example.

Example 5.24. Truecolor Plot of Histogram
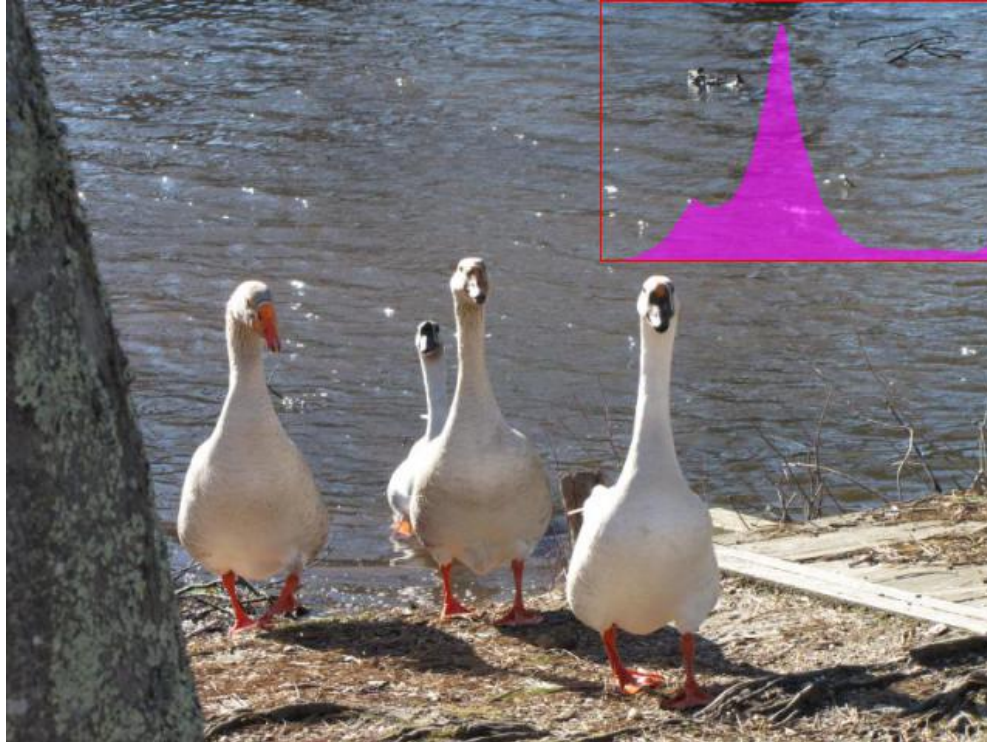

```php
<?php
# PHPlot Example - Histogram of a Photograph
# Display a photo image with its value histogram overlaid
# Note: This requires PHPlot-5.1.1 or higher for Truecolor support.
# Unlike the other examples, and contrary to the usual PHPlot recommendation,
# this scripts creates JPEG not PNG, because most of the image is the original
# photograph and PNG results in an overlarge file.
require_once 'phplot.php';

# Tunable parameters:
$param = array(
    'plot_image_width' => 640,       # Width of final image
    'plot_image_height' => 480,      # Height of final image
    'histogram_color' => 'magenta',  # Color to use for histogram lines
    'histogram_alpha' => 50,         # Histogram transparency (0=opaque, 127=clear)
    'draw_border' => True,           # If true, put a border around the histogram
    'border_color' => 'red',         # Border color, if draw_border is true
    'hx' => 0.6,                     # Upper left X relative position of histogram
    'hy' => 0.0,                     # Upper left Y relative position of histogram
    'h_width' => 0.4,                # Relative width of histogram
    'h_height' => 0.35,              # Relative height of histogram
);

/*
  Make a histogram from an image file, which can be palette or truecolor.
  Returns an array $histogram[i] where i is from 0 to 255. Each histogram[i]
  is the number of pixels in the image with grayscale value i.
  (Grayscale is computed using the NTSC formula, but with integers.)
*/
function get_histogram($image_file)
```

```
{
    list($width, $height, $imtype) = getimagesize($image_file);
    if (!empty($width)) {
        switch ($imtype) {
            case IMAGETYPE_JPEG:
                $im = imagecreatefromjpeg($image_file);
                break;
            case IMAGETYPE_PNG:
                $im = imagecreatefrompng($image_file);
                break;
            case IMAGETYPE_GIF:
                $im = imagecreatefromgif($image_file);
                break;
        }
    }
    if (empty($width) || empty($im)) {
        fwrite(STDERR, "Error invalid image file name: $image_file\n");
        return NULL;
    }

    # Initialize the histogram counters:
    $histogram = array_fill(0, 256, 0);

    # Process every pixel. Get the color components and compute the gray value.
    for ($y = 0; $y < $height; $y++) {
        for ($x = 0; $x < $width; $x++) {
            $pix = imagecolorsforindex($im, imagecolorat($im, $x, $y));
            $value = (int)((30 * $pix['red'] + 59 * $pix['green']
                           + 11 * $pix['blue']) / 100);
            $histogram[$value]++;
        }
    }
    return $histogram;
}

/*
  Make a 'plot', containing a scaled-down version of an image with
  a histogram overlay.
*/
function plot_histogram($image_filename, $param)
{
    extract($param);
    $histo = get_histogram($image_filename);
    if (empty($histo)) return;
    for ($i = 0; $i < 256; $i++) $data[$i] = array('', $histo[$i]);
    $p = new PHPlot_truecolor($plot_image_width, $plot_image_height);
    $p->SetFileFormat('jpg');
    $p->SetBgImage($image_filename, 'scale');
    $p->SetDataType('text-data');
    $p->SetDrawXAxis(False);
    $p->SetDrawYAxis(False);
    $p->SetDataValues($data);
    $p->SetXDataLabelPos('none');
    $p->SetXTickLabelPos('none');
    $p->SetYTickLabelPos('none');
    $p->SetXTickPos('none');
    $p->SetYTickPos('none');
    $p->SetDrawYGrid(False);
    $p->SetDataColors($histogram_color, NULL, $histogram_alpha);
    $p->SetPlotType('thinbarline');
```

```
    if ($draw_border) {
        $p->SetGridColor($border_color);
        $p->SetPlotBorderType('full');
    }
    # Compute the position of the histogram plot within the image.
    $hx0 = (int)($hx * $plot_image_width);
    $hy0 = (int)($hy * $plot_image_height);
    $hx1 = (int)($h_width * $plot_image_width) + $hx0;
    $hy1 = (int)($h_height * $plot_image_height) + $hy0;
    $p->SetPlotAreaPixels($hx0, $hy0, $hx1, $hy1);
    $p->DrawGraph();
}

/* Demo main. */
plot_histogram('examples/geese.jpg', $param);
```

5.25. Example - Creative Use of the Data Color Callback

This example uses the data_color callback to vary the colors used in a thinbarline plot. The callback function getcolor simply returns the row number, which corresponds to each point's position along the X axis. PHPlot will therefore use a different color for each plotted point (modulo the number of defined colors). A large data color array is also defined, with colors set to shades of blue from dark to light and back to dark.

A truecolor plot image is used to allow for more colors than would be allowed in a palette image.

Using the data color callback is described in Section 4.4, "Custom Data Color Selection". More information on callbacks can be found in Section 4.3, "Callbacks". More information on truecolor images can be found in Section 4.2, "Truecolor Images".

Example 5.25. Creative Use of the Data Color Callback
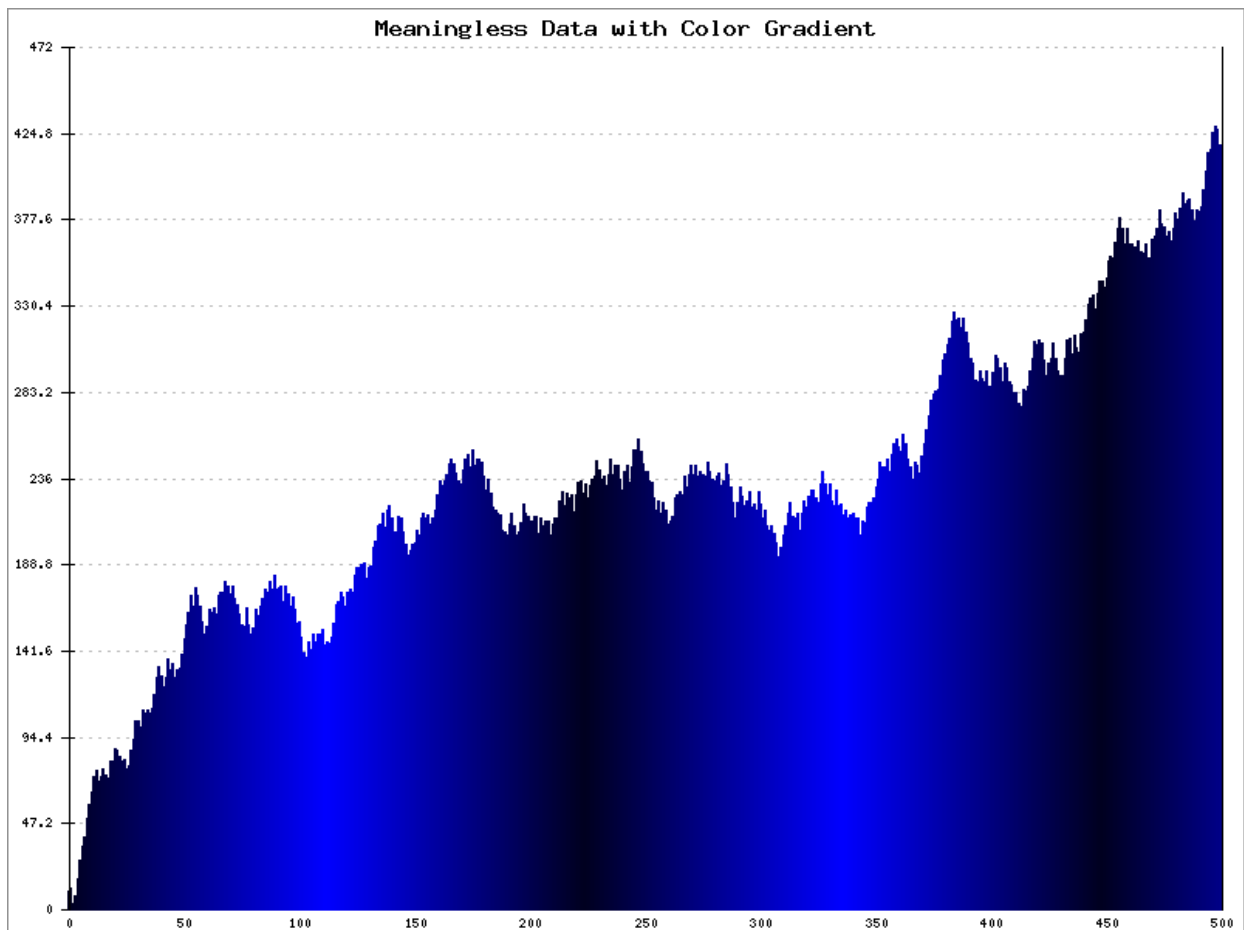

```php
<?php
# PHPlot Example: Creative use of data colors
require_once 'phplot.php';

# Callback for picking a data color.
```

```
# PHPlot will call this every time it needs a data color.
# This simply returns the row number as the color index.
function getcolor($img, $unused, $row, $col)
{
  return $row; // Use row, rather than column, as color index.
}

# Make some pseudo-random data.
mt_srand(1);
$data = array();
$value = 10;
for ($i = 0; $i < 500; $i++) {
  $data[] = array('', $i, $value);
  $value = max(0, $value + mt_rand(-9, 10));
}

# Make a color gradient array of blue:
$colors = array();
for ($b = 32; $b <= 255; $b += 2) $colors[] = array(0, 0, $b);
for ($b = 255; $b >= 32; $b -= 2) $colors[] = array(0, 0, $b);

# Use a truecolor plot image in order to get more colors.
$plot = new PHPlot_truecolor(800, 600);
$plot->SetImageBorderType('plain'); // Improves presentation in the manual

$plot->SetPlotType('thinbarline');
$plot->SetDataType('data-data');
$plot->SetDataValues($data);
$plot->SetLineWidths(2);
$plot->SetDataColors($colors);
$plot->SetXTickPos('none');
$plot->SetPlotAreaWorld(0, 0, 500, NULL);
$plot->SetTitle('Meaningless Data with Color Gradient');

# Establish the function 'getcolor' as a data color selection callback.
$plot->SetCallback('data_color', 'getcolor');

$plot->DrawGraph();
```

5.26. Example - Custom Bar Colors Using the Data Color Callback

This example uses the data_color callback to customize the colors in a bar chart. The goal is to have the bar colors depend on the value of the bar (rather than the position of the bar in the bar group). Bars above 80% will be drawn in green, bars below 60% will be red, and bars in between those two values will be yellow.

The function `pickcolor` is the data_color callback. It accesses the data array using its pass-through argument, indexing into it with the current row and column. (Note col+1 is used to skip over the row label.) It then checks the data value, and returns an index into the data colors array: 0, 1, or 2, depending on the value.

Using the data color callback is described in Section 4.4, "Custom Data Color Selection". More information on callbacks can be found in Section 4.3, "Callbacks".

Example 5.26. Custom Bar Colors Using the Data Color Callback
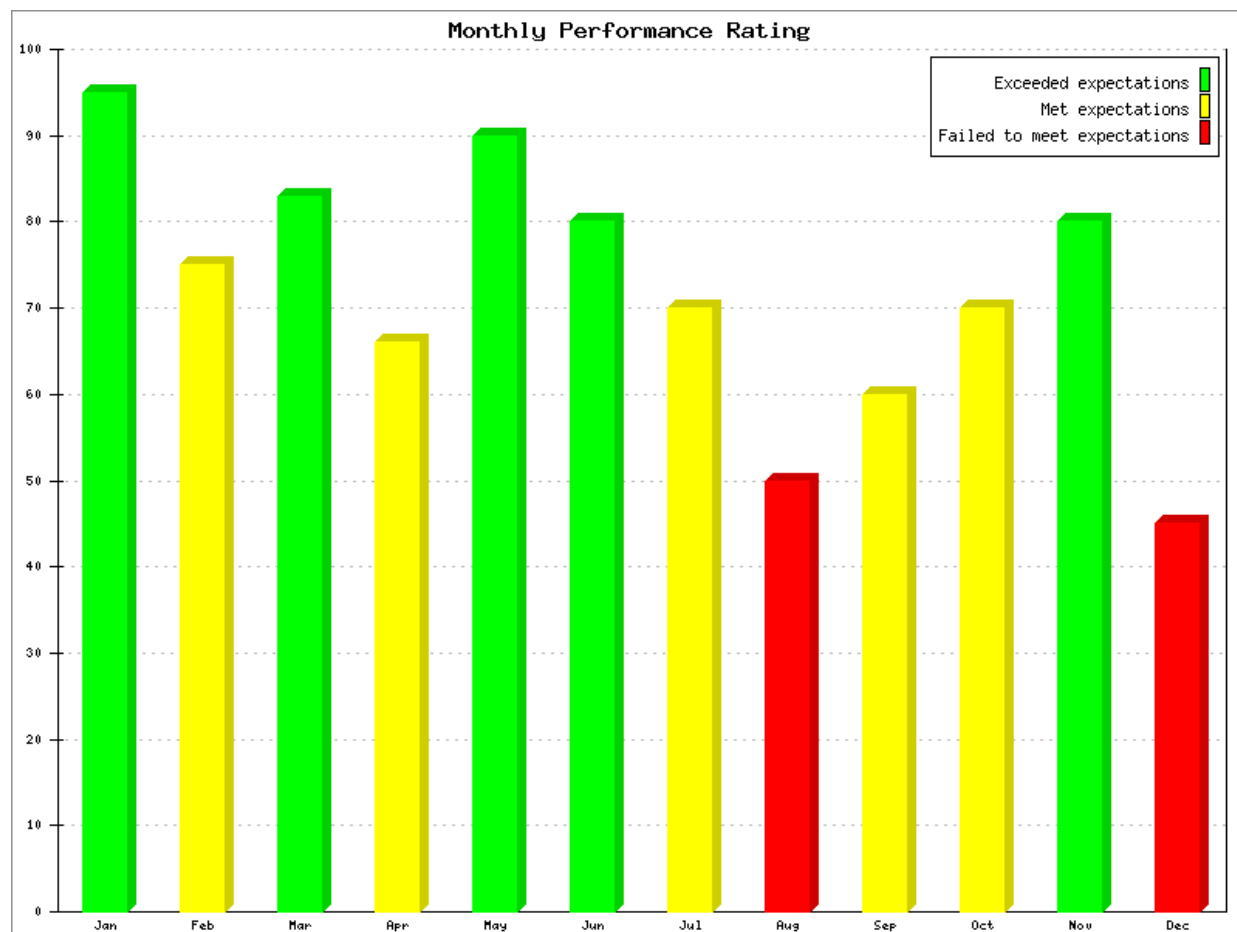

```
<?php
# PHPlot Example: Bar chart with bar color depending on value
require_once 'phplot.php';

# Callback for picking a data color.
```

```
# PHPlot will call this every time it needs a data color.
# This returns a color index which depends on the data value.
# Color 0 is for values >= 80%, 1 is for >= 60%, 2 is for < 60%.
# The data_array must have 'text-data' type.
function pickcolor($img, $data_array, $row, $col)
{
  $d = $data_array[$row][$col+1]; // col+1 skips over the row's label
  if ($d >= 80) return 0;
  if ($d >= 60) return 1;
  return 2;
}

# The data array has our monthly performance as a percentage.
$data = array(
    array('Jan',  95), array('Feb',  75), array('Mar',  83),
    array('Apr',  66), array('May',  90), array('Jun',  80),
    array('Jul',  70), array('Aug',  50), array('Sep',  60),
    array('Oct',  70), array('Nov',  80), array('Dec',  45),
);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain'); // Improves presentation in the manual
$plot->SetPlotType('bars');
$plot->SetDataValues($data);
$plot->SetDataType('text-data');
$plot->SetTitle('Monthly Performance Rating');

# Turn off X Tick labels which have no meaning here.
$plot->SetXTickPos('none');

# Force the Y axis to be exactly 0:100
$plot->SetPlotAreaWorld(NULL, 0, NULL, 100);

# Establish the function 'pickcolor' as a data color selection callback.
# Set the $data array as the pass-through argument, so the function has
# access to the data values without relying on global variables.
$plot->SetCallback('data_color', 'pickcolor', $data);

# The three colors are meaningful to the data color callback.
$plot->SetDataColors(array('green', 'yellow', 'red'));

# The legend will explain the use of the 3 colors.
$plot->SetLegend(array('Exceeded expectations', 'Met expectations',
  'Failed to meet expectations'));

$plot->DrawGraph();
```

5.27. Example - Horizontal Bar Chart

This example shows a horizontal bar chart. As always, the X axis is horizontal, and the Y axis is vertical. But the data array contains the X value for each implicit Y. The data type 'text-data-yx' indicates this is a horizontal plot.

This example also has a tiled background image under the plot area.

Example 5.27. Horizontal Bar Chart
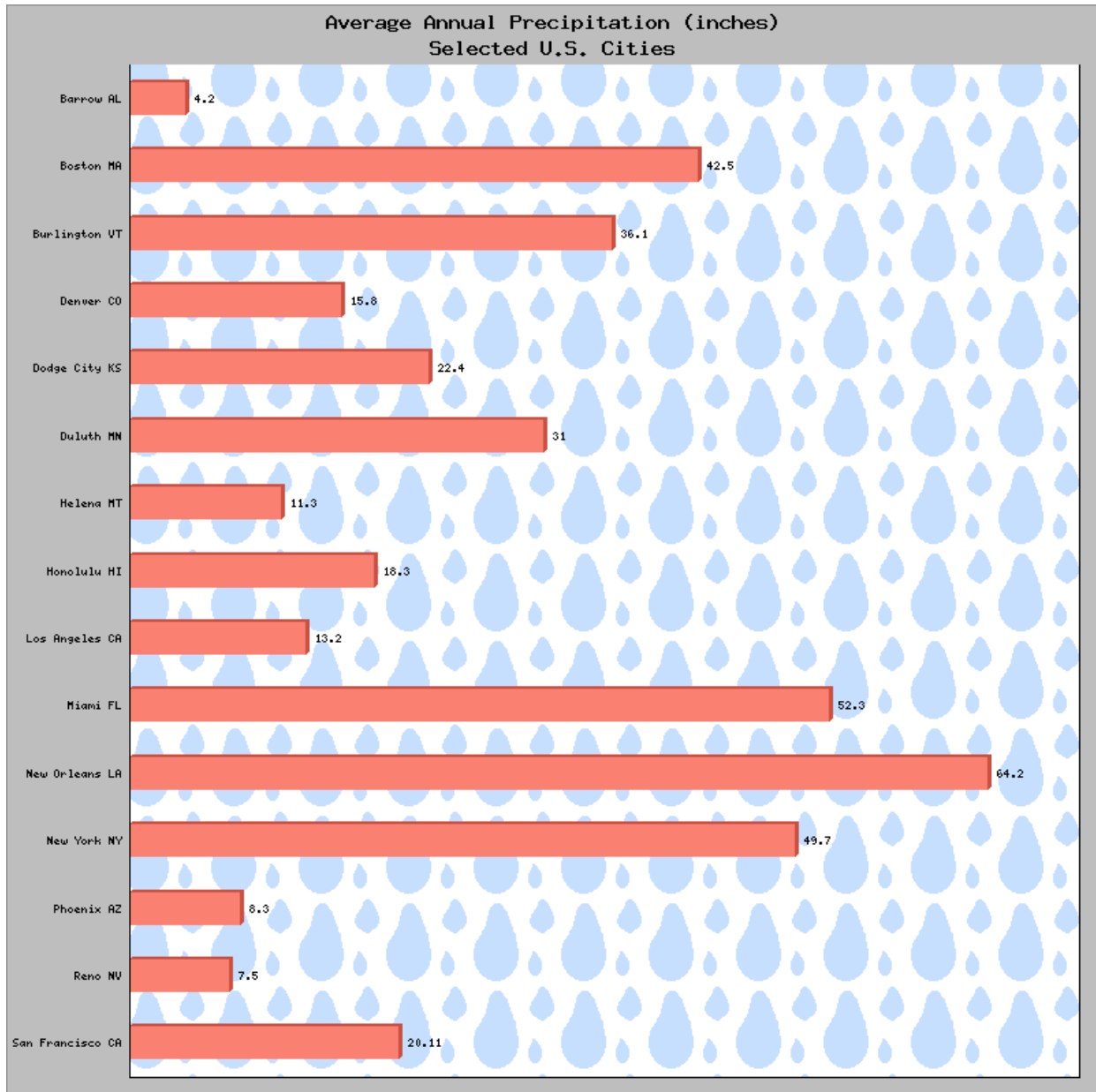

```
<?php
# PHPlot Example - Horizontal Bars
require_once 'phplot.php';
```

```
$data = array(
  array('San Francisco CA', 20.11),
  array('Reno NV', 7.5),
  array('Phoenix AZ', 8.3),
  array('New York NY', 49.7),
  array('New Orleans LA', 64.2),
  array('Miami FL', 52.3),
  array('Los Angeles CA', 13.2),
  array('Honolulu HI', 18.3),
  array('Helena MT', 11.3),
  array('Duluth MN', 31.0),
  array('Dodge City KS', 22.4),
  array('Denver CO', 15.8),
  array('Burlington VT', 36.1),
  array('Boston MA', 42.5),
  array('Barrow AL', 4.2),
);


$plot = new PHPlot(800, 800);
$plot->SetImageBorderType('plain'); // Improves presentation in the manual
$plot->SetTitle("Average Annual Precipitation (inches)\n"
              . "Selected U.S. Cities");
$plot->SetBackgroundColor('gray');
#  Set a tiled background image:
$plot->SetPlotAreaBgImage('images/drop.png', 'centeredtile');
#  Force the X axis range to start at 0:
$plot->SetPlotAreaWorld(0);
#  No ticks along Y axis, just bar labels:
$plot->SetYTickPos('none');
#  No ticks along X axis:
$plot->SetXTickPos('none');
#  No X axis labels. The data values labels are sufficient.
$plot->SetXTickLabelPos('none');
#  Turn on the data value labels:
$plot->SetXDataLabelPos('plotin');
#  No grid lines are needed:
$plot->SetDrawXGrid(FALSE);
#  Set the bar fill color:
$plot->SetDataColors('salmon');
#  Use less 3D shading on the bars:
$plot->SetShading(2);
$plot->SetDataValues($data);
$plot->SetDataType('text-data-yx');
$plot->SetPlotType('bars');
$plot->DrawGraph();
```

5.28. Example - Horizontal Stacked Bar Chart

This example shows a horizontal stacked bar chart. As always, the X axis is horizontal, and the Y axis is vertical. But the data array contains the X value for each implicit Y. The data type 'text-data-yx' indicates this is a horizontal plot.

Example 5.28. Horizontal Stacked Bar Chart
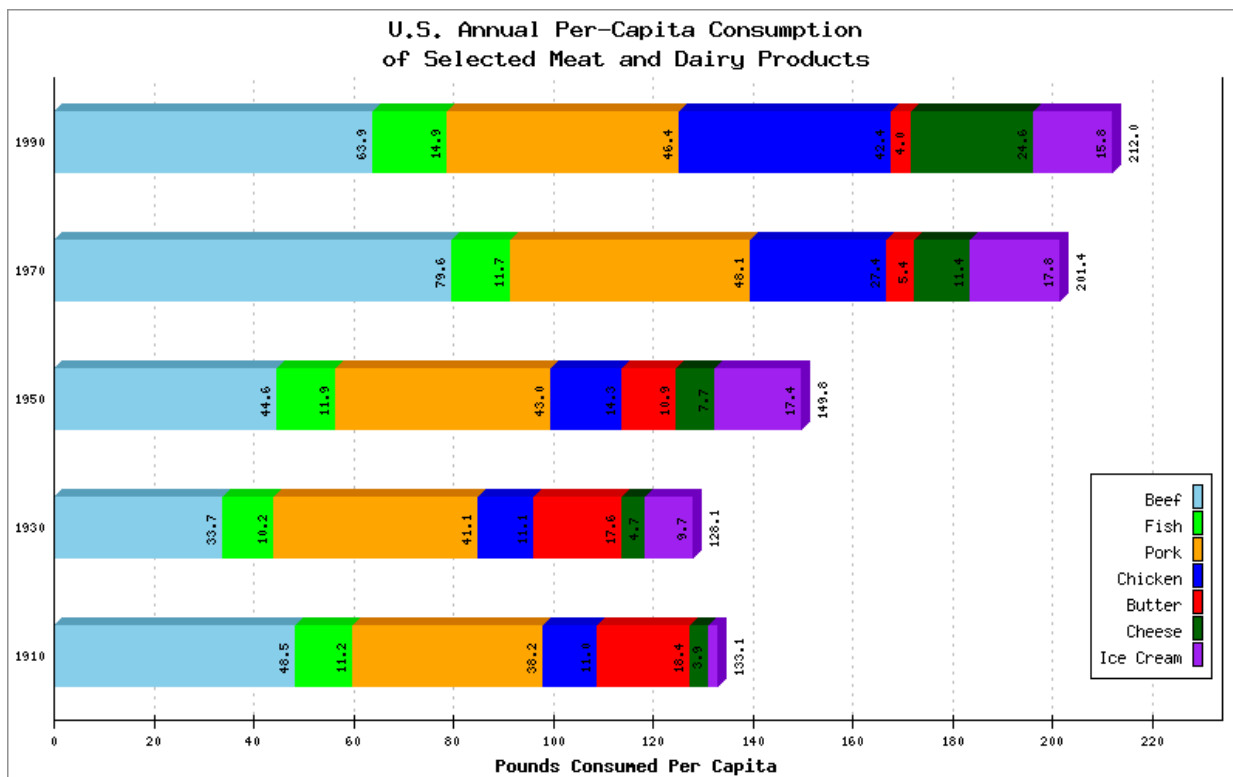

```php
<?php
# PHPlot Example - Horizontal Stacked Bars
require_once 'phplot.php';

$column_names = array(
                'Beef', 'Fish', 'Pork', 'Chicken', 'Butter',
                                                'Cheese',
                                                    'Ice Cream');
//                      |       |       |       |       |       |       |
$data = array(
    array('1910',   48.5,   11.2,   38.2,   11.0,   18.4,    3.9,    1.9),
    array('1930',   33.7,   10.2,   41.1,   11.1,   17.6,    4.7,    9.7),
    array('1950',   44.6,   11.9,   43.0,   14.3,   10.9,    7.7,   17.4),
    array('1970',   79.6,   11.7,   48.1,   27.4,    5.4,   11.4,   17.8),
    array('1990',   63.9,   14.9,   46.4,   42.4,    4.0,   24.6,   15.8),
);
$plot = new PHPlot(800, 500);
$plot->SetImageBorderType('plain'); // Improves presentation in the manual
$plot->SetTitle("U.S. Annual Per-Capita Consumption\n"
            . "of Selected Meat and Dairy Products");
$plot->SetLegend($column_names);
```

```
#  Move the legend to the lower right of the plot area:
$plot->SetLegendPixels(700, 300);
$plot->SetDataValues($data);
$plot->SetDataType('text-data-yx');
$plot->SetPlotType('stackedbars');
$plot->SetXTitle('Pounds Consumed Per Capita');
#  Show data value labels:
$plot->SetXDataLabelPos('plotstack');
#  Rotate data value labels to 90 degrees:
$plot->SetXDataLabelAngle(90);
#  Format the data value labels with 1 decimal place:
$plot->SetXDataLabelType('data', 1);
#  Specify a whole number for the X tick interval:
$plot->SetXTickIncrement(20);
#  Disable the Y tick marks:
$plot->SetYTickPos('none');
$plot->DrawGraph();
```

5.29. Example - Horizontal Thin Bar Line Plot

This example shows a horizontal thin bar line (thinbarline) plot. As always, the X axis is horizontal, and the Y axis is vertical. The data type 'data-data-yx' indicates this is a horizontal plot, with explicit independent (Y) variables. There is one X for each Y. With this data type, the independent variable values need not be in order or sequential in the data array.

Example 5.29. Horizontal Thin Bar Line Plot
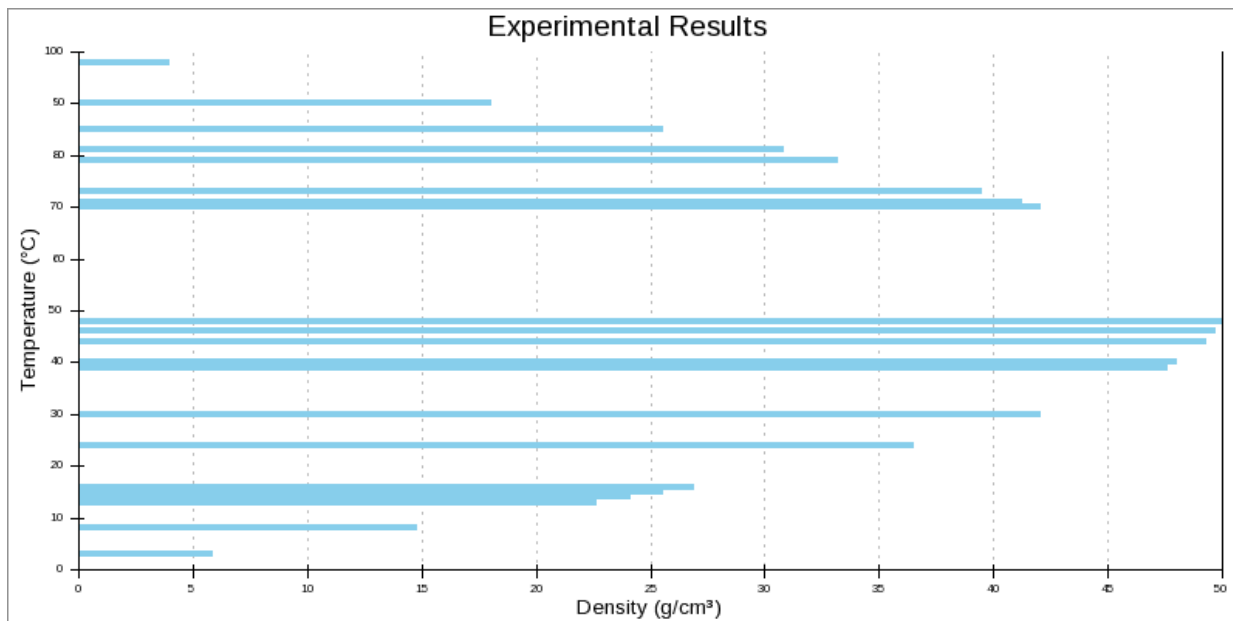

```php
<?php
# PHPlot example - horizontal thinbarline plot (impulse plot)
require_once 'phplot.php';
$data = array(
    array('',  79, 33.18), array('',  13, 22.62), array('',  71, 41.18),
    array('',   8, 14.72), array('',  48, 49.92), array('',  46, 49.68),
    array('',  90, 18.00), array('',  15, 25.50), array('',  73, 39.42),
    array('',  30, 42.00), array('',  24, 36.48), array('',  85, 25.50),
    array('',  14, 24.08), array('',   3,  5.82), array('',  98,  3.92),
    array('',  39, 47.58), array('',  70, 42.00), array('',  16, 26.88),
    array('',  81, 30.78), array('',  40, 48.00), array('',  44, 49.28),
);
$plot = new PHPlot(800, 400);
$plot->SetImageBorderType('plain'); // Improves presentation in the manual
$plot->SetUseTTF(True);
$plot->SetTitle('Experimental Results');
$plot->SetXTitle('Density (g/cm&#179;)'); // 179=superscript 3
$plot->SetYTitle('Temperature (&#176;C)'); // 176=degrees
$plot->SetPlotType('thinbarline');
$plot->SetDataType('data-data-yx');
$plot->SetDataValues($data);
$plot->SetPlotAreaWorld(0, 0, 50, 100);
$plot->SetLineWidths(4);
$plot->DrawGraph();
```

5.30. Example - Basic OHLC (Open, High, Low, Close) Financial Plot

This example shows an ohlc plot, which is a basic form of the Open, High, Low, Close (OHLC) financial plot. Each X is a point in time or interval, and there are 4 corresponding Y values for the four prices (open, high, low, and close). Compare this with the next two examples, Example 5.31, "Candlesticks OHLC Plot" and Example 5.32, "Filled Candlesticks OHLC Plot", which show the same data but with a different presentation.

In this example, the data array is read from an external file in Comma Separated Value (CSV) format. (Financial data in this format is available for download from sites such as Yahoo! Finance.) An portion example of the data file can be found below.

This example uses the dates from the data file as row labels in the data array, with text-data data format. For this to work, the rows have to be sorted by increasing date, so the read_prices_text_data() first reads the data into a temporary array, sorts by the date, then copies the data into a PHPlot data array. Compare this with the other two OHLC examples, where the same data is used differently.

Example 5.30. Basic OHLC Plot
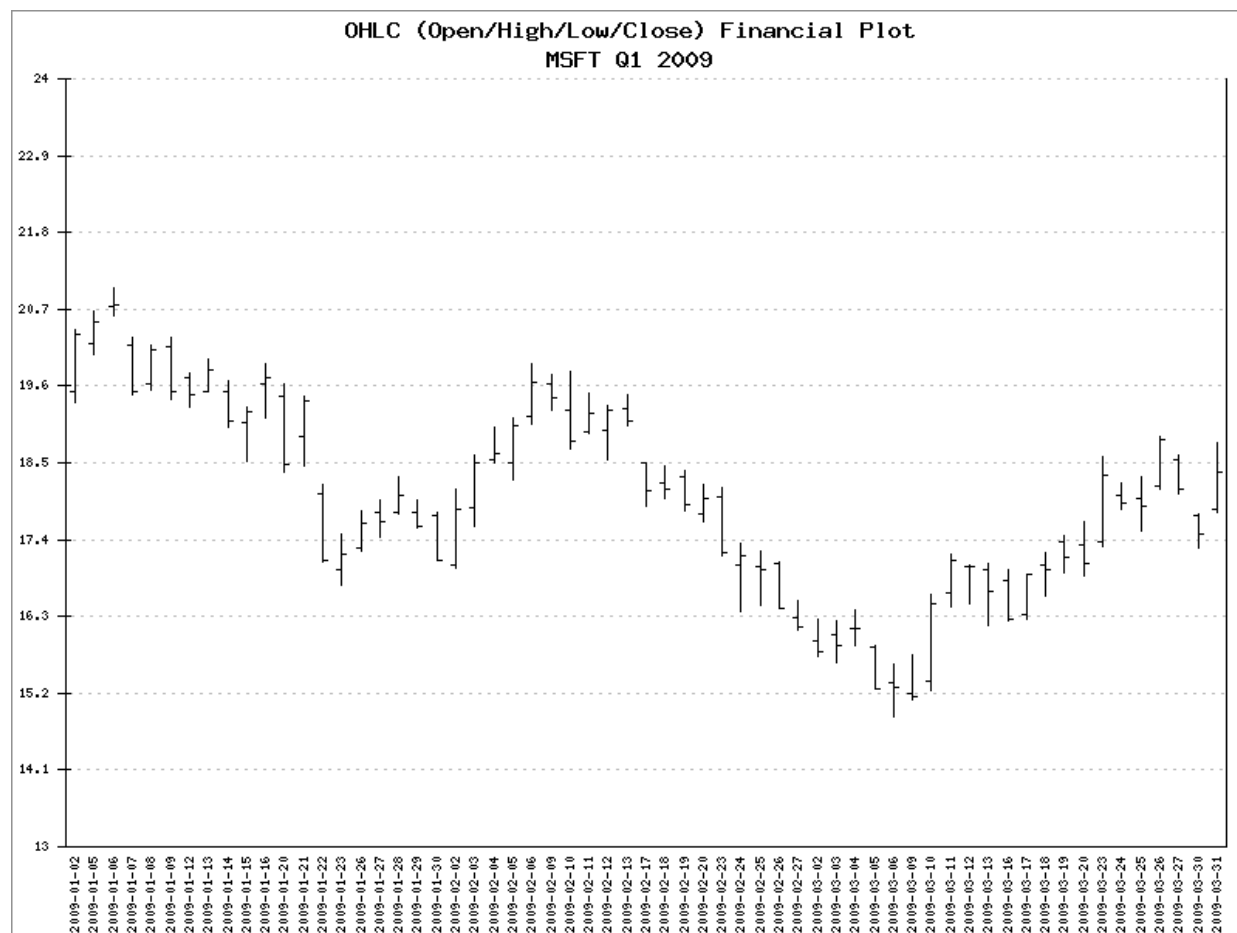

```php
<?php
# PHPlot Example: OHLC (Financial) plot, basic lines, using
```

```
# external data file, text-data format.
define('DATAFILE', 'examples/ohlcdata.csv'); // External data file
require_once 'phplot.php';

/*
  Read historical price data from a CSV data downloaded from Yahoo! Finance.
  The first line is a header which must contain: Date,Open,High,Low,Close[...]
  Each additional line has a date (YYYY-MM-DD), then 4 price values.
  The rows have to be sorted by date, because the original is reversed.
  This version of the function uses the date as a label, and returns a
  text-data (implied X) PHPlot data array.
*/
function read_prices_text_data($filename)
{
    $f = fopen($filename, 'r');
    if (!$f) {
        fwrite(STDERR, "Failed to open data file: $filename\n");
        return FALSE;
    }
    // Read and check the file header.
    $row = fgetcsv($f);
    if ($row === FALSE || $row[0] != 'Date' || $row[1] != 'Open'
            || $row[2] != 'High' || $row[3] != 'Low' || $row[4] != 'Close') {
        fwrite(STDERR, "Incorrect header in: $filename\n");
        return FALSE;
    }
    // Read the rest of the file into array keyed by date for sorting.
    while ($r = fgetcsv($f)) {
        $d[$r[0]] = array($r[1], $r[2], $r[3], $r[4]);
    }
    fclose($f);
    ksort($d);
    // Convert to a PHPlot data array with label and 4 values per row.
    foreach ($d as $date => $r) {
        $data[] = array($date, $r[0], $r[1], $r[2], $r[3]);
    }
    return $data;
}

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain'); // Improves presentation in the manual
$plot->SetTitle("OHLC (Open/High/Low/Close) Financial Plot\nMSFT Q1 2009");
$plot->SetDataType('text-data');
$plot->SetDataValues(read_prices_text_data(DATAFILE));
$plot->SetPlotType('ohlc');
$plot->SetDataColors('black');
$plot->SetXLabelAngle(90);
$plot->SetXTickPos('none');
$plot->DrawGraph();
```

Here is the top portion of the data file used for the three OHLC examples. This file is called ohlcdata.csv.

```
Date,Open,High,Low,Close,Volume,Adj Close
2009-03-31,17.83,18.79,17.78,18.37,92095500,17.81
2009-03-30,17.74,17.76,17.27,17.48,49633000,16.95
2009-03-27,18.54,18.62,18.05,18.13,47670400,17.58
2009-03-26,18.17,18.88,18.12,18.83,63775100,18.26
2009-03-25,17.98,18.31,17.52,17.88,73927100,17.34
...
```

5.31. Example - Candlesticks OHLC (Open, High, Low, Close) Financial Plot

This example shows a [candlesticks](#) plot, which is a form of the Open, High, Low, Close (OHLC) financial plot. Each X is a point in time or interval, and there are 4 corresponding Y values for the four prices (open, high, low, and close). Compare this with [Example 5.30, "Basic OHLC Plot"](#) and [Example 5.32, "Filled Candlesticks OHLC Plot"](#), which show the same data but with a different presentation.

The data values for this example are read from an external file. Refer to [Section 5.30, "Example - Basic OHLC (Open, High, Low, Close) Financial Plot"](#) for more information. Unlike that example, this candlesticks example uses the dates from the data file as the X values in the data array, with the PHPlot data type data-data. The dates read from the file are converted into timestamp values with strtotime() inside read_prices_data_data(). Since each row in a data-data array specifies its X value, the rows do not need to be sorted (as they were in the previous example).

PHPlot will format the X values as dates because [SetXLabelType](#) is used to select date/time formatting. Unlike the text-data example, this also lets us use [SetXTickIncrement](#) to control the label density along the X axis.

Example 5.31. Candlesticks OHLC Plot
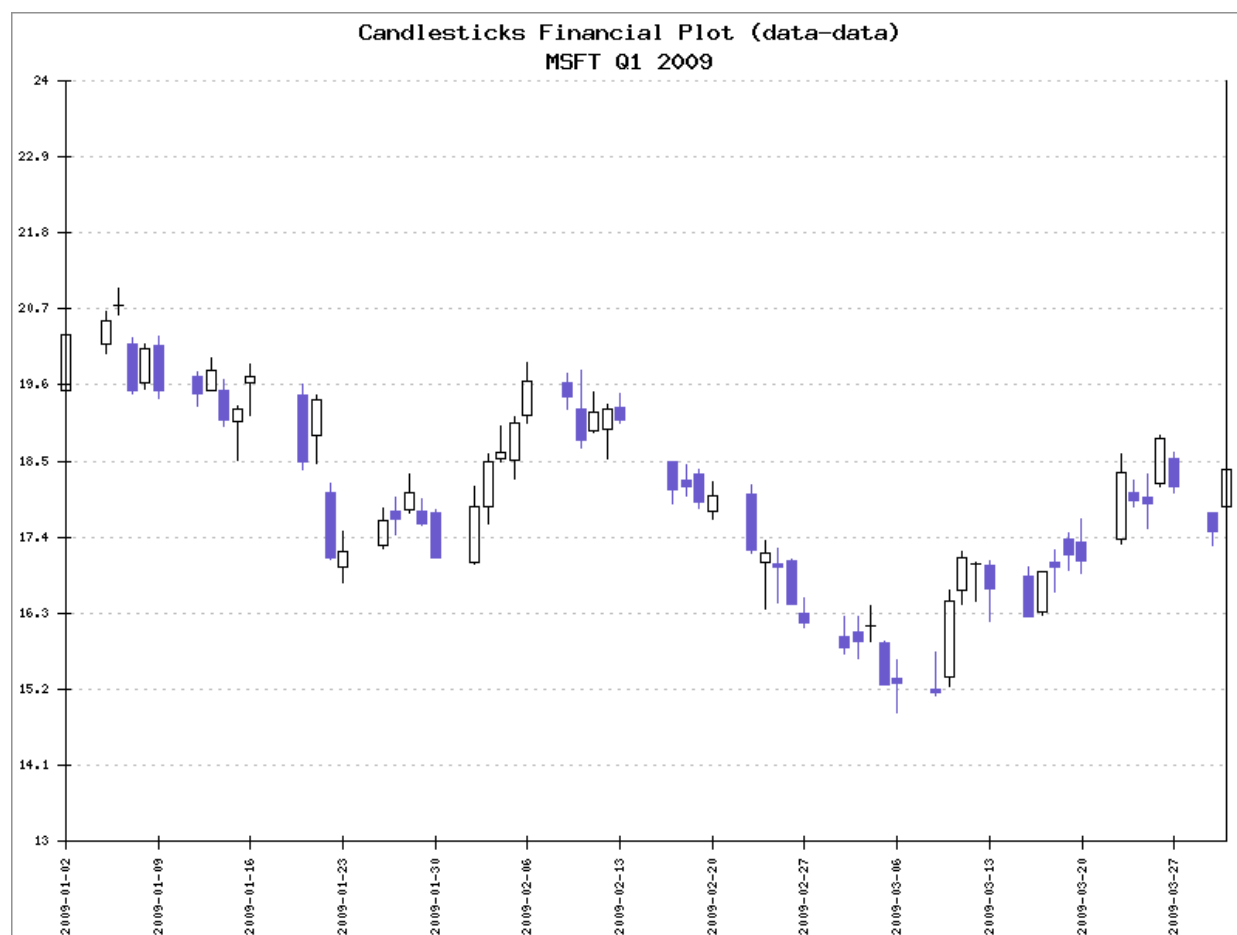

```
<?php
# PHPlot Example: OHLC (Financial) plot, Candlesticks plot, using
```

```
# external data file, data-data format with date-formatted labels.
define('DATAFILE', 'examples/ohlcdata.csv'); // External data file
require_once 'phplot.php';

/*
  Read historical price data from a CSV data downloaded from Yahoo! Finance.
  The first line is a header which must contain: Date,Open,High,Low,Close[...]
  Each additional line has a date (YYYY-MM-DD), then 4 price values.
  Convert to PHPlot data-data data array with empty labels and time_t X
  values and return the data array.
*/
function read_prices_data_data($filename)
{
    $f = fopen($filename, 'r');
    if (!$f) {
        fwrite(STDERR, "Failed to open data file: $filename\n");
        return FALSE;
    }
    // Read and check the file header.
    $row = fgetcsv($f);
    if ($row === FALSE || $row[0] != 'Date' || $row[1] != 'Open'
            || $row[2] != 'High' || $row[3] != 'Low' || $row[4] != 'Close') {
        fwrite(STDERR, "Incorrect header in: $filename\n");
        return FALSE;
    }
    // Read the rest of the file and convert.
    while ($d = fgetcsv($f)) {
        $data[] = array('', strtotime($d[0]), $d[1], $d[2], $d[3], $d[4]);
    }
    fclose($f);
    return $data;
}

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain'); // Improves presentation in the manual
$plot->SetTitle("Candlesticks Financial Plot (data-data)\nMSFT Q1 2009");
$plot->SetDataType('data-data');
$plot->SetDataValues(read_prices_data_data(DATAFILE));
$plot->SetPlotType('candlesticks');
$plot->SetDataColors(array('SlateBlue', 'black', 'SlateBlue', 'black'));
$plot->SetXLabelAngle(90);
$plot->SetXLabelType('time', '%Y-%m-%d');
$plot->SetXTickIncrement(7*24*60*60); // 1 week interval
$plot->DrawGraph();
```

5.32. Example - Filled Candlesticks OHLC (Open, High, Low, Close) Financial Plot

This example shows a candlesticks2 plot, which is a form of the Open, High, Low, Close (OHLC) financial plot. Each X is a point in time or interval, and there are 4 corresponding Y values for the four prices (open, high, low, and close). Compare this with Example 5.30, "Basic OHLC Plot" and Example 5.31, "Candlesticks OHLC Plot", which show the same data but with a different presentation. With the candlesticks2 plot type, all the candlestick bodies are filled in, meaning you must set meaningful data colors in order to be able to tell if a security closes up or down.

The data values for this example are read from an external file. Refer to Section 5.30, "Example - Basic OHLC (Open, High, Low, Close) Financial Plot" for more information. This example uses the data-data format, with the dates read from the file converted to X values in the data array.

Example 5.32. Filled Candlesticks OHLC Plot


```
<?php
# PHPlot Example: OHLC (Financial) plot, Filled Candlesticks plot, using
# external data file, data-data format with date-formatted labels.
define('DATAFILE', 'examples/ohlcdata.csv'); // External data file
require_once 'phplot.php';
```

```
/*
  Read historical price data from a CSV data downloaded from Yahoo! Finance.
  The first line is a header which must contain: Date,Open,High,Low,Close[...]
  Each additional line has a date (YYYY-MM-DD), then 4 price values.
  Convert to PHPlot data-data data array with empty labels and time_t X
  values and return the data array.
*/
function read_prices_data_data($filename)
{
    $f = fopen($filename, 'r');
    if (!$f) {
        fwrite(STDERR, "Failed to open data file: $filename\n");
        return FALSE;
    }
    // Read and check the file header.
    $row = fgetcsv($f);
    if ($row === FALSE || $row[0] != 'Date' || $row[1] != 'Open'
            || $row[2] != 'High' || $row[3] != 'Low' || $row[4] != 'Close') {
        fwrite(STDERR, "Incorrect header in: $filename\n");
        return FALSE;
    }
    // Read the rest of the file and convert.
    while ($d = fgetcsv($f)) {
        $data[] = array('', strtotime($d[0]), $d[1], $d[2], $d[3], $d[4]);
    }
    fclose($f);
    return $data;
}

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain'); // Improves presentation in the manual
$plot->SetTitle("Filled Candlesticks Financial Plot (data-data)\nMSFT Q1 2009");
$plot->SetDataType('data-data');
$plot->SetDataValues(read_prices_data_data(DATAFILE));
$plot->SetPlotType('candlesticks2');
$plot->SetDataColors(array('red', 'DarkGreen', 'red', 'DarkGreen'));
$plot->SetXLabelAngle(90);
$plot->SetXLabelType('time', '%Y-%m-%d');
$plot->SetXTickIncrement(7*24*60*60); // 1 week interval
$plot->DrawGraph();
```

5.33. Example - Linepoints Plot with Data Value Labels

This example shows a linepoints plot with data value labels. These are text strings which show the Y value above each point. (This feature was implemented for lines, points, and linepoints plots in PHPlot-5.3.0.) Because the Y values are shown with the data value labels, we have chosen to turn off the Y axis ticks and tick labels, which would be somewhat redundant. This example also has data label lines (see SetDrawXDataLabelLines), which are the lines drawn from the X axis up to the data points, to help associate the point with the X axis label.

Example 5.33. Linepoints Plot with Data Value Labels
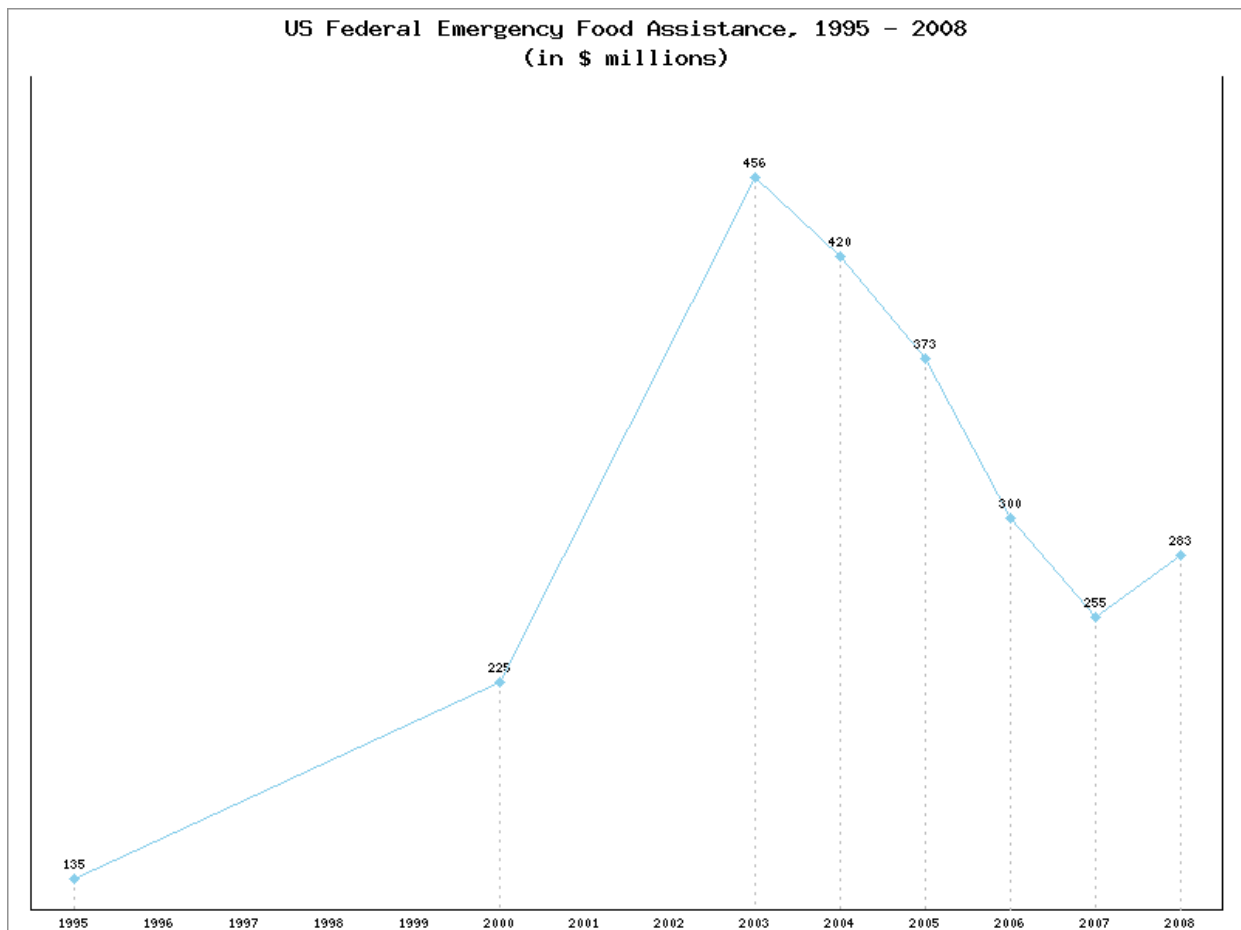

```php
<?php
# PHPlot Example: Linepoints plot with Data Value Labels
require_once 'phplot.php';

$data = array(
  array('1995', 135),
  array('1996', ''), // Missing data point
  array('1997', ''),
  array('1998', ''),
  array('1999', ''),
```

```
  array('2000', 225),
  array('2001', ''),
  array('2002', ''),
  array('2003', 456),
  array('2004', 420),
  array('2005', 373),
  array('2006', 300),
  array('2007', 255),
  array('2008', 283),
);

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain'); // Improves presentation in the manual
$plot->SetPlotType('linepoints');
$plot->SetDataType('text-data');
$plot->SetDataValues($data);
$plot->SetTitle("US Federal Emergency Food Assistance, 1995 - 2008\n"
                . "(in $ millions)");

# Turn on Y data labels:
$plot->SetYDataLabelPos('plotin');

# Turn on X data label lines (drawn from X axis up to data point):
$plot->SetDrawXDataLabelLines(True);

# With Y data labels, we don't need Y ticks, Y tick labels, or grid lines.
$plot->SetYTickLabelPos('none');
$plot->SetYTickPos('none');
$plot->SetDrawYGrid(False);
# X tick marks are meaningless with this data:
$plot->SetXTickPos('none');
$plot->SetXTickLabelPos('none');

$plot->DrawGraph();
```

5.34. Example - Overlaying Plots

This example shows overlay plots, where multiple plots are drawn at the same position on the same image. In this case, one plot contains a stacked bar plot, and the second is a linepoints plot. The two plots also have different Y axis scales.

See Section 4.6, "Multiple Plots Per Image" for more information.

Example 5.34. Overlaying Plots
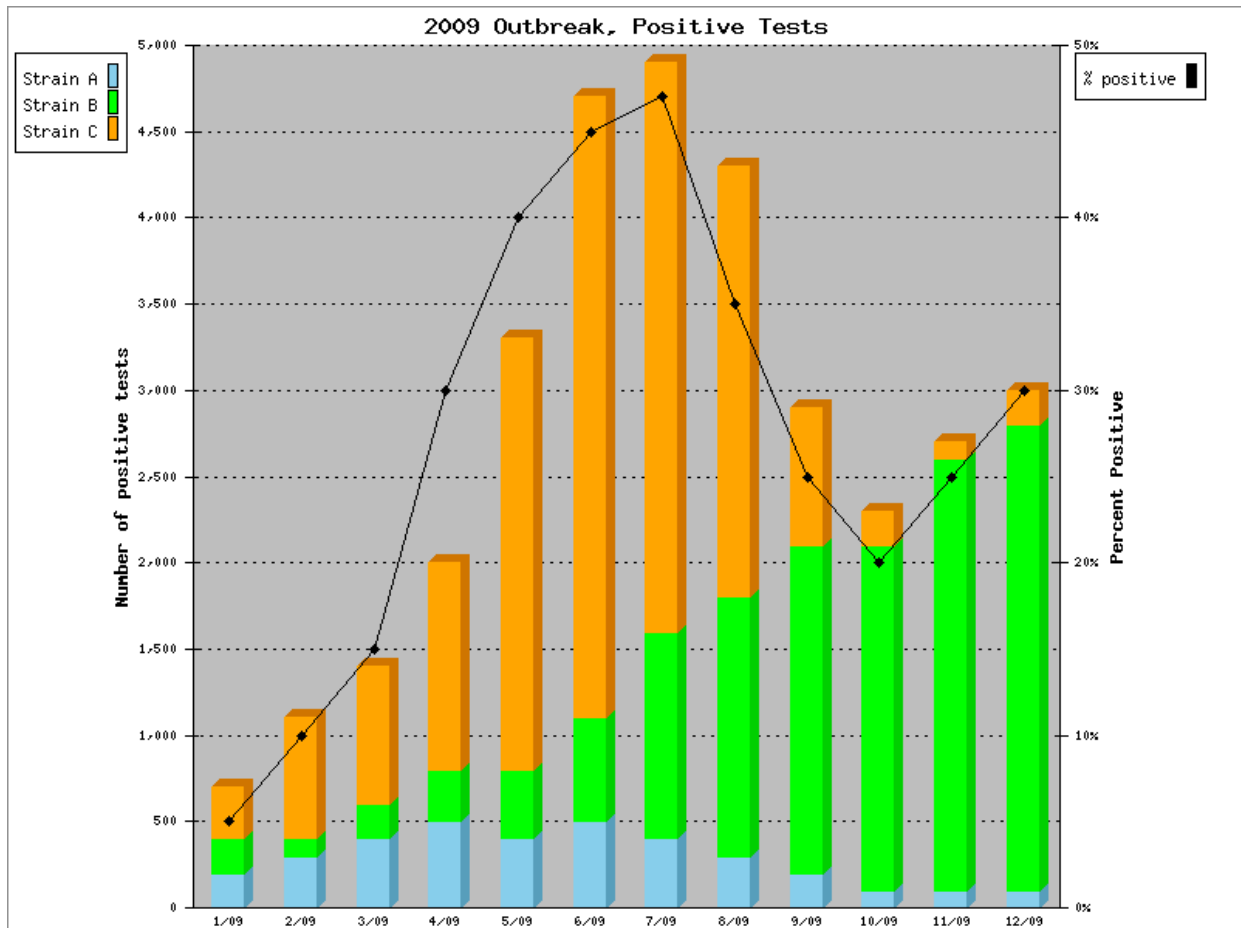

```
<?php
# PHPlot Example: Plot Overlay (lines and stackedbars)
require_once 'phplot.php';

$title = '2009 Outbreak, Positive Tests';

# Note: Graph is based on the real thing, but the data is invented.
# Data for plot #1: stackedbars:
$y_title1 = 'Number of positive tests';
$data1 = array(
    array('1/09',  200,  200,  300),
    array('2/09',  300,  100,  700),
    array('3/09',  400,  200,  800),
    array('4/09',  500,  300, 1200),
```

```
    array('5/09',  400,  400, 2500),
    array('6/09',  500,  600, 3600),
    array('7/09',  400, 1200, 3300),
    array('8/09',  300, 1500, 2500),
    array('9/09',  200, 1900,  800),
    array('10/09', 100, 2000,  200),
    array('11/09', 100, 2500,  100),
    array('12/09', 100, 2700,  200),
);
$legend1 = array('Strain A', 'Strain B', 'Strain C');

# Data for plot #2: linepoints:
$y_title2 = 'Percent Positive';
$data2 = array(
    array('1/09',   5),
    array('2/09',  10),
    array('3/09',  15),
    array('4/09',  30),
    array('5/09',  40),
    array('6/09',  45),
    array('7/09',  47),
    array('8/09',  35),
    array('9/09',  25),
    array('10/09', 20),
    array('11/09', 25),
    array('12/09', 30),
);
$legend2 = array('% positive');

$plot = new PHPlot(800, 600);
$plot->SetImageBorderType('plain'); // For presentation in the manual
$plot->SetPrintImage(False); // Defer output until the end
$plot->SetTitle($title);
$plot->SetPlotBgColor('gray');
$plot->SetLightGridColor('black'); // So grid stands out from background

# Plot 1
$plot->SetDrawPlotAreaBackground(True);
$plot->SetPlotType('stackedbars');
$plot->SetDataType('text-data');
$plot->SetDataValues($data1);
$plot->SetYTitle($y_title1);
# Set and position legend #1:
$plot->SetLegend($legend1);
$plot->SetLegendPixels(5, 30);
# Set margins to leave room for plot 2 Y title on the right.
$plot->SetMarginsPixels(120, 120);
# Specify Y range of these data sets:
$plot->SetPlotAreaWorld(NULL, 0, NULL, 5000);
$plot->SetYTickIncrement(500);
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');
# Format Y tick labels as integers, with thousands separator:
$plot->SetYLabelType('data', 0);
$plot->DrawGraph();

# Plot 2
$plot->SetDrawPlotAreaBackground(False); // Cancel background
$plot->SetDrawYGrid(False); // Cancel grid, already drawn
$plot->SetPlotType('linepoints');
```

```
$plot->SetDataValues($data2);
# Set Y title for plot #2 and position it on the right side:
$plot->SetYTitle($y_title2, 'plotright');
# Set and position legend #2:
$plot->SetLegend($legend2);
$plot->SetLegendPixels(690, 30);
# Specify Y range of this data set:
$plot->SetPlotAreaWorld(NULL, 0, NULL, 50);
$plot->SetYTickIncrement(10);
$plot->SetYTickPos('plotright');
$plot->SetYTickLabelPos('plotright');
$plot->SetDataColors('black');
# Format Y tick labels as integers with trailing percent sign:
$plot->SetYLabelType('data', 0, '', '%');
$plot->DrawGraph();

# Now output the graph with both plots:
$plot->PrintImage();
```

Chapter 6. PHPlot Functions By Category

This chapter presents a grouping of the PHPlot functions by category. Descriptions of the functions can be found in [PHPlot Function Reference](#).

6.1. Core

The functions in this section form the core of PHPlot. You cannot produce plots without using these. If you want to produce plots with little or no customization, you need use no other functions. These functions are used to create a PHPlot object, select the type of plot you want, provide the data to be plotted, and output the plot. For more information, see [Section 3.2, "Programming Overview"](#).

- [PHPlot](#)

- [PHPlot_truecolor](#)

- [DrawGraph](#)

- [PrintImage](#)

- [SetDataType](#)

- [SetDataValues](#)

- [SetPlotType](#)

6.2. Input/Output Control

The functions in this section control the overall input and output of the plot. Use the functions in this section to select the graphics file format, direct the output, and control browser caching.

- [SetBrowserCache](#)

- [SetFileFormat](#)

- [SetIsInline](#)

- [SetOutputFile](#)

- [SetPrintImage](#)

6.3. Colors and Line Styles

The functions in this section are used to control the colors and line styles of PHPlot elements. More information about using colors in PHPlot can be found in [Section 3.5, "Colors"](#) and [Section 4.2, "Truecolor Images"](#).

- [SetBackgroundColor](#)

- [SetDataBorderColors](#)

- SetDataColors

- SetDefaultDashedStyle

- SetErrorBarColors

- SetGridColor

- SetImageBorderColor

- SetImageBorderWidth

- SetLightGridColor

- SetLineStyles

- SetLineWidths

- SetPlotBgColor

- SetRGBArray

- SetTextColor

- SetTickColor

- SetTitleColor

- SetTransparentColor

- SetXTitleColor

- SetYTitleColor

6.4. Additional Style Controls

The functions in this section control additional style attributes of the plot, including background images, borders, point style and size for point plots, and shading for 3D affects.

- SetBgImage

- SetDrawBrokenLines

- SetDrawPlotAreaBackground

- SetImageBorderType

- SetPointShapes

- SetPointSizes

- SetPlotBorderType

- SetPlotAreaBgImage

- SetShading

6.5. Error Bar Controls

The functions in this section control the appearance of error bars, used in plots with plot data type `data-data-error`.

See also [SetErrorBarColors](), which sets the color used for error bars.

- [SetErrorBarLineWidth]()

- [SetErrorBarShape]()

- [SetErrorBarSize]()

6.6. Text Fonts

The functions in this section control the text fonts used by PHPlot. For more information, see [Section 3.8, "Text Fonts"]().

- [SetDefaultTTFont]()

- [SetFont]()

- [SetFontGD]()

- [SetFontTTF]()

- [SetLineSpacing]()

- [SetTTFPath]()

- [SetUseTTF]()

6.7. Titles

The functions in this section set the title strings: the overall plot title, and optional titles for the X and Y axes. For more information, see [Section 3.7.1, "Titles"](). See also [SetTitleColor](), [SetXTitleColor](), and [SetYTitleColor]() to set the title colors, and [SetFont]() to set the title fonts.

- [SetTitle]()

- [SetXTitle]()

- [SetYTitle]()

6.8. Legend

The functions in this section control the legend. For more information, see [Section 3.7.2, "Legend"](). See also [SetTextColor]() to set the legend text color, [SetGridColor]() to set the legend border color, and [SetFont]() to set the legend text font.

- [SetLegend]()

- [SetLegendPixels]()

- [SetLegendStyle]()

- SetLegendWorld

6.9. Axis Controls

The functions in this section control aspects of the X and Y axes. You can control the axis position and optionally use logarithmic scaling. You can also prevent drawing the axis lines.

- SetDrawXAxis

- SetDrawYAxis

- SetXAxisPosition

- SetXScaleType

- SetYAxisPosition

- SetYScaleType

6.10. Grid Controls

The functions in this section control the optional grid lines. For more information, see Section 3.7.3, "Grid Lines". See also SetLightGridColor to set the grid color.

- SetDrawDashedGrid

- SetDrawXGrid

- SetDrawYGrid

6.11. Labels

The functions in this section control the formatting and display of labels. This includes data labels (axis data labels which come from your data array, and data value labels), tick labels (automatically generated by PHPlot), and the special data labels used on pie charts to show the percentage of each pie slice. For more information, see Section 3.6, "Labels".

- SetDrawXDataLabelLines

- SetLabelScalePosition

- SetNumberFormat

- SetPrecisionX

- SetPrecisionY

- SetXDataLabelAngle

- SetXDataLabelPos

- SetXDataLabelType

- SetXLabelAngle

- [SetXLabelType](#)

- [SetXTickLabelPos](#)

- [SetXTimeFormat](#)

- [SetYDataLabelAngle](#)

- [SetYDataLabelPos](#)

- [SetYDataLabelType](#)

- [SetYLabelAngle](#)

- [SetYLabelType](#)

- [SetYTickLabelPos](#)

- [SetYTimeFormat](#)

6.12. Ticks

The functions in this section control the X axis and Y axis tick marks. For more information, see [Section 3.7.4, "Tick Marks"](#).

- [SetNumXTicks](#)

- [SetNumYTicks](#)

- [SetSkipBottomTick](#)

- [SetSkipLeftTick](#)

- [SetSkipRightTick](#)

- [SetSkipTopTick](#)

- [SetXTickCrossing](#)

- [SetXTickIncrement](#)

- [SetXTickLength](#)

- [SetXTickPos](#)

- [SetYTickCrossing](#)

- [SetYTickIncrement](#)

- [SetYTickLength](#)

- [SetYTickPos](#)

6.13. Scaling and Translation

The functions in this section control the overall data scaling and positioning of the plot on the image, or translate coordinates.

- [GetDeviceXY](#)

- [SetMarginsPixels](#)

- [SetPlotAreaPixels](#)

- [SetPlotAreaWorld](#)

6.14. Callbacks

The functions in this section are used to implement callbacks. This is an advanced feature described in more detail in [Section 4.3, "Callbacks"](#).

- [GetCallback](#)

- [RemoveCallback](#)

- [SetCallback](#)

PHPlot Function Reference

This part of the PHPlot Reference Manual contains the reference information for the PHPlot functions. Note that all the functions (except the class constructor) are implemented as methods of the class PHPlot, and are therefore called through an object which is an instance of the class. In this text, `$plot` is used to represent an instance of the PHPlot class.

DrawGraph

DrawGraph — Draw the current graph onto the image

Synopsis

```
$plot->DrawGraph()
```

Description

`DrawGraph` actually draws the current graph onto the image. That is, until DrawGraph is used, nothing happens except the recording of settings and data. DrawGraph also outputs the image with [PrintImage](), unless [SetPrintImage]() was used.

Parameters

None

GetCallback

GetCallback — Returns the current callback function registered for the given reason

Synopsis

```
$plot->GetCallback($reason)
```

Description

GetCallback returns the current callback function registered for the given reason. That is, it returns the function argument value used when a callback function was registered with SetCallback.

Parameters

$reason
>   A PHPlot-defined name for the callback. See Section 4.3.3, "Available Callbacks".

Return Value

Returns the function name as a string, or a 2-element array for object instance and method calls. Returns False if there is no callback registered for this reason, or if the given reason is not valid.

Notes

Since no valid function name (or array of object instance and method) evaluates to false when directly tested, it is not necessary to check using the identical-to (===) operator.

Callbacks are an experimental feature added to PHPlot-5.0.4. Refer to Section 4.3, "Callbacks" for more information.

GetDeviceXY

GetDeviceXY — Translate world coordinates into device coordinates

Synopsis

```
list($x, $y) = $plot->GetDeviceXY($x_world, $y_world)
```

Description

GetDeviceXY translates values in world coordinates into values in device coordinates. This is useful if you want to annotate a plot with text or graphics positioned relative to specific data values. Given the coordinates of a point in the coordinate space of your data values, this function returns the pixel coordinates of that point in the image.

Parameters

$x_world
   The X coordinate to translate from world coordinates.

$y_world
   The Y coordinate to translate from world coordinates.

Return Value

Returns an array of two values ($x, $y) in device coordinates which correspond to the world coordinate parameters.

Notes

This function only works after scaling factors have been established, which happens in DrawGraph. So it can only be used in two cases:

- From a drawing callback (see Section 4.3, "Callbacks") - that is, a callback whose name starts with 'draw'.

- If SetPrintImage(False) is used to disable automatic output of the image file, then GetDeviceXY() can be used after DrawGraph returns.

GetDeviceXY() will fail with an error message if it is called before scaling is set up.

If the world coordinates represent a point that is not visible on the plot, the returned device coordinates will be outside the plot area, or even outside the image area.

To see how this can be used in callbacks, see Section 4.3.5, "Using Callbacks to Annotate Plots".

History

This function was added in PHPlot-5.1.0.

PHPlot

PHPlot — Construct a new PHPlot Class Object

Synopsis

```
$plot = new PHPlot([$width], [$height], [$output_file], [$input_file])
```

Description

This is the class constructor for PHPlot. It creates a new plot object and initializes all internal settings to default values.

Parameters

$width
    Optional width of the plot image, in pixels. Default is 600.

$height
    Optional height of the plot image, in pixels. Default is 400.

$output_file
    Optional name of a file where the image output will be written. This is the same as using [SetOutputFile](#). Default is no output file, meaning the image is written to standard output (that is, sent back to the browser).

$input_file
    Optional name of a file to use as a starting image. This becomes the background for the plot. If an input_file is given, any width and height given to the constructor are ignored, and the size of the image in the named input_file are the plot image size. Default is no input file, meaning a blank image will be created at the given or default width and height.

Return Value

Returns an object, an instance of the PHPlot class.

Notes

The output_file will be ignored unless [SetIsInline](#)(True) is called.

If no input_file is supplied, the PHPlot constructor creates a [palette](#) plot image, and the [PHPlot_truecolor](#) constructor creates a [truecolor](#) plot image. If an input_file is supplied, the two constructors are equivalent, and the type of the input file (truecolor or palette) determines the type of the plot image.

History

Earlier versions of this manual said that the created object should always be returned as a reference, like this:

```
$plot =& new PHPlot(...);  // Do not use this
```

This was because PHPlot included a function to deallocate memory used by the object at script shutdown, but that would only work if a reference assignment was used. This quasi-destructor was removed at PHPlot-5.0.4 because it

interfered with memory deallocation until the script ended. So the reference assignment should not be used. In addition, reference assignment of a newly created object instance is deprecated starting with PHP5.

PHPlot_truecolor

PHPlot_truecolor — Construct a new PHPlot Truecolor Class Object

Synopsis

```
$plot = new PHPlot_truecolor([$width], [$height], [$output_file], [$input_file])
```

Description

This is the constructor for the PHPlot_truecolor class, which is an extended class that inherits from the PHPlot class. Like the PHPlot class, it creates a new plot object and initializes all internal settings to default values, but the resulting image will be a truecolor image, rather than a palette image. (See the notes below regarding an exception to this rule.)

Parameters

$width
Optional width of the plot image, in pixels. Default is 600.

$height
Optional height of the plot image, in pixels. Default is 400.

$output_file
Optional name of a file where the image output will be written. This is the same as using SetOutputFile. Default is no output file, meaning the image is written to standard output (that is, sent back to the browser).

$input_file
Optional name of a file to use as a starting image. This becomes the background for the plot. If an input_file is given, any width and height given to the constructor are ignored, and the size of the image in the named input_file are the plot image size. Default is no input file, meaning a blank image will be created at the given or default width and height.

Return Value

Returns an object, an instance of the PHPlot_truecolor class, which inherits all the functions (methods) of the PHPlot class.

Notes

Refer to PHPlot for the base class constructor. Refer to Section 4.2, "Truecolor Images" for more information on truecolor images.

The output_file will be ignored unless SetIsInline(True) is called.

If no input_file is supplied, the PHPlot_truecolor constructor creates a truecolor plot image, and the PHPlot constructor creates a palette plot image. If an input_file is supplied, the two constructors are equivalent, and the type of the input file (truecolor or palette) determines the type of the plot image.

The type of the plot image (truecolor or palette) might not be the same as the type of output file or stream which is generated by PHPlot. For example, a truecolor image is converted to palette if the output format (as set with SetFileFormat) is GIF, which supports only palette images. A palette image is converted to truecolor if the output

format is JPEG, which supports only truecolor images. More information can be found in <u>Section 4.2.5, "Image Formats and File Formats, Palette and Truecolor"</u>.

History

The PHPlot_truecolor class and its constructor were added in PHPlot-5.1.1. For that release only, there was a greater dependency between which constructor was used and which features were available, but this was removed in the next release.

PrintImage

PrintImage — Output the generated graph image and clean up the internal storage space.

Synopsis

```
$plot->PrintImage()
```

Description

`PrintImage` outputs the generated graph image and cleans up the internal storage space used. Output goes to the browser by default, or to the output file set by SetOutputFile.

Using PrintImage is not normally needed, since DrawGraph calls PrintImage unless it was told not to using SetPrintImage.

Parameters

None

RemoveCallback

RemoveCallback — Unregisters any callback registered for the given reason

Synopsis

```
$plot->RemoveCallback($reason)
```

Description

RemoveCallback unregisters any callback registered for the given reason. It undoes the effect of SetCallback.

Parameters

$reason
    A PHPlot-defined name for the callback. See Section 4.3.3, "Available Callbacks".

Return Value

Returns True if the given reason is valid (whether or not there was actually a callback registered for it). Returns False if reason is not a valid callback reason.

Notes

Callbacks are an experimental feature added to PHPlot-5.0.4. Refer to Section 4.3, "Callbacks" for more information.

SetBackgroundColor

SetBackgroundColor — Sets the overall background color.

Synopsis

```
$plot->SetBackgroundColor($color)
```

Description

`SetBackgroundcolor` sets the overall background color. This is the color of the background of the whole image.

Parameters

$color
> Color value to use. See Section 3.5, "Colors" for more on color values.

Notes

The default background color is white. Background image overrides background color; background color is ignored if a background image was set with SetBgImage.

SetBgImage

SetBgImage — Set a graphic file to be used in the graph background

Synopsis

```
$plot->SetBgImage($input_file, [$mode])
```

Description

`SetBgImage` sets an image file to be used as the graph background. The image can be scaled or tiled to fit.

Parameters

$input_file
    Path to the file to be used. The file can be any type allowed by GD, which usually includes JPEG, GIF, and PNG.

$mode
    Optional display mode for the background image: one of the strings 'centeredtile', 'tile', or 'scale'. The default is 'centeredtile'.

Notes

If a background image has been set, background color (set with [SetBackgroundColor](#)) is ignored.

Scale mode scales the supplied background image file to fit the image area of the entire graph. Tile and centeredtile modes repeat the supplied background image file as needed to fit the image area of the graph. The difference is that centeredtile offsets the start position within the background image by half its size, which works better for some images.

SetBgImage sets a background for the entire image area, while [SetPlotAreaBgImage](#) sets a background for the plot area (generally, the area between the axes). If both are used, the plot area background overlays that portion of the overall background.

If `$input_file` is `NULL`, no background image will be used.

SetBrowserCache

SetBrowserCache — Control browser-side image caching

Synopsis

```
$plot->SetBrowserCache($browser_cache)
```

Description

`SetBrowserCache` controls whether to allow the browser to cache the image generated by PHPlot. By default, PHPlot sends out HTTP headers to tell the browser not to cache the generated image, since it is assumed that the image is generated from dynamic data and a cached copy would not be accurate. You can use this function to allow the browser to cache the image.

Parameters

$browser_cache
    True to allow the browser to cache the image; False to not allow the browser to cache the image.

SetCallback

SetCallback — Registers a callback function

Synopsis

```
$plot->SetCallback($reason, $function, [$arg])
```

Description

SetCallback registers a callback function. That is, it arranges for the caller-provided function to be called at a specific point or points inside PHPlot's internal processing.

Parameters

$reason
> A PHPlot-defined name for the callback. See [Section 4.3.3, "Available Callbacks"](#).

$function
> The function to be called. This can be either the name of a function as a string, or a two-element array with an object class instance and method name. See [Section 4.3.1, "Callbacks Application Interface"](#) for more information.

$arg
> An optional opaque argument passed-through to the callback function when PHPlot triggers the callback. If not supplied, the callback function will get a NULL argument.

Return Value

Returns True if the callback has been registered. Returns False on error. The only error condition is if the given callback reason is not valid. Note that the function name is not validated until the callback is triggered.

Notes

If a callback is already registered for the given reason, the new callback replaces the old one.

Callbacks are an experimental feature added to PHPlot-5.0.4. Refer to [Section 4.3, "Callbacks"](#) for more information.

SetDataBorderColors

SetDataBorderColors — Set the rectangle border color on bars and stacked bars

Synopsis

```
$plot->SetDataBorderColors($border)
```

Description

`SetDataBorderColors` sets the colors used for the borders of the bars and stacked bars for plot types 'bars' and 'stackedbars'. The borders are only drawn if shading is turned off with SetShading; the default is to draw bars with a drop-shadow for a 3-D look and no borders.

Parameters

$border
> An array of color values, one for the bar border of each data set. Or, a single color value (not an array) to use for all data sets. For other possibilities, see Notes. See Section 3.5, "Colors" for more on color values.

Notes

If an array is used for *$border*, it must use zero based sequential integer indexes. This is what the PHP manual calls 'the usual integer indices (starting from zero, increasing by one)'.

If this function is never called, and SetShading is called to turn shading off, a default color map is used which sets all data borders to black.

This is only used for plot types 'bars' and 'stackedbars', and only if shading is turned off.

If you want flat (unshaded), borderless bars, call SetShading(0) to turn off the shading, and call SetDataBorderColors with the same color (or color array) that you use with SetDataColors.

If *$border* is not an array, but a single color value, then that color will be used for all data sets. However, the array(r,g,b) notation is not allowed in this case (because it looks like an array of 3 separate color values). You can get around this restriction if you want to specify a single color as an R, G, B array by wrapping the array in another array, for example: `array(array(102, 0, 192))`.

Two special uses of the *$border* argument are available. If the argument is an empty string, or boolean False, the color map is reset to the defaults. This can be used to restore the default color map. If the argument is NULL or missing from the function call, the color map is reset to the defaults, but only if it has not already been set. This is used internal to PHPlot for one-time initialization.

A data colors callback, as described in Section 4.4, "Custom Data Color Selection", also controls selection of the color for data borders (if used).

SetDataColors

SetDataColors — Set the colors for plotting data sets

Synopsis

```
$plot->SetDataColors($data_colors, [$border], [$default_alpha])
```

Description

`SetDataColors` sets the colors used for plotting the data.

Parameters

$data_colors
> An array of color values, one for each data set. Or, a single color value (not an array) to use for all data sets. For other possibilities, see Notes. See [Section 3.5, "Colors"](#) for more on color values.

$border
> Argument provided for backward compatibility. Use [SetDataBorderColors](#) instead.

$default_alpha
> A default alpha value to apply to all data colors which do not have an alpha value. This is generally useful only with Truecolor images. A value of zero means opaque, and 127 means fully transparent. See [Section 4.2, "Truecolor Images"](#) for more information.

Notes

If an array is used for *$data_colors*, it must use zero based sequential integer indexes. This is what the PHP manual calls 'the usual integer indices (starting from zero, increasing by one)'.

Usually the *$data_colors* argument is an array of colors, one for each data set to be plotted. For example:

```
$plot->SetDataColors(array('red', 'green', 'blue'));
$plot->SetDataType('data-data');
$plot->SetDataValues(array( array('', 1, 4, 10, 5),
                            array('', 2, 6, 20, 3)));
```

This will plot a red line from (1,4) to (2,6), a green line from (1,10) to (2,20), and a blue line from (1,5) to (2,3).

If *$data_colors* is not an array, but a single color value, then that color will be used for all data sets. However, the array(r,g,b) notation is not allowed in this case (because it looks like an array of 3 separate color values). You can get around this restriction if you want to specify a single color as an R, G, B array by wrapping the array in another array, for example: `array(array(102, 0, 192))`.

Two special uses of the *$data_colors* argument are available. If the argument is an empty string, or boolean False, the color map is reset to the defaults. This can be used to restore the default color map. If the argument is NULL or missing from the function call, the color map is reset to the defaults, but only if it has not already been set. This is used internal to PHPlot for one-time initialization.

If SetDataColors is never called, a default color map is used which contains 16 colors starting with SkyBlue, green, orange, and blue. For the full list, see [Section 3.5.3, "Plotting Colors"](#). By default, all colors are opaque (alpha=0).

You can keep the default color map but set all colors in it to use a transparency (alpha) value like this:

```
$plot->SetDataColors(NULL, NULL, 60);
```

This applies alpha=60 (meaning 60/127 transparency) to all the default data colors.

You can control how the data colors array is used with a data colors callback. See Section 4.4, "Custom Data Color Selection" for more information.

History

The optional $default\_alpha$ argument was added in PHPlot-5.1.1 when truecolor images were implemented.

Through PHPlot-5.0.7, the default color map contained these 8 colors: SkyBlue, green, orange, blue, orange, red, violet, and azure1. These were used if SetDataColors was never called. Unfortunately, orange is used twice, and azure1 is so close to the white background that it is invisible. Also, through PHPlot-5.0.7, if SetDataColors was called with an empty string argument, the color map was set to these 4 colors: blue red green orange. Starting with PHPlot-5.1.0, a new default color map with 16 colors was defined. Given an empty string (or False), SetDataColors now restores the default color map.

SetDataType

SetDataType — Indicate the format of the data array

Synopsis

```
$plot->SetDataType($dt)
```

Description

SetDataType tells PHPlot how to interpret the data array set with SetDataValues. More information on data types can be found in Section 3.3, "PHPlot Data Types".

Parameters

$dt
> The data array format type, which must be one of the following values:

Format	Description
text-data	Each data array entry is an array with a label, followed by one or more Y values.
data-data	Each data array entry is an array with a label, an X value, then one or more Y values.
data-data-error	Each data array entry is an array with a label, an X value, then one or more triplets of Y value, error in the positive directory, and error in the negative direction.
text-data-single	This is used only for pie charts. Each data array entry is an array with a label and a single data value.
text-data-yx	Each data array entry is an array with a label, followed by one or more X values. The Y value is implied. This is used for horizontal plots.
data-data-yx	Each data array entry is an array with a label, a Y value, then one or more X values. This is used for horizontal plots.

Notes

The default data type is text-data.

An example of a text-data data array is:

```
$data = array(  array('Jan', 100, 150, 200),
                array('Feb', 110, 140, 210),
                array('Mar', 120, 145, 200),
                array('Apr', 110, 160, 220) );
```

This defines the data for 3 plots with 4 points on each, and a month name as label for each point. This is also a valid data array for data type text-data-yx.

An example of a `data-data` data array is:

```
$data = array(  array('', 2, 15),
                array('', 4, 14),
                array('', 6, 10),
                array('', 8, 20) );
```

Here the labels are empty strings, next are the X values, then a single set of Y values (1 plot). This is also a valid data array for data type `data-data-yx`, with each row containing a Y value, followed by a single X value.

An example of a `data-data-error` data array is:

```
$data = array(  array('1999', 1, 23.5, 5, 3),
                array('2000', 2, 20.1, 4, 4),
                array('2001', 3, 19.1, 3, 4),
                array('2002', 4, 16.8, 4, 3) );
```

Here the labels are years, next are the X values 1-4, then a single set of Y values with error ranges between 3 and 5 for each point.

An example of a `text-data-single` data array, used only for pie charts, is:

```
$data = array(  array('', 10),
                array('', 40),
                array('', 50) );
```

Here the labels are empty (they aren't used with a pie chart anyway), and 3 slices with relative weights of 10, 40, and 50 are defined.

History

Data type `text-data-yx` for horizontal plots was added in PHPlot-5.1.2. Data type `data-data-yx` for horizontal plots was added in PHPlot-5.1.3.

SetDataValues

SetDataValues — Set the data array for plotting

Synopsis

```
$plot->SetDataValues($dv)
```

Description

SetDataValues sets the data array which contains the data values to be plotted. Use of this function is required.

Parameters

$dv
    The data array, containing values according to the data type format set by SetDataType.

Notes

The data array *$dv* must use zero based sequential integer indexes. This is what the PHP manual calls 'the usual integer indices (starting from zero, increasing by one)'. Each entry in the data array is also an array, representing one 'record'. The record arrays need not use zero based sequential integer indexes; the entries are processed in the same order in which they were defined, regardless of the index values.

SetDefaultDashedStyle

SetDefaultDashedStyle — Sets the on/off pattern for dashed lines.

Synopsis

```
$plot->SetDefaultDashedStyle($style)
```

Description

SetDefaultDashedStyle sets the line style for dashed lines. That is, it customizes the look of dashed lines by specifying the dash and gap lengths.

Parameters

$style

A string specifying the number of alternating colored and transparent dots, in order. For example, '4-3' means 4 colored, 3 transparent; '2-3-1-2' means 2 colored, 3 transparent, 1 colored, 2 transparent.

Notes

The default dashed style is '2-4', meaning 2 pixels drawn, followed by a gap of 4 pixels.

All dashed lines on a plot must use the same dashed style. Dashed lines are used for the grid (unless disabled with SetDrawDashedGrid), and for any data sets with line style set to dashed with SetLineStyles.

SetDefaultTTFont

SetDefaultTTFont — Set the default TrueType font

Synopsis

```
$plot->SetDefaultTTFont($font)
```

Description

`SetDefaultTTFont` sets the default TrueType font, resets all text elements to use that font, and makes TrueType fonts the default font type.

Parameters

$font
> Name of the TrueType font file to use as default. Specify either a filename in the default TrueType font directory (or one that can be found by GD using its own rules), or the full pathname to a font file.

Notes

This function selects TrueType fonts as the default font type as if [SetUseTTF](True) was called.

The supplied font name is first checked as given. If that does not work, it is checked prefixed with the default TrueType font directory as set with [SetTTFPath]. If that does not work either, a fatal error results. See [Section 3.8.2, "TrueType Font Selection"] for more information.

This function resets all elements to use the named font and default sizes, so it undoes all prior [SetFont], [SetFontGD], and [SetFontTTF] calls.

Change the font used by individual text elements with [SetFont], [SetFontGD], and [SetFontTTF] after using SetDefaultTTFont.

History

Starting with PHPlot-5.1.3, TrueType fonts are validated using GD. This allows GD to apply its own rules to try to locate a font file. On at least some platforms, this allows fonts to be specified by filename only, without having to set the PHPlot default font directory. (Through PHPlot-5.1.2, the existence of the font file was checked, which did not allow GD to try to find the font using its own rules.)

Through PHPlot-5.0.5, setting a default TrueType font with this function also forced all text on the graph to use TrueType text. Starting with PHPlot-5.0.6, it just sets the default font type. Set [SetUseTTF] for more information on this change.

This behavior of this function was changed significantly at PHPlot-5.0rc3.

SetDrawBrokenLines

SetDrawBrokenLines — Sets whether lines should be broken at missing data

Synopsis

```
$plot->SetDrawBrokenLines($bl)
```

Description

`SetDrawBrokenLines` determines how to plot lines with missing data points (that is, missing Y values). By default, PHPlot will act as if the point does not exist, connecting the points before and after the missing datum in the usual way. Use SetDrawBrokenLines to leave a gap between the points before and after missing data instead.

This only applies to 'lines' and 'squared' plot types.

Parameters

$bl
    True to break the lines at missing Y data points. False to connect the lines around missing data.

Notes

The default is to ignore missing data and connect lines around missing points.

A missing Y value can be represented in the data array by an empty string. (Anything non-numeric works.)

SetDrawDashedGrid

SetDrawDashedGrid — Use solid or dashed lines for the grid

Synopsis

```
$plot->SetDrawDashedGrid($ddg)
```

Description

`SetDrawDashedGrid` determines whether the grid will be drawn with solid or dashed lines. The default is to used dashed lines.

Parameters

$ddg
  True to use dashed lines, False to use solid lines.

SetDrawPlotAreaBackground

SetDrawPlotAreaBackground — Enables drawing of a plot area background color

Synopsis

```
$plot->SetDrawPlotAreaBackground($dpab)
```

Description

`SetDrawPlotAreaBackground` enables or disables drawing of a solid fill color behind the plot area (the area inside the axes, typically). By default, no plot area background color is used, which results in the overall image background color applying to the plot area.

Parameters

$dpab
   If True, draw the plot area background color. If False, ignore the plot area background color.

Notes

The actual color which will be drawn in the plot area background is set with [SetPlotBgColor](#).

Plot area background color is ignored if a plot area background image was set with [SetPlotAreaBgImage](#).

SetDrawXAxis

SetDrawXAxis — Enable or disable drawing of the X axis line

Synopsis

```
$plot->SetDrawXAxis($draw)
```

Description

`SetDrawXAxis` enables or disables drawing of the X axis line. Disabling the X axis line should be necessary only in special applications.

Parameters

$draw
    True to draw the X axis line, False to not draw the X axis line.

Notes

By default, the X axis line is drawn.

Disabling the X axis line does not disable associated plot elements, nor change the plot margin calculation. To produce a completely 'bare' plot, you must turn off the grid, tick marks, tick labels, and data labels. You also need to turn off the plot area border, which defaults to left and right sides. Lastly, you might want to reduce the margins, since the default minimum margin is 15 pixels on each side of the plot area, even if there are no axis lines, labels, or titles. Here is an example:

```
// 'Bare' plot partial code example
$plot->SetXTickPos('none');          // Turn off X tick marks
$plot->SetXTickLabelPos('none');     // Turn off X tick labels
$plot->SetXDataLabelPos('none');     // Turn off X data labels
$plot->SetYTickPos('none');          // Turn off Y tick marks
$plot->SetYTickLabelPos('none');     // Turn off Y tick labels
$plot->SetPlotBorderType('none');    // Turn off plot area border
$plot->SetDrawXGrid(False);          // Turn off X grid lines
$plot->SetDrawYGrid(False);          // Turn off Y grid lines
$plot->SetDrawXAxis(False);          // Don't draw X axis line
$plot->SetDrawYAxis(False);          // Don't draw Y axis line
$plot->SetMarginsPixels(2, 2, 2, 2); // Reduce plot margins to 2 pixels
```

See Section 5.24, "Example - Using Truecolor To Make a Histogram" for an example of a plot where axis lines are not desired. (The configuration data as shown in the example draws a full plot area border, so the axis lines would be covered anyway. But if the border is turned off, the axis lines would be visible unless they are suppressed as shown.)

History

This function was added in PHPlot-5.3.0.

SetDrawXDataLabelLines

SetDrawXDataLabelLines — Draw data lines

Synopsis

```
$plot->SetDrawXDataLabelLines($dxdl)
```

Description

`SetDrawXDataLabelLines` enables drawing of data label lines. Data label lines are vertical lines drawn from the X axis data label positions to the data points. In the usual case, with the X axis at the bottom of the plot and the data labels just below the X axis, the data label lines would be drawn up from the X axis to the data points. Depending on the data label locations set with [SetXDataLabelPos](), the lines would be drawn down or up (or both, or neither) from the data points to the bottom or top of the plot area.

To use data label lines, you generally want to turn off ticks, tick labels, and the X grid lines.

Parameters

$dxdl
    True to draw the data label lines, False to not draw the lines.

Notes

Data label lines only work with these plot types: lines, points, and linepoints.

If a graph contains multiple data sets, data label lines drawn down will start at the maximum Y value for each X value. Data label lines drawn up will start at the minimum Y value for each X value.

By default, data label lines are not drawn.

SetDrawXGrid

SetDrawXGrid — Whether or not to draw the X grid lines

Synopsis

```
$plot->SetDrawXGrid($dxg)
```

Description

`SetDrawXGrid` enables or disables the drawing of the X grid lines. (The X grid lines are the vertical lines which intersect the X axis and are parallel to the Y axis.) The default is to not draw the X grid for vertical plots, and to draw the X grid for horizontal plots.

Parameters

$dxg
    True to draw the X grid lines, False to not draw the X grid lines.

Notes

[SetLightGridColor](#) sets the color of the grid lines.

SetDrawYAxis

SetDrawYAxis — Enable or disable drawing of the Y axis line

Synopsis

```
$plot->SetDrawYAxis($draw)
```

Description

SetDrawYAxis enables or disables drawing of the Y axis line. Disabling the Y axis line should be necessary only in special applications.

Parameters

$draw
    True to draw the Y axis line, False to not draw the Y axis line.

Notes

By default, the Y axis line is drawn.

Disabling the Y axis line does not disable associated plot elements, nor change the plot margin calculation. To produce a completely 'bare' plot, you must turn off the grid, tick marks, tick labels, and data labels. You also need to turn off the plot area border, which defaults to left and right sides. Lastly, you might want to reduce the margins, since the default minimum margin is 15 pixels on each side of the plot area, even if there are no axis lines, labels, or titles. See SetDrawXAxis for sample code.

History

This function was added in PHPlot-5.3.0.

SetDrawYGrid

SetDrawYGrid — Whether or not to draw the Y grid lines

Synopsis

```
$plot->SetDrawYGrid($dyg)
```

Description

`SetDrawYGrid` enables or disables the drawing of the Y grid lines. (The Y grid lines are the horizontal lines which intersect the Y axis and are parallel to the X axis.) The default is to draw the Y grid for vertical plots, and not draw the Y grid for horizontal plots.

Parameters

$dyg
    True to draw the Y grid lines, False to not draw the Y grid lines.

Notes

SetLightGridColor sets the color of the grid lines.

SetErrorBarColors

SetErrorBarColors — Sets the colors used for data error bars

Synopsis

```
$plot->SetErrorBarColors($error_bar_colors)
```

Description

`SetErrorBarColors` sets the colors used for each data set's error bars, in the same way [SetDataColors](#) sets the colors used for the data plot itself.

Parameters

$error_bar_colors
> An array of color values, one for each data set's error bars. Or, a single color value (not an array) to use for all data set error bars. For other possibilities, see Notes. See [Section 3.5, "Colors"](#) for more on color values.

Notes

If an array is used for *$error_bar_colors*, it must use zero based sequential integer indexes. This is what the PHP manual calls 'the usual integer indices (starting from zero, increasing by one)'.

If *$error_bar_colors* is not an array, but a single color value, then that color will be used for all data sets' error bars. However, the array(r,g,b) notation is not allowed in this case (because it looks like an array of 3 separate color values). You can get around this restriction if you want to specify a single color as an R, G, B array by wrapping the array in another array, for example: `array(array(102, 0, 192))`.

Two special uses of the *$error_bar_colors* argument are available. If the argument is an empty string, or boolean False, the color map is reset to the defaults. This can be used to restore the default color map. If the argument is NULL or missing from the function call, the color map is reset to the defaults, but only if it has not already been set. This is used internal to PHPlot for one-time initialization.

If SetErrorBarColors is never called, the default color map is the same as for [SetDataColors](#). It contains 16 colors, starting with SkyBlue, green, orange, and blue. For the full list, see [Section 3.5.3, "Plotting Colors"](#).

If you change the data colors with [SetDataColors](#), you probably want to change the error bar colors to the same array.

A data colors callback, as described in [Section 4.4, "Custom Data Color Selection"](#), also controls selection of the color for error bars.

History

Through PHPlot-5.0.7, the default color map contained these 8 colors: SkyBlue, green, orange, blue, orange, red, violet, and azure1. These were used if SetErrorBarColors was never called. Also, through PHPlot-5.0.7, if SetErrorBarColors was called with an empty string argument, the color map was set to just contain 'black'. Starting with PHPlot-5.1.0, a new default color map with 16 colors was defined. Given an empty string (or False), SetErrorBarColors now restores the default color map.

SetErrorBarLineWidth

SetErrorBarLineWidth — Set the line width used for error bars

Synopsis

```
$plot->SetErrorBarLineWidth($seblw)
```

Description

`SetErrorBarLineWidth` sets the line width used for error bars. This width is used to draw the vertical lines indicating the error range in the positive and negative direction, and also the tees (if enabled) and the top and bottom.

Parameters

$seblw
    Desired width in pixels of the lines used to draw error bars.

Notes

The default is to use 1 pixel wide lines.

This is only used with `data-data-error` plot data type.

All lines in a plot use the same error bar width.

SetErrorBarShape

SetErrorBarShape — Select line or tee-style error bars

Synopsis

```
$plot->SetErrorBarShape($ebs)
```

Description

`SetErrorBarShape` selects the shape used for error indicators. Two shapes are available: 'tee' puts a short horizontal line at the top and bottom of each error bar, and 'line' does not.

Parameters

$ebs
    Error bar shape: either 'tee' or 'line'.

Notes

The default error bar shape is 'tee'.

This is only used with `data-data-error` plot data type.

All lines in a plot use the same error bar shape.

SetErrorBarSize

SetErrorBarSize — Set the size of the error bar tee.

Synopsis

```
$plot->SetErrorBarSize($ebs)
```

Description

SetErrorBarSize sets the length of the 'tee' drawn at the top and bottom of error bars, if the error bar shape is set to 'tee' with SetErrorBarShape (or defaulted to 'tee').

Parameters

$ebs
    Error bar tee length in pixels.

Notes

The default size is 5 pixels.

This is only used with data-data-error plot data type, and with error bar shape 'tee'.

All lines in a plot use the same error bar size.

SetFileFormat

SetFileFormat — Select the graphic image format generated by PHPlot

Synopsis

```
$plot->SetFileFormat($format)
```

Description

`SetFileFormat` selects a graphic image format from the available image formats. Depending on how PHP and/or GD were built on your system, available formats include JPEG, PNG, GIF, and WBMP.

Parameters

$format
    What graphic image format to use: 'jpg', 'png', 'gif', or 'wbmp'.

Notes

The default file format is 'png'.

Depending on how GD was built, not all of these formats will be available. You can use `phpinfo()` to see what formats are supported by your PHP/GD installation.

JPEG is generally a bad choice for this type of image, as the lossy compression reduces the quality of lines and text.

SetFont

SetFont — Select which font to use for a plot element

Synopsis

```
$plot->SetFont($elem, $font, [$size], [$line_spacing])
```

Description

`SetFont` selects the font and size to use for one plot element (for example, the title). This functions works differently depending on whether or not you are using TrueType fonts. If using TrueType fonts, call either [SetDefaultTTFont](#) or [SetUseTTF](#) before calling SetFont.

Parameters

$elem

The name of the element to change the font for. Use one of the following strings: 'title', 'legend', 'generic', 'x_label', 'y_label', 'x_title', or 'y_title'. (The 'generic' font is currently used for pie chart percentage labels and error message images.)

$font

Selects the font to use. For TrueType fonts, this is either the full pathname of a TrueType font filename, or the filename (without path) if the font file is either located in the default TrueType font directory set with [SetTTFPath](#) or can be found by GD using its default search rules. An empty string or NULL can be specified to use the default TrueType font.

For built-in GD fonts, this is a number between 1 and 5 which selects one of the built-in GD fonts. Font 1 is the smallest, and font 5 is the largest.

$size

The font size in points for TrueType fonts. Ignored for built-in GD fonts. If not specified, a default value of 12 is used. See note below.

$line_spacing

Optional line spacing adjustment for this text element. This is interpreted differently for GD and TrueType text. See [SetLineSpacing](#) for details. If not specified, the value set by [SetLineSpacing](#) is used.

Notes

When using built-in GD fonts, the default fonts are shown in the following table, where font 1 is the smallest font and font 5 is the biggest font.

Element	Default Built-in Font
generic	2
legend	2
title	5
x_label	1
y_label	1

Element	Default Built-in Font
x_title	3
y_title	3

When using TrueType fonts, the default font sizes are shown in the following table. Use SetDefaultTTFont to set the default TrueType font.

Element	Default TrueType Font Size (points)
generic	8
legend	8
title	14
x_label	6
y_label	6
x_title	10
y_title	10

Simultaneous use of GD and TrueType font text is allowed in the same plot. To mix font types, use SetFontGD and SetFontTTF to specify the font and font type of an element, instead of using SetFont.

SetFont implicitly uses the default font type. When a PHPlot object instance is created, the default font type is GD. Using SetUseTTF(True), or selecting a default font with SetDefaultTTFont, sets the default font type to TrueType. Using SetUseTTF(False) sets the default font type back to GD. Either of these three operations will also reset all current text elements to the defaults indicated above, negating any prior SetFont, SetFontGD, or SetFontTTF calls. Note that SetTTFPath, which selects the directory where TrueType fonts can be found, does not affect the default font type nor does it change any existing font selections.

Although PHP documents the TrueType font sizes as being given in points (where there are about 72 points per inch), it doesn't know the output device resolution, so it just assumes a fixed resolution of 72 pixels per inch. As a result, the TrueType font size argument actually measures the approximate font height in pixels. For example, if you use $size=18, the text will be about 18 pixels high in the user's browser. The actual size seen by the user will depend on the resolution of the user's display. On a 72 pixels per inch display, the text size will be 18 points, but at 96 pixels per inch it would only be 13.5 points.

History

Starting with PHPlot-5.1.3, TrueType fonts are validated by trying to use the font with a non-drawing operation, rather than by seeing if the font file exists. See Section 3.8.2, "TrueType Font Selection" for more information.

Simultaneous use of GD and TrueType font text was added at PHPlot-5.0.6. Through PHPlot-5.0.5, all text in a plot used GD fonts, or all text used TrueType fonts.

The line_spacing parameter was added at PHPlot-5.0.6 to allow finer control over the line spacing for different elements. Through PHPlot-5.0.5, the same line spacing was used for all text elements.

The described behavior for finding TrueType font files (first using the name as given, then looking in the SetTTFPath font directory) was implemented in PHPlot-5.0rc3.

SetFontGD

SetFontGD — Select a GD font to use for a plot element

Synopsis

```
$plot->SetFontGD($elem, $font, [$line_spacing])
```

Description

`SetFontGD` selects a GD font to use for one plot element (for example, the title). This function supplements SetFont, which selects a GD or TTF font depending on the currently selected font type. SetFontGD always selects a GD font, even if TrueType fonts are in use.

Parameters

$elem
    The name of the element to change the font for. Use one of the following strings: 'title', 'legend', 'generic', 'x_label', 'y_label', 'x_title', or 'y_title'. (The 'generic' font is currently used for pie chart percentage labels and error message images.)

$font
    Selects the GD font to use. This is a number between 1 and 5 which selects one of the built-in GD fonts. Font 1 is the smallest, and font 5 is the largest.

$line_spacing
    Optional line spacing adjustment for this text element. For GD fonts, this is the number of pixels between lines. If not specified, the value set by SetLineSpacing is used.

Notes

See SetFont for more information about fonts. See SetLineSpacing for more information about line spacing.

Use SetUseTTF (or SetDefaultTTFont, which calls it) to set the default font type. Use SetFont to specify the font to use for an element using the default font type. Use SetFontGD and SetFontTTF to specify the font of an element using the specific type of font.

History

This function was added at PHPlot-5.0.6, along with SetFontTTF, to allow mixing GD and TrueType fonts in the same graph.

SetFontTTF

SetFontTTF — Select a TrueType font to use for a plot element

Synopsis

```
$plot->SetFontTTF($elem, $font, [$size], [$line_spacing])
```

Description

SetFontTTF selects a TrueType font and size to use for one plot element (for example, the title). This function supplements [SetFont](), which selects a GD or TTF font depending on the currently selected font type. SetFontTTF always selects a TrueType font, even if TrueType fonts are not the default font type.

Parameters

$elem
>   The name of the element to change the font for. Use one of the following strings: 'title', 'legend', 'generic', 'x_label', 'y_label', 'x_title', or 'y_title'. (The 'generic' font is currently used for pie chart percentage labels and error message images.)

$font
>   Selects the TrueType font to use. This is either the full pathname of a TrueType font filename, or the filename (without path) if the font file is either located in the default TrueType font directory set with [SetTTFPath]() or can be found by GD using its default search rules. See [Section 3.8.2, "TrueType Font Selection"]() for more information. An empty string or NULL can be specified to use the default TrueType font.

$size
>   The TrueType font size in points. If not specified, a default value of 12 is used.

$line_spacing
>   Optional line spacing adjustment for this text element. For TrueType text, this is an adjustment factor for the built-in font spacing. See [SetLineSpacing]() for details.

Notes

See [SetFont]() for more information about fonts. See [SetLineSpacing]() for more information about line spacing.

Use [SetUseTTF]() (or [SetDefaultTTFFont](), which calls it) to set the default font type. Use [SetFont]() to specify the font to use for an element using the default font type. Use [SetFontGD]() and SetFontTTF to specify the font of an element using the specific type of font.

History

Starting with PHPlot-5.1.3, TrueType fonts are validated by trying to use the font with a non-drawing operation, rather than by seeing if the font file exists. See [Section 3.8.2, "TrueType Font Selection"]() for more information.

This function was added at PHPlot-5.0.6, along with [SetFontGD](), to allow mixing GD and TrueType fonts in the same graph.

SetGridColor

SetGridColor — Set the color used for the axes and borders

Synopsis

```
$plot->SetGridColor($color)
```

Description

`SetGridColor` sets the color used for the X and Y axes, the plot border, the legend border, and pie chart percentage labels (but not the grid).

Parameters

$color
    Color value to use. See Section 3.5, "Colors" for more on color values.

Notes

The default color is black.

This doesn't actually set the color used by the grid; for that see SetLightGridColor. We have no idea why this is so.

For pie charts drawn with no shading, this also sets the color of the pie slice borders.

SetImageBorderColor

SetImageBorderColor — Set image border color, if enabled

Synopsis

```
$plot->SetImageBorderColor($color)
```

Description

`SetImageBorderColor` sets the color to use for a border around the entire image.

Parameters

$color
    Color value to use. See Section 3.5, "Colors" for more on color values.

Notes

The image border is only drawn if SetImageBorderType is called with a type other than 'none'. By default there is no border. If the border is enabled but no color is set, the default color is gray (194,194,194).

Either the specified color, a darker shade of that color, or both are used to draw the image border. See SetImageBorderType for details.

The border width can be set with SetImageBorderWidth.

SetImageBorderType

SetImageBorderType — Draw a border around the image

Synopsis

```
$plot->SetImageBorderType($sibt)
```

Description

`SetImageBorderType` controls the drawing of a border around the entire image. By default, no border is drawn.

Parameters

$sibt
> A string indicating the desired border type: 'raised', 'plain', 'solid', or 'none'. Use 'none' to revert to the default of no border. Use 'solid' for a border drawn with the specified color. Use 'plain' for a border drawn with a darker shade of the specified color. Use 'raised' for a border that appears to raise the image, by drawing the top and left sides with the specified color and the bottom and right sides with a darker shade of the specified color.

Notes

The base color for the image border is set with SetImageBorderColor. Either that color, a darker shade of that color, or both are used to draw the border, as described above.

The image border width can be set with SetImageBorderWidth. The default width is 1 pixel for types 'plain' and 'solid', and 2 pixels for 'raised'.

It can be useful to have a border drawn around images when embedded in an HTML page. An alternative to this function is to use the BORDER attribute in the IMG tag when embedding the image.

History

Through PHPlot-5.1.1, the choices were 'none', 'plain' for a 1 pixel wide border using the darker shade, or 'raised' for a 2 pixel wide border using the specified color for the top and left sides and a darker shade for the bottom and right. There was no way to control the border width. Starting with PHPlot-5.1.2, a new choice 'solid' was added to use the exact color specified with SetImageBorderColor rather than using the darker shade like 'plain' uses (which may have been a bug). A new function SetImageBorderWidth was added to control the border width.

Use of 'none' to turn the border off was added in PHPlot-5.1.0.

SetImageBorderWidth

SetImageBorderWidth — Set the width for the image border

Synopsis

```
$plot->SetImageBorderWidth($width)
```

Description

`SetImageBorderWidth` sets the width of an image border, if enabled by [SetImageBorderType](#).

Parameters

$width
    The desired border width, in pixels.

Notes

Use [SetImageBorderType](#) to enable the border. Use [SetImageBorderColor](#) to set the border color.

By default, the image border width is 1 pixel for a 'solid' or 'plain' border type, and 2 pixels for a 'raised' border type.

History

This function was added in PHPlot-5.1.2.

SetIsInline

SetIsInline — Set the output image to be inline - without HTTP headers

Synopsis

```
$plot->SetIsInline($is_inline)
```

Description

SetIsInline determines if HTTP headers are sent along with the output image or not. By default, HTTP headers are sent, identifying the image type for browsers. If PHPlot output is not being directed to a browser, or in other special-purpose applications, you can turn off the HTTP content type header using this function.

Parameters

$is_inline
>   True to suppress HTTP content type headers, False to include the headers.

Notes

The default is to include the headers. However, the PHP CLI (command line interface) never outputs headers, so using this function is not necessary if you are using the PHP CLI to create image files by redirecting standard output to a file.

You must call SetIsInline(True) if you are sending PHPlot output to a file with [SetOutputFile](#), or by supplying a filename argument to the [PHPlot](#) constructor.

SetLabelScalePosition

SetLabelScalePosition — Position pie-chart percentage labels

Synopsis

```
$plot->SetLabelScalePosition($blp)
```

Description

`SetLabelScalePosition` adjusts the position of the pie chart data labels which show the percentage values of the slices.

Parameters

$blp
  Position factor for pie chart labels. This is a floating point number between 0 and 1.0. The initial value is 0.5, which places the labels outside the circumference of the pie. Smaller values move the labels in to the center, and larger values move them out from the center.

Notes

This affects pie charts only. The maximum value is 1.0 but as that value is approached the labels will move off the page. The minimum value is 0 which makes the labels pile up in the center of the pie.

The labels always display percentages. If you want to display numeric data for each slice, try putting it in the legend with [SetLegend](#).

SetLegend

SetLegend — Add text to a legend box

Synopsis

```
$plot->SetLegend($leg)
```

Description

SetLegend sets the text to be displayed in the legend. A legend is often needed when a plot contains more than one data set, to identify the purpose of the different data sets being plotted. The legend text consists of multiple lines, with each line identifying one data set on the plot. PHPlot adds an identifying color box next to each line in the legend.

Parameters

$leg
    An array with each element containing the text for one line of the legend. Or, if not an array, the one line to be appended to the legend. See notes.

Notes

The legend text needs to contain one line of text for each data set plotted on the graph, in the same order as the data array. You can supply all the legend lines in a single call to SetLegend as an array, or you can build up the legend one line at a time with multiple calls to SetLegend, supplying one line per call (in the same order as the data sets in the data array).

By default, no legend is displayed.

To cancel a legend (perhaps as part of drawing multiple plots on an image), pass an empty array or NULL as $leg.

History

Accepting NULL as a valid argument value was added in PHPlot-5.3.1. Through PHPlot-5.3.0, passing NULL would cause an error.

SetLegendPixels

SetLegendPixels — Position the legend on the image (device coordinates)

Synopsis

```
$plot->SetLegendPixels([$x, $y])
```

Description

`SetLegendPixels` lets you position the legend on the image, using device coordinates, rather than letting PHPlot position it for you.

Parameters

$x , *$y*
    Absolute device coordinates of the upper left corner of the legend box. The units are pixels and the origin is in the upper left corner of the image. If the values are omitted or NULL, the default behavior is restored.

Notes

The default behavior is for PHPlot to position the legend in the upper right corner of the plot area.

See also SetLegendWorld.

History

Through PHPlot-5.3.0, the arguments were required. To restore the default behavior, it was necessary to use `SetLegendPixels(NULL, NULL)`. Starting with PHPlot-5.3.1, the arguments may be omitted to restore the default behavior.

SetLegendStyle

SetLegendStyle — Control the appearance of the legend

Synopsis

```
$plot->SetLegendStyle($text_align, [$colorbox_align], [$style])
```

Description

SetLegendStyle controls the appearance of the legend, which is drawn if [SetLegend](#) is used. The legend contains of a series of text strings and color boxes, identifying the plot lines. SetLegendStyle sets the alignment of the text strings and color boxes. The color boxes are aligned within the legend box, left or right, and then the text strings are aligned within the remaining space, left or right. SetLegendStyle can also turn off the color boxes.

Parameters

$text_align
: A string indicating the alignment of the text strings: 'left' or 'right'. If 'left', the text strings are left-aligned between the legend box edge and the color boxes. If 'right', the text strings are right-aligned between the legend box edge and the color boxes.

$colorbox_align
: Optional string indicating the alignment of the color boxes: 'left', 'right', or 'none'. If 'left', the color boxes are drawn along the left side of the legend box. If 'right', the color boxes are drawn along the right side of the legend box. If 'none', no color boxes are drawn. If this parameter is omitted, the same alignment as $text_align is used.

$style
: Optional parameter, reserved for future use, and currently ignored.

Notes

By default, the color boxes are lined up along the right side of the legend box, and the text strings are right-aligned just left of the color boxes. The following figure shows the four possible alignment choices.
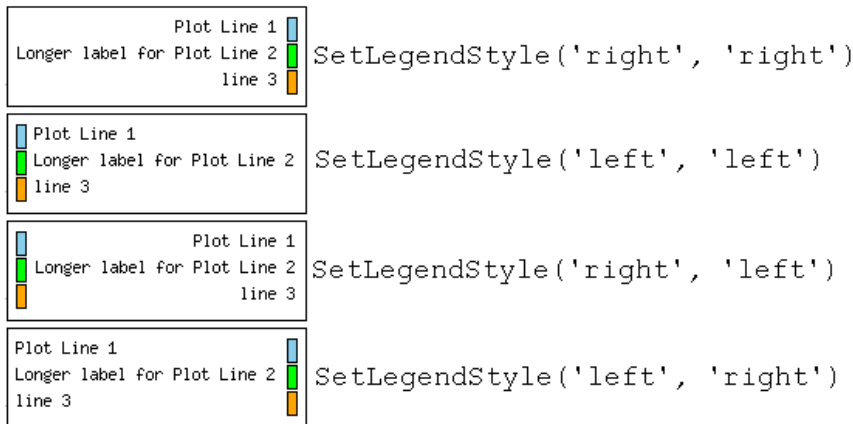
Using 'none' for $colorbox_align results in a legend with only text lines. This is not recommended for multi-line or multi-dataset plots, unless you have provided some other way to indicate which legend text line goes with which plot, or if you are using the legend for some purpose other than identifying the plot lines.

History

This function was added to PHPlot-5.0.4.

SetLegendWorld

SetLegendWorld — Position the legend on the image (world coordinates)

Synopsis

```
$plot->SetLegendWorld($x, $y)
```

Description

`SetLegendWorld` lets you position the legend on the image, using world coordinates, rather than letting PHPlot position it for you. (World coordinates are the coordinate space of your data points.)

Parameters

$x , *$y*
> World coordinates of the upper left corner of the legend box. The units and origin are the same as the data you are plotting.

Notes

The default behavior is for PHPlot to position the legend in the upper right corner of the plot area.

See also [SetLegendPixels](#).

History

Through PHPlot-5.0rc3, it was required that the data array, axis types, and any other setting which affects the scale of the data be set up before this function is used. Starting at PHPlot-5.0.4 this is no longer required, as the coordinates you supply are not scaled until the plot is drawn.

SetLightGridColor

SetLightGridColor — Set the color for grid lines and X data label lines

Synopsis

```
$plot->SetLightGridColor($color)
```

Description

`SetLightGridColor` sets the color used for the X and Y grid lines. This color is also used for X Data Lines, if enabled with [SetDrawXDataLabelLines](#).

Parameters

$color
    Color value to use. See [Section 3.5, "Colors"](#) for more on color values.

Notes

This function, not [SetGridColor](#), sets the color for the grid lines. The default color is gray.

SetLineSpacing

SetLineSpacing — Set spacing between lines of multi-line text elements

Synopsis

```
$plot->SetLineSpacing($spc)
```

Description

`SetLineSpacing` sets the default spacing between lines of a multi-line text element. Multiple lines can be specified in titles, for example, by placing a newline between lines (for example: "Line 1\nLine 2"). Line spacing also affects the legend. You can override the default line spacing for individual text elements (such as the X title) using [SetFont](), [SetFontGD](), and [SetFontTTF]().

Parameters

$spc
   Desired default line spacing factor. For the built-in GD fonts, this is the number of pixels between lines. For TrueType fonts, this is an adjustment factor for the font's built-in line spacing (see notes).

Notes

The default line spacing value is 4. For GD fonts, this is the number of pixels between text lines, and is independent of the font size.

TrueType fonts have a built-in line spacing amount, which is the distance between the baseline of one text line and the top of the next text line. With TrueType text, PHPlot uses the line spacing value (set with SetLineSpacing or one of the SetFont functions) as an adjustment factor for this built-in line spacing. A value of 4 produces the built-in line spacing, lower values reduce the spacing and larger values increase the spacing. A value of 0 for line spacing would result in the baseline of one line nearly touching the top of the next line.

History

The interpretation of the line spacing as an adjustment factor for TrueType text, rather than a value in pixels, began with PHPlot-5.0.6. This is also when the line spacing could be adjusted for each text element.

At PHPlot-5.0.5, line spacing was the number of pixels between lines of text, for both GD or TrueType fonts. Before PHPlot-5.0.5, line spacing did not affect TrueType font text at all, except in the legend.

Before PHPlot-5.0.5, the line spacing had to be set before setting titles, because the title height was calculated when setting the title. This restriction was removed in PHPlot-5.0.5, so line spacing and titles can be set in either order.

SetLineStyles

SetLineStyles — Set the line style (solid or dashed) for each data set

Synopsis

```
$plot->SetLineStyles($ls)
```

Description

SetLineStyles selects the line style for each plotted data set.

Parameters

$ls
   An array of strings, each either 'solid', for solid lines; 'dashed', for dashed lines, or 'none', to suppress the lines (see notes). Or, a single value of 'solid' or 'dashed' to apply to all data sets.

Notes

If an array is used for *$ls*, it must use zero based sequential integer indexes. This is what the PHP manual calls 'the usual integer indices (starting from zero, increasing by one)'.

The line style for dashed lines can be set with [SetDefaultDashedStyle](#).

By default, the line styles are 'solid', 'solid', and 'dashed'. (As with all style arrays, PHPlot duplicates the array as needed for the number of data sets to be plotted.) This means the 3rd and 6th data sets will plot as dashed lines unless you use SetLineStyles to change it.

A line style can be set to 'none' to suppress the line for that data set. This is only useful with 'linepoints' plot types, and results in a 'points' plot type for that data set: markers only, but no lines. (This is available with PHPlot-5.0rc3 and higher.)

SetLineWidths

SetLineWidths — Set line width (thickness) for each data set

Synopsis

`$plot->SetLineWidths($lw)`

Description

`SetLineWidths` sets the desired line width to be used when plotting each data set.

Parameters

$lw
    An array of line widths in pixels, one for each data set to be plotted. Or, a single value to be used for all data sets.

Notes

If an array is used for *$lw*, it must use zero based sequential integer indexes. This is what the PHP manual calls 'the usual integer indices (starting from zero, increasing by one)'.

By default, all data set plot lines are 1 pixel wide.

SetMarginsPixels

SetMarginsPixels — Set margins around the plot area

Synopsis

```
$plot->SetMarginsPixels([$lm], [$rm], [$tm], [$bm])
```

Description

`SetMarginsPixels` sets the size of the margins around the plot area. By default, the margin sizes are automatically calculated based on the space needed. Use SetMarginsPixels to override these automatic calculations and set specific margin sizes. The four margins are specified independently and in pixels.

Parameters

$lm

Optional argument specifying the left margin, in pixels. If omitted or NULL, the value is automatically calculated.

$rm

Optional argument specifying the right margin, in pixels. If omitted or NULL, the value is automatically calculated.

$tm

Optional argument specifying the top margin, in pixels. If omitted or NULL, the value is automatically calculated.

$bm

Optional argument specifying the bottom margin, in pixels. If omitted or NULL, the value is automatically calculated.

Notes

By default, the margin sizes are automatically calculated based on the space needed (for the axis labels, tick labels, and tick marks). Use SetMarginsPixels to override these automatic calculations and control the margins.

The upper left corner of the plot area is at device coordinates (Left_Margin, Top_Margin). The lower right corner of the plot area is at device coordinates (Image_Width - Right_Margin, Image_Height - Bottom_Margin).

SetMarginsPixels and [SetPlotAreaPixels](#) perform the same function with different semantics. It makes no sense to use both - only the last one called will have an effect.

Trailing defaulted arguments can be omitted, but non-trailing defaulted arguments must be specified as NULL. For example, to set the right margin to 100 pixels, and let PHPlot calculate the other three margins, use:

```
$plot->SetMarginsPixels(NULL, 100);
```

History

Through PHPlot-5.0.6, SetMarginsPixels required all 4 arguments be specified and not be NULL. Starting with PHPlot-5.0.7, each margin can either be specified or automatically calculated.

SetNumberFormat

SetNumberFormat — Set the separators used when formatting number labels

Synopsis

```
$plot->SetNumberFormat($decimal_point, $thousands_sep)
```

Description

`SetNumberFormat` sets the separator characters used when formatting number labels. Labels are formatted as numbers when the 'data' format type is selected with SetXLabelType or SetYLabelType.

Parameters

$decimal_point
    The character used as a decimal point, to separate the integer part of the label from the fraction part.

$thousands_sep
    The character used as a thousands grouping separator (placed between every group of 3 digits left of the decimal point).

Notes

These separators are only used for labels when 'data' mode formatting is selected with SetXLabelType or SetYLabelType.

If `SetNumberFormat` is not used, PHPlot attempts to get the proper separator characters from your system locale. If this works and your locale is set correctly, you will probably not need to use this function. If locale information is not available, the default for decimal_point is a period, and the default for thousands_sep is a comma.

If your system locale is set to "C" or "POSIX", you might find that there are no thousands separators in your formatted labels. This is the correct behavior for those locales. If you cannot select a more specific locale, use `SetNumberFormat` to set the correct separators.

To set the number of decimal places, use SetPrecisionX and SetPrecisionY.

If you are trying to force a specific locale with setlocale(), it will not work, because PHPlot uses setlocale(LC_ALL, '') to import locale information from the system, and this overrides a forced locale from your script. On non-Windows platforms, you can force a locale using environment variables, but this does not work on Windows. To address this, PHPlot has (see the History section below) a special member variable *locale_override* that prevents PHPlot from importing locale settings from the system. For example, if the following code is used, numeric formatting will use the fr_CA locale settings, regardless of the system locale.

```
setlocale(LC_ALL, 'fr_CA');  # On Windows use: 'French_Canada'
$plot = new PHPlot(800, 600);
$plot->locale_override = True;
```

History

The *locale_override* hook was added in PHPlot-5.1.0. Before that, there was no way to force a specific locale on Windows, and on other platforms a locale could be forced only by using environment variables. The hook was added primarily for testing on Windows, but could be needed in other situations too.

This function was added to PHPlot-5.0.4. Versions up to and including 5.0rc3 always used a period for decimal point, and comma for thousands separator.

SetNumXTicks

SetNumXTicks — Set the number of X tick marks

Synopsis

```
$plot->SetNumXTicks([$nt])
```

Description

SetNumXTicks sets the exact number of tick marks to draw on the X axis. You can use either this function or SetXTickIncrement (but not both) to control the tick mark spacing.

Parameters

$nt
   Integer number of tick marks to draw. If the value is omitted or an empty string, the default behavior is restored.

Notes

If neither SetNumXTicks nor SetXTickIncrement is used, the tick mark interval is calculated as 1/10th of the X data range.

History

Through PHPlot-5.3.0, the argument was required. To restore the default behavior, it was necessary to use SetNumXTicks(''). Starting with PHPlot-5.3.1, the argument may be omitted to restore the default behavior.

SetNumYTicks

SetNumYTicks — Set the number of Y tick marks

Synopsis

```
$plot->SetNumYTicks([$nt])
```

Description

SetNumYTicks sets the exact number of tick marks to draw on the Y axis. You can use either this function or SetYTickIncrement (but not both) to control the tick mark spacing.

Parameters

$nt
   Integer number of tick marks to draw. If the value is omitted or an empty string, the default behavior is restored.

Notes

If neither SetNumYTicks nor SetYTickIncrement is used, the tick mark interval is calculated as 1/10th of the Y data range.

History

Through PHPlot-5.3.0, the argument was required. To restore the default behavior, it was necessary to use SetNumYTicks(''). Starting with PHPlot-5.3.1, the argument may be omitted to restore the default behavior.

SetOutputFile

SetOutputFile — Redirect PHPlot output to a file

Synopsis

```
$plot->SetOutputFile($output_file)
```

Description

`SetOutputFile` arranges for image output to be sent to a file instead of to the browser (or standard output). By default, output is sent to the browser (or standard output, if running from the command line).

Parameters

$output_file
    Pathname of the file to write the image data into.

Notes

The output file will only be produced if [SetIsInline](True) is called.

By default, there is no output file, and the image is written to the browser or standard output.

SetPlotAreaBgImage

SetPlotAreaBgImage — Set a graphic file to be used in the plot area background

Synopsis

```
$plot->SetPlotAreaBgImage($input_file, [$mode])
```

Description

`SetPlotAreaBgImage` sets an image file to be used as the plot area background. The image can be scaled or tiled to fit.

Parameters

$input_file
    Path to the file to be used. The file can be any type allowed by GD, which usually includes JPEG, GIF, and PNG.

$mode
    Optional display mode for the background image: one of the strings 'centeredtile', 'tile', or 'scale'. The default is 'tile'.

Notes

If a background image has been set, background color (set with [SetPlotBgColor](#)) is ignored.

Scale mode scales the supplied background image file to fit the plot area. Tile and centeredtile modes repeat the supplied background image file as needed to fit the plot area. The difference is that centeredtile offsets the start position within the background image by half its size, which works better for some images.

SetPlotAreaBgImage sets a background for the plot area, while [SetBgImage](#) sets a background for the entire image area. If both are used, the plot area background overlays that portion of the overall background.

If `$input_file` is `NULL`, no background image will be used for the plot area.

SetPlotAreaPixels

SetPlotAreaPixels — Set the limits for the plot area in device coordinates

Synopsis

```
$plot->SetPlotAreaPixels([$x1], [$y1], [$x2], [$y2])
```

Description

`SetPlotAreaPixels` sets the area to be used for the plot within the image, in device coordinates. (Device coordinates are GD coordinates, with the origin at the top left of the image, X values increase to the right, Y values increase down, and the units in pixels.) The plot area is equal to the image area minus the margins. By default, the margins (and therefore the plot area) are automatically calculated based on the space needed for titles, labels, etc. Use SetPlotAreaPixels to override these automatic calculations and control the plot area. The four coordinate values are specified independently. SetPlotAreaPixels and [SetMarginsPixels](#) perform the same function with different semantics.

Parameters

$x1 , *$y1*
> Device coordinates of the top left corner of the area to use for the plot. Each value is optional, and if omitted or NULL the automatically calculated value is used.

$x2 , *$y2*
> Device coordinates of the bottom right corner of the area to use for the plot. Each value is optional, and if omitted or NULL the automatically calculated value is used.

Notes

The plot area is equal to the image area minus the margins. By default, the margins (and therefore the plot area) are automatically calculated based on the space needed (for the axis labels, tick labels, and tick marks). Use SetPlotAreaPixels to override these automatic calculations and control the plot area.

SetPlotAreaPixels and [SetMarginsPixels](#) perform the same function with different semantics. It makes no sense to use both - only the last one called will have an effect.

SetPlotAreaPixels is also used when placing multiple plots on an image. See [Section 4.6, "Multiple Plots Per Image"](#) for more information.

Trailing defaulted arguments can be omitted, but non-trailing defaulted arguments must be specified as NULL. For example, to fix the bottom right corner of the plot area at (600, 400) and let PHPlot calculate the upper left corner use:

```
$plot->SetPlotAreaPixels(NULL, NULL, 600, 400);
```

You do not have to specify both X and Y. The following example sets the right edge of the plot area at X=600 and lets the other 3 edges be automatically calculated:

```
$plot->SetPlotAreaPixels(NULL, NULL, 600);
```

History

Through PHPlot-5.0.6, SetPlotAreaPixels required all 4 arguments be specified and not be NULL. Starting with PHPlot-5.0.7, each X and Y parameter can either be specified or automatically calculated.

SetPlotAreaPixels and SetPlotAreaWorld can be called in either order. Through PHPlot-5.0.4 this was because SetPlotAreaPixels would reset the scale factors if SetPlotAreaWorld was already called. Starting with PHPlot-5.0.5, both functions just store the information, and no calculations take place until DrawGraph is used.

SetPlotAreaWorld

SetPlotAreaWorld — Override automatic data scaling to device coordinates

Synopsis

```
$plot->SetPlotAreaWorld([$xmin], [$ymin], [$xmax], [$ymax])
```

Description

`SetPlotAreaWorld` changes the range for World Coordinate space. This is the coordinate space of the data to be plotted, and is translated and scaled to fit into the Device Coordinate space of the image. By default, PHPlot defines the world coordinate space by the actual limits of the data to be plotted. By using SetPlotAreaWorld, you can override one or more of these calculated limits. Each parameter you specify overrides the corresponding calculated limit. Each defaulted or NULL parameter is ignored and the calculated limit is used.

Since tick marks are automatically positioned based on the world coordinate limits, SetPlotAreaWorld is also used to force a specific range for tick marks.

Parameters

$xmin

Optional argument specifying the desired X data range minimum value. If omitted or NULL, the value is calculated from the actual plot data.

$ymin

Optional argument specifying the desired Y data range minimum value. If omitted or NULL, the value is calculated from the actual plot data.

$xmax

Optional argument specifying the desired X data range maximum value. If omitted or NULL, the value is calculated from the actual plot data.

$ymax

Optional argument specifying the desired Y data range maximum value. If omitted or NULL, the value is calculated from the actual plot data.

Notes

Trailing defaulted arguments can be omitted, but non-trailing defaulted arguments must be specified as NULL. For example, to set the minimum X value to 10, the maximum X value to 100, and default the Y scaling, you can use:

```
$plot->SetPlotAreaWorld(10, NULL, 100);
```

History

SetPlotAreaPixels and SetPlotAreaWorld can be called in either order. Through PHPlot-5.0.4 this was because SetPlotAreaWorld would reset the scale factors if SetPlotAreaPixels was already called. Starting with PHPlot-5.0.5, both functions just store the information, and no calculations take place until DrawGraph is used.

Through PHPlot-5.0.4, SetPlotAreaWorld needed to access and interpret the data array, so it had to be called after SetDataValues sets the data array, and after SetDataType (if used). Starting with PHPlot-5.0.5, this restriction no longer applies and the functions can be called in any order.

SetPlotBgColor

SetPlotBgColor — Set plot area background color

Synopsis

```
$plot->SetPlotBgColor($color)
```

Description

`SetPlotBgColor` sets the background color of the plot area of the image (the area inside the margins, title, and usually the axes).

Parameters

$color
    Color value to use. See Section 3.5, "Colors" for more on color values.

Notes

Background color is disabled by default. If you want a background color for the plot area, you must enabled it with SetDrawPlotAreaBackground.

Background image takes priority over background color. If a background image was set with SetPlotAreaBgImage, then no background color will be drawn, even if it was enabled.

The default background color for the plot area is white (if enabled as noted above). If background color for the plot area is not enabled, the overall background color (or image) will be seen in the plot area.

SetPlotBorderType

SetPlotBorderType — Control how much of a border is drawn around the plot

Synopsis

```
$plot->SetPlotBorderType($pbt)
```

Description

`SetPlotBorderType` controls how much of a border is drawn around the plot. By default, left and right side borders are drawn.

Parameters

$pbt
> A string or array indicating where to draw the plot borders. If a string, it must be one of the following values:

Value	Description
bottom	Border on bottom plot
full	Border on all four sides of the plot
left	Border on left side of plot
none	No plot area border
right	Border on right side of plot
sides	Border on left and right sides of plot
top	Border on top of plot

> An array of strings can be used. Each element in the array must be one of the above values. For example, use either `array('left', 'right', 'bottom')` or `array('sides', 'bottom')` to get a plot border on the sides and bottom but not the top.

Notes

Plot border color is set by [SetGridColor](this color also applies to the axes and other elements). The default color is black.

The X and Y axis lines and the plot border are all drawn in the same color. Usually, the X axis line is drawn at the bottom of the plot area, and the Y axis line is drawn on the left side of the plot area. (See [SetXAxisPosition](and [SetYAxisPosition](for more information on the axis positions.) Enabling or disabling a plot border edge that corresponds to an axis line will have no visible effect because the axis line will be drawn there.

History

Through PHPlot-5.1.1, the available choices were 'left', 'sides', 'full', or 'none', and only a single string option was supported. Starting with PHPlot-5.1.2, the additional choices 'right', 'top', and 'bottom' were added, as well as the ability to pass an array of choices.

SetPlotType

SetPlotType — Select the type of plot - how the data will be graphed

Synopsis

```
$plot->SetPlotType($pt)
```

Description

SetPlotType selects a type of plot from among the plot types supported by PHPlot. This includes bar charts, line plots, point plots, and others. A complete description of the type of plots available can be found in [Section 3.4, "PHPlot Plot Types"](#).

Parameters

$pt

A string indicating the type of plot. The following types are available:

- area
- bars
- candlesticks
- candlesticks2
- linepoints
- lines
- ohlc
- pie
- points
- squared
- stackedarea
- stackedbars
- thinbarline

Notes

The default plot type is 'linepoints'.

History

Plot types 'candlesticks', 'candlesticks2', and 'ohlc' were added in PHPlot-5.3.0.

Plot type 'stackedarea' was added in PHPlot-5.1.1.

SetPointShapes

SetPointShapes — Select a point shape for each data set

Synopsis

```
$plot->SetPointShapes($pt)
```

Description

SetPointShapes assigns a point shape to each data set in a plot. 'Point' here refers to the marker drawn at each data point in 'points' and 'linepoints' type plots. For example, if each data row contains 4 Y values, the first point shape will be used for the first Y value, the second point shape for the second Y value, etc. There are 20 point shapes to chose from.

Parameters

$pt
> An array of point shape names, or a string naming a single point shape. If a string, that shape name is used for all data sets. If an array, the array values name the point shapes for each subsequent data set in a plot. The following shapes are available:

Shape	Shape Name	Description
⋈	bowtie	Two filled triangles pointing right and left towards the point.
□	box	A square outline centered on the point.
○	circle	A hollow circle centered on the point.
×	cross	An X centered on the point.
▲	delta	A filled triangle pointing up, centered on the point.
◆	diamond	A filled diamond (square rotated 45 degrees), centered on the point.
●	dot	A filled circle centered on the point.
▽	down	An unfilled triangle pointing down, centered on the point.
–	halfline	A short line from the point going left.
⬠	home	A filled 5-sided shape, centered on the point.
⧗	hourglass	Two filled triangles pointing up and down towards the point.
—	line	A horizontal line centered on the point.
+	plus	A plus sign centered on the point.
■	rect	A filled square centered on the point.
✳	star	Four lines crossing at the point.
▣	target	A square outline with two filled squares and two open squares, centered on the point.

Shape	Shape Name	Description
▼	triangle	A filled triangle pointing down from the point.
▼	trianglemid	A filled triangle pointing down to the point.
△	up	An unfilled triangle pointing up, centered on the point.
▼	yield	A filled triangle pointing down, centered on the point.
	none	No marker (see notes).

Example 5.7, "Line/Point Plot, Point Shapes" also shows all of the point shapes.

Notes

If an array is used for pt, it must use zero based sequential integer indexes. This is what the PHP manual calls 'the usual integer indices (starting from zero, increasing by one)'.

This applies only to 'points' and 'linepoints' plot types.

By default, ten shapes are used in order: diamond, dot, delta, home, yield, box, circle, up, down, and cross.

A point shape can be set to 'none' to suppress the point markers for that data set. This is only useful with 'linepoints' plot types, and results in a 'lines' plot type for that data set: a line only, but no markers.

PHPlot duplicates the entries in the shorter of the two arrays, point sizes (set by SetPointSizes) and point shapes, to make both arrays the same size. Then it uses the entries in order, restarting at the beginning, for each data set at each X value. For example, if point sizes is (6, 10), and point shapes is ('diamond', 'dot', 'rect'), then PHPlot first extends point sizes to (6, 10, 6) to match the point shapes. If there are 4 data sets to plot, PHPlot draws the point markers at each X value as: diamond (size 6), dot (size 10), rect (size 6), diamond (size 6).

History

Through PHPlot-5.0.7, these ten shapes were available: halfline, line, plus, cross, rect, circle, dot, diamond, triangle, trianglemid, and none. The default shape for all data sets was 'diamond'. Starting with PHPlot-5.1.0, ten new point shapes were added, and different shape defaults were assigned for ten data sets. To restore the behavior in PHPlot-5.0.7 and earlier, call `SetPointShapes('diamond')`.

Using 'none' as a point shape was added in PHPlot-5.0rc3.

SetPointSizes

SetPointSizes — Sets the point size for each data set

Synopsis

```
$plot->SetPointSizes($ps)
```

Description

SetPointSizes assigns a point size to each data set in a plot. 'Point' here refers to the marker drawn at each data point in 'points' and 'linepoints' type plots. For example, if each data row contains 4 Y values, the first point size will be used for the first Y value, the second point size for the second Y value, etc.

Parameters

$ps
:   An array of point sizes, or a single value. All values are in pixels. If a single value, that size is used for all data sets. If an array, the array values are the sizes for each subsequent data set in a plot.

Notes

If an array is used for *$ps*, it must use zero based sequential integer indexes. This is what the PHP manual calls 'the usual integer indices (starting from zero, increasing by one)'.

This applies only to 'points' and 'linepoints' plot types.

PHPlot duplicates the entries in the shorter of the two arrays, point sizes and point shapes (set by [SetPointShapes](#)), to make both arrays the same size. Then it uses the entries in order, restarting at the beginning, for each data set at each X value. For example, if point sizes is (6, 10), and point shapes is ('diamond', 'dot', 'rect'), then PHPlot first extends point sizes to (6, 10, 6) to match the point shapes. If there are 4 data sets to plot, PHPlot draws the point markers at each X value as: diamond (size 6), dot (size 10), rect (size 6), diamond (size 6).

By default, all point sizes are 6 pixels.

History

Through PHPlot-5.0.7, the default size array was (5, 5, 3). However, there were bugs in processing the sizes of the point shapes and point sizes arrays. Also, for some point sizes, PHPlot rounded the size up to the next even number.

SetPrecisionX

SetPrecisionX — Set precision for numeric formated X labels

Synopsis

```
$plot->SetPrecisionX($prec)
```

Description

`SetPrecisionX` sets the desired numeric precision for X tick and data labels, and also enables 'data' mode formatting of those labels with [SetXLabelType](#).

Note

This function is retained for compatibility, but use of [SetXLabelType](#) is preferred.

Parameters

$prec
   The desired numeric precision. This is the number of decimal positions to output.

Notes

Setting numeric precision with this function automatically enables 'data' mode formatting as if `SetXLabelType('data')` was called.

The default is to format numbers with 1 decimal position, but only if 'data' mode formatting is selected.

This applies to both X data labels and X tick labels (only one of which should be enabled). Using 'data' format for X data labels only makes sense when your data array contains numeric data in the label position.

History

Starting with PHPlot-5.0.6, using `SetPrecisionX($n)` is exactly the same as calling `SetXLabelType('data', $n)`.

SetPrecisionY

SetPrecisionY — Set precision for numeric formated Y tick labels

Synopsis

```
$plot->SetPrecisionY($prec)
```

Description

`SetPrecisionY` sets the desired numeric precision for Y tick and data labels, and also enables 'data' mode formatting of those labels with [SetYLabelType](#).

Note

This function is retained for compatibility, but use of [SetYLabelType](#) is preferred.

Parameters

$prec
   The desired numeric precision. This is the number of decimal positions to output.

Notes

Setting numeric precision with this function automatically enables 'data' mode formatting as if `SetYLabelType('data')` was called.

The default is to format numbers with 1 decimal position, but only if 'data' mode formatting is selected.

This function applies to Y tick labels, and also to the Y data labels available with bar charts and stacked bar charts.

History

Starting with PHPlot-5.0.6, using `SetPrecisionY($n)` is exactly the same as calling `SetYLabelType('data', $n)`.

SetPrintImage

SetPrintImage — Determine whether or not to automatically output the image when the plot is drawn

Synopsis

```
$plot->SetPrintImage($pi)
```

Description

`SetPrintImage` determines whether or not to automatically output the image (as if [PrintImage](#) was used) when a plot is drawn (with [DrawGraph](#)). The default is True. When putting multiple plots on one image, is is necessary to defer PrintImage until after all the plots have been drawn.

Parameters

$pi

    True to automatically print the image when drawn with DrawGraph; False to defer printing the image until PrintImage is explicitly called.

Notes

See [Section 4.6, "Multiple Plots Per Image"](#) for more information on putting multiple plots on an image.

SetRGBArray

SetRGBArray — Select a color map

Synopsis

```
$plot->SetRGBArray($color_array)
```

Description

`SetRGBArray` selects a color map to use. A color map is an array of colors available to be used in the image. You can select from pre-defined color maps, or define your own. Each color in a color map has a name, and 3 or 4 color component values (red, green, blue, and optional alpha). The red, green, and blue components are in the range 0 through 255, and the optional alpha component is in the range 0 through 127.

Parameters

$color_array
> An array or a string. If an array, each element defines a color in the color map. The array element key is the color name, and the array element value is an array of three or four components. (See example below).
>
> Or, a string selecting a built-in color map. Use 'small' to select a map of 36 colors, or 'large' to select a much larger color map.

Notes

If SetRGBArray is not called, the 'small' color map is used.

More information about the color maps can be found in [Section 3.5.2, "Built-in Color Maps"](). More information about using the alpha component can be found in [Section 4.2, "Truecolor Images"]().

Color names are case sensitive.

For the large color map to be loaded with SetRGBArray('large'), the file `rgb.inc.php` must be found on the PHP include path or in the same directory as `phplot.php`.

When defining a color map, care must be taken to avoid invalid color errors. PHPlot uses a number of pre-defined color names, and these must either be defined in your color map or changed. If you define your own color map, it should always contain colors named 'black', 'white', and 'gray', and you should use [SetDataColors]() and [SetErrorBarColors]() to select colors which are in your color map.

Examples

An example of a user-defined color map is:

```
array( 'black' => array(0, 0, 0),
       'white' => array(255, 255, 255),
       'gray'  => array(190, 190, 190),
       'red'   => array(255, 0, 0),
       'green' => array(0, 255, 0),
       'blue'  => array(0, 0, 255) )
```

SetShading

SetShading — Set the size of the drop shadow for bar and pie charts.

Synopsis

```
$plot->SetShading($s)
```

Description

`SetShading` sets the size in pixels of the drop shadow used to give bar and pie charts a 3-D look. The 3-D look can be disabled by setting the shading to zero.

Parameters

$s
> Desired shading size in pixels. If this is set to 0, pie charts are flat (not rotated away from the screen) with no shadow, and bars in bar charts and stackedbar charts are drawn as rectangles without drop shadows.

Notes

The default is shading enabled with a size of 5 pixels.

If shading is turned off for bar and stackedbar charts, the bars will be drawn with borders. The color of the borders is set with [SetDataBorderColors](#).

SetSkipBottomTick

SetSkipBottomTick — Suppress the bottom Y axis tick mark and label

Synopsis

```
$plot->SetSkipBottomTick($skip)
```

Description

`SetSkipBottomTick` can be used to skip (suppress) the bottom-most Y tick mark and its label. See also [SetSkipLeftTick](#), [SetSkipRightTick](#), and [SetSkipTopTick](#).

Parameters

$skip
    If True, don't draw the bottom Y tick mark and label; if False, draw the bottom Y tick mark and label.

Notes

By default, no tick marks or labels are skipped.

SetSkipLeftTick

SetSkipLeftTick — Suppress the first X axis tick mark and label

Synopsis

```
$plot->SetSkipLeftTick($skip)
```

Description

`SetSkipLeftTick` can be used to skip (suppress) the left-most X tick mark and its label. See also [SetSkipBottomTick](), [SetSkipRightTick](), and [SetSkipTopTick]().

Parameters

$skip
    If True, don't draw the first X tick mark and label; if False, draw the first X tick mark and label.

Notes

By default, no tick marks or labels are skipped.

SetSkipRightTick

SetSkipRightTick — Suppress the last X axis tick mark and label

Synopsis

```
$plot->SetSkipRightTick($skip)
```

Description

`SetSkipRightTick` can be used to skip (suppress) the right-most X tick mark and its label. See also [SetSkipBottomTick](), [SetSkipLeftTick](), and [SetSkipTopTick]().

Parameters

$skip
    If True, don't draw the last X tick mark and label; if False, draw the last X tick mark and label.

Notes

By default, no tick marks or labels are skipped.

SetSkipTopTick

SetSkipTopTick — Suppress the top Y axis tick mark and label

Synopsis

```
$plot->SetSkipTopTick($skip)
```

Description

`SetSkipTopTick` can be used to skip (suppress) the top-most Y tick mark and its label. See also SetSkipBottomTick, SetSkipLeftTick, and SetSkipRightTick.

Parameters

$skip
    If True, don't draw the top Y tick mark and label; if False, draw the top Y tick mark and label.

Notes

By default, no tick marks or labels are skipped.

SetTextColor

SetTextColor — Set general text color

Synopsis

```
$plot->SetTextColor($color)
```

Description

`SetTextColor` sets the color which is used for several text elements on the plot, including the tick labels, data labels, and the text in the legend.

Parameters

$color
    Color value to use. See <u>Section 3.5, "Colors"</u> for more on color values.

Notes

The default text color is black.

SetTickColor

SetTickColor — Set the color of the axis tick marks

Synopsis

```
$plot->SetTickColor($color)
```

Description

`SetTickColor` sets the color of the axis tick marks.

Parameters

$color
     Color value to use. See Section 3.5, "Colors" for more on color values.

Notes

The default color for the tick marks is black.

SetTitle

SetTitle — Set the main title text for the plot

Synopsis

```
$plot->SetTitle($title)
```

Description

SetTitle sets the main plot title text. This is displayed centered at the top of the plot.

Parameters

$title
    The title text to be displayed. The string can contain multiple lines, separated by newlines (in PHP: "\n").

SetTitleColor

SetTitleColor — Set the color of the main plot title

Synopsis

```
$plot->SetTitleColor($color)
```

Description

`SetTitleColor` sets the color of the main plot title (as set with [SetTitle](#)), and the default color for the X and Y titles (as set with [SetXTitle](#) and [SetYTitle](#)).

Parameters

$color
    Color value to use. See [Section 3.5, "Colors"](#) for more on color values.

Notes

The default color for the main title is black.

Use just SetTitleColor if you want all three titles to have the same color. If you want different colors for the main, X, and/or Y titles, use SetTitleColor to set the main title color, and use [SetXTitleColor](#) and/or [SetYTitleColor](#) to set the color of the other titles.

History

Through PHPlot-5.1.3, this function set the color for all three titles. Starting with PHPlot-5.2.0, it sets the main title color, and the default color for X and Y titles.

SetTransparentColor

SetTransparentColor — Designate one color to be transparent

Synopsis

```
$plot->SetTransparentColor($color)
```

Description

`SetTransparentColor` designates one color in the image to be transparent. The designated color will not be visible (assuming the image is viewed with a program which supports transparency) - instead, everything drawn in that color will be transparent. By default, no color is transparent.

Parameters

$color
    Name of the color in the current color map (see [SetRGBArray](#)) which should become transparent.

Notes

This will only work if both the selected image file format (see [SetFileFormat](#)) and the user's browser or viewer support transparency. GIF format supports transparency. PNG format also supports transparency, but viewer support is more limited.

Example

To set the plot image background to be transparent, pick a color (here 'yellow') that won't be used anywhere else on the image, and use code like this:

```
$plot->SetBackgroundColor('yellow');
$plot->SetTransparentColor('yellow');
```

SetTTFPath

SetTTFPath — Set the default TrueType font directory

Synopsis

```
$plot->SetTTFPath($path)
```

Description

`SetTTFPath` sets the directory where [SetFont](#) and [SetFontTTF](#) can find TrueType font files.

Parameters

$path
Full path to a directory containing TrueType fonts.

Notes

The default TrueType font directory is '.', meaning fonts will be loaded from the directory containing your main PHP script. PHPlot also attempts to load fonts without any path, which allows GD to try to locate the font using its own internal rules. This works on some platforms, but not on others.

SetTTFPath does not enable the use of TrueType fonts. See the note at the end of the reference for [SetUseTTF](#) on how to set up and use TrueType fonts.

The default TrueType font directory is also used to find an internally-set default font. This means that if you use SetTTFPath, but do not specify a font name (with [SetDefaultTTFont](#), [SetFont](#), or [SetFontTTF](#)), PHPlot will attempt to select a default font from its internal list, but using the directory you specify with SetTTFPath. This can be useful on platforms where you are willing to use a default font, but the internal GD rules on how to locate fonts do not work.

See [Section 3.8.2, "TrueType Font Selection"](#) for more information.

History

Use of the default TrueType font directory to locate the initial, default font was implemented in PHPlot-5.1.3. Also with this release, fonts are first searched for without a path, which allows GD to try to locate the font using its internal rules.

Starting with PHPlot-5.0rc3, the default TrueType font directory is used both for the default font set with [SetDefaultTTFont](#) and for fonts set with [SetFont](#). Refer to those two functions for details.

SetUseTTF

SetUseTTF — Set the default font type

Synopsis

```
$plot->SetUseTTF($ttf)
```

Description

`SetUseTTF` sets the default font type to TrueType fonts or built-in GD fonts, and re-initializes all font settings.

Parameters

$ttf
> True to use TrueType fonts by default, False to use built-in GD fonts by default.

Notes

Changing the font type re-initializes all the font settings to the defaults.

The default is False, to use the built-in GD fonts.

When you enable TrueType fonts with SetUseTTF, there must be a valid default font. You can use SetDefaultTTFont to establish a default font, but since this also enables TrueType fonts as the default, you need not use SetUseTTF in that case. On some platforms, PHPlot can find a valid default font on its own. See Section 3.8.2, "TrueType Font Selection" for more information on the default font.

After enabling TrueType fonts, you can use SetFont or SetFontTTF to select fonts and sizes for individual text elements in the plot. You can also use SetFontGD to use GD fonts for one or more elements, overriding the default font type.

History

Through PHPlot-5.1.2, there was a fixed default font called `benjamingothic.ttf`, which used to be included with PHPlot, but was removed in 2006. (It was not generally suitable for plotting anyway.) As a result, you had to use SetDefaultTTFont to set a valid default font before enabling TrueType fonts with SetUseTTF(). But since SetDefaultTTFont() itself turned on TrueType fonts, SetUseTTF() was rarely useful. Starting with PHPlot-5.1.3, PHPlot will try a number of sans-serif font names, trying to find a valid font to use as the default. On platforms where this works, SetUseTTF(True) can be used without any other font setup in order to use the default TrueType font for all text.

Through PHPlot-5.0.5, this function enabled or disabled the use of TrueType font text, since all text on a graph had to be either TrueType or all GD. Starting with PHPlot-5.0.6, this function selects the default font type (and still re-initializes all fonts). Both GD and TrueType font text can now be used on a graph.

SetXAxisPosition

SetXAxisPosition — Move the X axis

Synopsis

```
$plot->SetXAxisPosition([$pos])
```

Description

`SetXAxisPosition` sets the position of the X axis.

Parameters

$pos
>   The Y position in world coordinates for the X axis. (World coordinates are the coordinate space of your data points.) If the value is omitted or an empty string, the default behavior is restored.

Notes

The given position is truncated (towards 0) to an integer value.

The default axis position differs for vertical and horizontal plots. For vertical plots, the X axis position defaults to Y=0, provided Y=0 is within the range of the graph. If Y=0 is not within the range of the graph, the X axis position will default to the edge with the smallest absolute Y value. This means the X axis will be at the bottom of the graph if all values of Y are greater than zero, and at the top of the graph if all values of Y are less than zero. (For log scale plots, however, the default X axis position is Y=1.) For horizontal plots, the X axis position defaults to the bottom of the plot.

History

Through PHPlot-5.3.0, the argument was required, and there was no way to restore the default behavior (because the argument value was always converted to an integer). Starting with PHPlot-5.3.1, the argument may be given as an empty string, or omitted, to restore the default behavior.

Through PHPlot-5.1.3, there was no special handling for horizontal plots. The X axis always defaulted to Y=0 or the Y with the smallest absolute value.

Through PHPlot-5.1.0, the default position for the X axis was the bottom of the graph whenever Y=0 was not within the range the graph, regardless of whether the data was all positive or all negative.

SetXDataLabelAngle

SetXDataLabelAngle — Set the text angle for X data labels

Synopsis

```
$plot->SetXDataLabelAngle($xdla)
```

Description

`SetXDataLabelAngle` sets the text angle for X data labels. If using TrueType fonts, any angle can be used. If using built-in GD fonts, only 0 degree and 90 degree text can be used.

Parameters

$xdla
    Desired angle for label text, in degrees.

Notes

This function applies to both X axis data labels and X data value labels. See Section 3.6, "Labels" for more information on data labels.

By default, X Data Labels are drawn at the same angle as X Tick Labels, as set with SetXLabelAngle.

History

This function was added to PHPlot in 5.1.0. Through PHPlot-5.0.7, SetXLabelAngle set the angle for both text and data labels.

Through PHPlot-5.0.7, there was a deprecated function by the same name that simply called SetXLabelAngle.

SetXDataLabelPos

SetXDataLabelPos — Position and control X data labels

Synopsis

```
$plot->SetXDataLabelPos($xdlp)
```

Description

`SetXDataLabelPos` determines if and where X data labels are drawn. For vertical plots, these are X axis data labels, which display the label strings from your data array. The labels can be drawn at the bottom of the plot (below the X axis), above the plot, in both positions, or neither. For horizontal plots, these are X data value labels, displaying the value of the data point. They can be drawn within the plot to label bars and bar segments. This type of label is only available for 'bars' and 'stackedbars' plots.

Parameters

$xdlp
> A string indicating the desired position for the X data labels:

Position	Description
plotdown	Data labels below the plot. This is for vertical plots.
plotup	Data labels above the plot. This is for vertical plots.
both	Data labels both below and above the plot. This is for vertical plots.
plotin	Data value labels to the right (or left) of each bar. For stacked bar charts, this means only the bar total labels and not the bar segment labels. This is valid for horizontal bar and horizontal stacked bar plots only.
plotstack	Data value labels to the right of each bar, and just left of the end of each bar segment. This is valid for horizontal stacked bar plots only. It turns on both bar total labels (as with 'plotin') and bar segment labels.
none	No data labels or data value labels

Notes

With vertical plots, this function controls the X axis data labels. For an example, see Section 5.3, "Example - Area Plot", where the labels are enabled by default and displayed below the X axis. With horizontal plots, this function controls the X data value labels. For examples, see Example 5.27, "Horizontal Bar Chart" and Example 5.28, "Horizontal Stacked Bar Chart".

The default position for X data labels (for vertical plots) is below the plot. However, PHPlot only enables the data labels if SetXDataLabelPos was used to position them, or if SetXTickLabelPos was not used to enable the tick labels and the data labels are not all empty.

The default position for X data value labels (for horizontal plots) is 'none', meaning no labels. X data value labels only apply to horizontal bar and stackedbar plots.

For tick labels, see SetXTickLabelPos.

The bar segment labels, if enabled ('plotstack'), are drawn inside the bars and may not be very visible if dark colors are used for the bar fill. Bar segment labels will be omitted for segments which are too short.

X data value labels will be drawn to the left of the bars for negative values. This only applies to horizontal bar charts. Stacked bar charts are not allowed to have negative values.

If X data lines are enabled with SetDrawXDataLabelLines, then SetXDataLabelPos() also determines the direction of the lines which are drawn from the data points.

The X data label text angle is set with SetXDataLabelAngle. The X data label text format can be controlled with SetXDataLabelType or SetXLabelType.

History

Horizontal bar plot data value labels were added in PHPlot-5.1.3. Through PHPlot-5.1.2, the 'plotin' and 'plotstack' values were not available.

Through PHPlot-5.0.7, the default position for X data labels was below the plot ('plotdown'). This would result in overlaid data and tick labels by default. In addition, positioning the X tick labels with SetXTickLabelPos with a position other than 'none' resulted in disabling the X data labels, and vice-versa. If both tick and data labels were positioned, the later setting overrode the earlier, which was turned off. Starting with PHPlot-5.1.0, PHPlot handles tick and data label positions as described in the notes above.

SetXDataLabelType

SetXDataLabelType — Set formatting type for X data labels

Synopsis

```
$plot->SetXDataLabelType($type, [...])
```

Description

SetXDataLabelType sets the formatting type for X data labels. By default, data labels are formatted the same as tick labels. If [SetXLabelType](#) is not used, then there is no special formatting for either label type, so the labels are output as-is. Available format types are 'data', 'time', 'printf', and 'custom'.

'data' formatting formats the labels as floating point numbers, with digits grouped into thousands (3 digit groups), and with user-defined precision. Grouping separator characters can be set with [SetNumberFormat](#). The precision (number of digits after the decimal point) can be set as an additional argument to SetXDataLabelType. A prefix and suffix string can also be specified.

'time' formatting formats the labels as date/time values, with the format string specified as an additional argument to SetXDataLabelType.

'printf' formatting formats the labels using the standard sprintf function, with the format string specified as an additional argument to SetXDataLabelType.

'custom' formatting formats the labels using a caller-provided function, with an optional pass-through argument. This provides the maximum flexibility in formatting labels.

Parameters

There is one required argument, $type. Other arguments depend on the value of the $type argument.

$type
> A string indicating the desired formatting mode: 'data', 'time', 'printf', or 'custom'. Or, an empty string meaning revert to no formatting.

For type 'data', there are three optional arguments:

$precision
> The formatting precision, or number of decimal places (optional). If omitted, the default is 1.

$prefix
> A prefix string to be placed before the formatted label values. This could be used for a currency symbol, for example. The default is an empty string.

$suffix
> A suffix string to be placed after the formatted label values. This could be used for a currency symbol, for example. The default is an empty string.

For type 'time', there is one optional argument:

$format
> Formatting string, used with strftime(). For example, '%Y-%m-%d' results in formatting a time_t value as a year, month, and day numbers. If omitted, the default is '%H:%M:%S' (hours, minutes, and seconds).

For type 'printf', there is one optional argument:

$format
> Formatting string, used with `sprintf()`. If omitted, the default value of '%e' uses scientific notation with default field sizes.

For type 'custom', there is one required argument and one optional argument:

$callback
> A callback function to format the label. This is either the name of a function (as a string), or a two-element array with an object instance and method name. (Refer to the PHP documentation for more information on the callback type.) The callback will be called with two arguments: the value of the label to be formatted, and the pass-through argument (see next).

$callback_arg
> A pass-through argument for the callback function. If omitted, NULL is used.

Notes

This function applies to both X axis data labels and X data value labels. See Section 3.6, "Labels" for more information on data labels.

The default formatting mode is to do no special formatting of the labels. Strings will be output as-is, and numbers will be output using PHP's default formatting. If you need to change label formatting back to the default, or to override a format type you set for tick labels and have no formatting for data labels, use SetXDataLabelType without arguments, or with an empty string argument.

When using a custom label formatting function, do not assume the labels are formatted in any particular order, or only once each.

Examples

See SetXLabelType.

History

This function was added in PHPlot-5.1.0. Through PHPlot-5.0.7, data labels and tick labels always used the same formatting, as set with SetXLabelType.

SetXLabelAngle

SetXLabelAngle — Set the text angle for X labels

Synopsis

```
$plot->SetXLabelAngle($xla)
```

Description

`SetXLabelAngle` sets the text angle for X tick labels. Unless [SetXDataLabelAngle](#) is called, the same angle is also used for X data labels. If using TrueType fonts, any angle can be used. If using built-in GD fonts, only 0 degree and 90 degree text can be used.

Parameters

$xla
    Desired angle for label text, in degrees.

Notes

The default text angle for X labels is 0 degrees, for horizontal text.

History

Through PHPlot-5.0.7, SetXLabelAngle sets the angle for both tick and data labels. Starting with PHPlot-5.1.0, these can be controlled independently using [SetXDataLabelAngle](#). For compatibility, data label angles default to the value set for tick label angles with SetXLabelAngle.

SetXLabelType

SetXLabelType — Set formatting type for X tick labels

Synopsis

```
$plot->SetXLabelType($type, [...])
```

Description

SetXLabelType sets the formatting type for X tick labels, and the default formatting type for X data labels. (If SetXDataLabelType is never called, SetXLabelType effectively sets the formatting type for both X tick labels and X data labels.) By default, there is no special formatting, so the labels are output as-is. Available format types are 'data', 'time', 'printf', and 'custom'.

'data' formatting formats the labels as floating point numbers, with digits grouped into thousands (3 digit groups), and with user-defined precision. Grouping separator characters can be set with SetNumberFormat. The precision (number of digits after the decimal point) can be set with SetPrecisionX, or as an additional argument to SetXLabelType. A prefix and suffix string can also be specified.

'time' formatting formats the labels as date/time values, using a format specifier set by SetXTimeFormat or using an additional argument to SetXLabelType.

'printf' formatting formats the labels using the standard sprintf function, with the format string specified as an additional argument to SetXLabelType.

'custom' formatting formats the labels using a caller-provided function, with an optional pass-through argument. This provides the maximum flexibility in formatting labels.

Parameters

There is one required argument, $type. Other arguments depend on the value of the $type argument.

$type
> A string indicating the desired formatting mode: 'data', 'time', 'printf', or 'custom'. Or, an empty string meaning revert to no formatting.

For type 'data', there are three optional arguments:

$precision
> The formatting precision, or number of decimal places (optional). If omitted, the value set with SetPrecisionX is used, or if that was never called then the default is 1.

$prefix
> A prefix string to be placed before the formatted label values. This could be used for a currency symbol, for example. The default is an empty string.

$suffix
> A suffix string to be placed after the formatted label values. This could be used for a currency symbol, for example. The default is an empty string.

For type 'time', there is one optional argument:

$format
>   Formatting string, used with `strftime()`. For example, '%Y-%m-%d' results in formatting a `time_t` value as a year, month, and day numbers. If omitted, the value set with [SetXTimeFormat](#) is used, or if that was never called then the default is '%H:%M:%S' (hours, minutes, and seconds).

For type 'printf', there is one optional argument:

$format
>   Formatting string, used with `sprintf()`. If omitted, the default value of '%e' uses scientific notation with default field sizes.

For type 'custom', there is one required argument and one optional argument:

$callback
>   A callback function to format the label. This is either the name of a function (as a string), or a two-element array with an object instance and method name. (Refer to the PHP documentation for more information on the callback type.) The callback will be called with two arguments: the value of the label to be formatted, and the pass-through argument (see next).

$callback_arg
>   A pass-through argument for the callback function. If omitted, NULL is used.

Notes

The default formatting mode is to do no special formatting of the labels. Strings will be output as-is, and numbers will be output using PHP's default formatting. If you need to change label formatting back to the default, use SetXLabelType without arguments, or with an empty string argument.

A side effect of [SetPrecisionX](#) is to call this function SetXLabelType and set the format type to 'data'. Note that [SetXTimeFormat](#) does not have a corresponding side effect on the format type.

When using a custom label formatting function, do not assume the labels are formatted in any particular order, or only once each.

Examples

The following table shows some label formatting examples. These also apply to `SetXLabelType`, [SetXDataLabelType](#), [SetYLabelType](#), and [SetYDataLabelType](#).

Code:	Value:	Result:
Data (numeric) formatting with two digits of precision. Grouping and decimal separators depend on locale.		
`$plot->SetXLabelType('data', 2);`	1234.56	1,234.56
Data (numeric) formatting with prefix. € is the entity code for the Euro sign in Unicode. (Numeric entity codes are handled by the GD library, but not named character entity codes.) Here we use it as a prefix, common usage for English. The Euro sign may appear differently in your browser. But when used with PHPlot it requires a Unicode font on the server.		
`$plot->SetXLabelType('data', 0, '&#8364;');`	100000	€1,000,000
Data (numeric) formatting with suffix. Here we use the Euro as a suffix, common usage for French. Unlike the previous example, here we represent it as a 3 byte UTF-8 sequence. You can use html_entity_decode() with UTF-8 as the character set to translate € into this sequence. (You cannot used named character entity codes in PHPlot strings that are processed by GD, as GD only handles UTF-8 sequences or numeric entity codes.) The thousands and		

Code:	Value:	Result:
decimal separator default to locale-dependent values, but here we set them ourselves with SetNumberFormat. The Euro sign may appear differently in your browser. But when used with PHPlot it requires a Unicode font on the server.		

`$plot->SetNumberFormat(',', '.');`
`$plot->SetXLabelType('data', 2, '',`
`                     '\xe2\x82\xac');`
	100000	1.000.000,00€

Date/time formatting. The given value is mktime(0,0,0,4,15,2008). The format string could be set with SetXTimeFormat instead.		
`$plot->SetXLabelType('time', '%m/%Y');`	1208232000	04/2008
Formatting using printf. Note PHP printf may differ from the standard C library. For example, PHP outputs only a one digit exponent here.		
`$plot->SetXLabelType('printf', '%8.2e');`	1234	1.23e+3
A custom formatting function is used to format values in decimal degrees as degrees, minutes, and seconds. (This only works for non-negative angles.)		

`function deg_min_sec($value)`
`{`
`  $deg = (int)$value;`
`  $value = ($value - $deg) * 60;`
`  $min = (int)$value;`
`  $sec = (int)(($value - $min) * 60);`
`  return "{$deg}d {$min}m {$sec}s";`
`}`
`$plot->SetXLabelType('custom', 'deg_min_sec');`
	75.12345	75d 7m 24s

History

Through PHPlot-5.0.7, this function set the format type for both X tick labels and X data labels. Starting with PHPlot-5.1.0, a new function SetXDataLabelType was added to allow separate control of tick and data labels. `SetXLabelType` now sets the format type for X tick labels, and the default format type for X data labels.

New label format types 'printf' and 'custom' were added at PHPlot-5.0.6, as well as all arguments after the first. In PHPlot-5.0.5 and earlier, you must use SetXTimeFormat and SetPrecisionX to set the formatting parameters. Starting with PHPlot-5.0.6, you have the choice of using those, or providing additional arguments to SetXLabelType. Also added was the ability to add a prefix and suffix to 'data' formatted labels. In PHPlot-5.0.5 and earlier, there was an undocumented class variable *data_units_text* that was applied as a suffix to 'data' mode labels, for both X and Y. This continues to work, but is deprecated.

Starting with PHPlot-5.0.6, you can use an empty string or no argument at all to reset to the default of no formatting.

Starting with PHPlot-5.0.4, empty string data labels are ignored when formatting with 'data' or 'time' formats. You can use this to suppress some data labels, or control label density, with 'data' and 'time' formatted labels.

Through PHPlot-5.0rc3, empty strings would still be formatted. With 'data' format, an empty string would result in a zero value, and with 'time' format an empty string would cause an error. As a result, with older releases, if you don't want to use data labels when using 'data' or 'time' formats, you must turn off X data label display with SetXDataLabelPos, even if your data array labels are empty strings.

Through PHPlot-5.0rc3, when the formatting mode is 'data' the thousands grouping separator was always a comma, and a period was used as a decimal point. Starting with 5.0.4, PHPlot attempts to get the correct values for your locale. You can set the separator characters yourself instead with SetNumberFormat.

SetXScaleType

SetXScaleType — Select linear or logarithmic scale

Synopsis

```
$plot->SetXScaleType($st)
```

Description

`SetXScaleType` sets the scale type along the X axis to be either linear (the default) or logarithmic.

Parameters

$st
> A string specifying the scale type: 'linear' or 'log'.

Notes

No X value may be less than or equal to 0 with logarithmic X scale.

Support for logarithmic scales in PHPlot is limited. One problem is that tick interval is fixed for the entire range of data, which is inappropriate for logarithmic scales where the data spans more than one magnitude.

The default X and Y scale types are linear.

SetXTickCrossing

SetXTickCrossing — Set crossing length of X tick marks

Synopsis

```
$plot->SetXTickCrossing($xc)
```

Description

`SetXTickCrossing` sets the length by which the X tick marks cross the X axis or plot border (depending on the tick position set with [SetXTickPos](#)) pointing inwards. See figure below.

Parameters

$xc
   Desired X tick crossing length in pixels.

Notes

The default tick crossing length is 3 pixels.

The following figure shows the four length measurements used to draw the tick marks. (In this plot, the tick lengths have been increased from the defaults.)
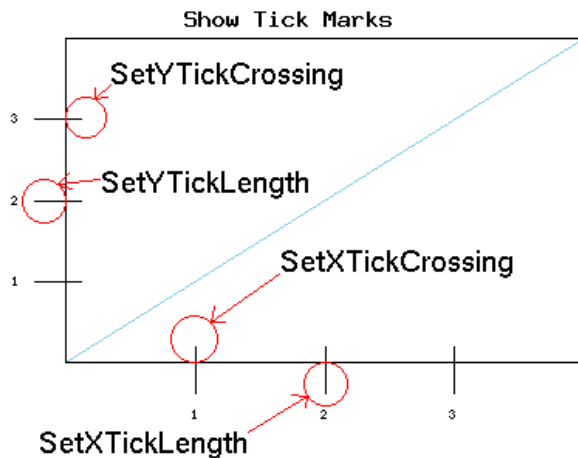
SetXTickIncrement

SetXTickIncrement — Set the step between X ticks

Synopsis

```
$plot->SetXTickIncrement([$ti])
```

Description

`SetXTickIncrement` sets the interval between X tick marks. You can use either this function or [SetNumXTicks](but not both) to control the tick mark spacing.

Parameters

$ti
    Desired tick interval, in world coordinates. If the value is omitted or an empty string, the default behavior is restored.

Notes

If neither `SetXTickIncrement` nor [SetNumXTicks](#) is used, the tick mark interval is calculated as 1/10th of the X data range.

SetXTickLabelPos

SetXTickLabelPos — Position the X tick labels

Synopsis

```
$plot->SetXTickLabelPos($xtlp)
```

Description

`SetXTickLabelPos` determines where (and if) the X tick labels are drawn. The labels can be drawn at the bottom of the plot, above the plot, in both positions, at the X axis (even if it is in the middle of the plot), or not drawn at all.

Parameters

$xtlp
>   A string indicating the desired position for the X tick labels:

Position	Description
plotdown	Tick labels below the plot
plotup	Tick labels above the plot
both	Tick labels both below and above the plot
xaxis	Tick labels at X axis (even if the axis is in the middle of the plot)
none	No tick labels

Notes

The default position for the X tick labels is chosen to avoid overlapping tick and data labels. For vertical plots, the X tick labels will default to 'none' if the X data labels have been enabled using SetXDataLabelPos. If neither SetXTickLabelPos nor SetXDataLabelPos are used, PHPlot will enable just data labels if they are non-empty, and otherwise it will enable just tick labels, and position them below the plot. For horizontal plots, there is no conflict (because data labels are drawn along Y), so the tick label position defaults to 'plotdown'.

This applies only to tick labels. For data labels, see SetXDataLabelPos. You may want the tick marks to be in the same position as the tick labels. To position the tick marks, see SetXTickPos.

See SetXAxisPosition for positioning the X axis.

History

Through PHPlot-5.0.7, the default position for X tick labels was below the plot ('plotdown'). This would result in overlaid data and tick labels by default. In addition, positioning the X data labels with SetXDataLabelPos with a position other than 'none' resulted in disabling the X tick labels, and vice-versa. If both tick and data labels were positioned, the later setting overrode the earlier, which was turned off. Starting with PHPlot-5.1.0, PHPlot handles tick and data label positions as described in the notes above.

SetXTickLength

SetXTickLength — Set outer length of X tick marks

Synopsis

```
$plot->SetXTickLength($xln)
```

Description

`SetXTickLength` sets the length of the X tick marks pointing outwards from the plot. For example, for tick marks on an X axis at the bottom of the plot, this is the length from the X axis down.

Parameters

$xln
    Desired X tick length in pixels.

Notes

The default tick length is 5 pixels.

See figure under [SetXTickCrossing](#).

SetXTickPos

SetXTickPos — Position the X tick marks

Synopsis

```
$plot->SetXTickPos($tp)
```

Description

SetXTickPos determines where (and if) the X tick marks are drawn. The tick marks can be drawn at the bottom of the plot, above the plot, in both positions, at the X axis (even if it is in the middle of the plot), or not drawn at all.

Parameters

$xtp
>A string indicating the desired position for the X tick marks:

Position	Description
plotdown	Tick marks below the plot
plotup	Tick marks above the plot
both	Tick marks both below and above the plot
xaxis	Tick marks at X axis (even if the axis is in the middle of the plot)
none	No tick marks

Notes

The default position for the X tick marks is below the plot.

This applies only to tick marks. You may want the tick labels to be in the same positions as the tick marks. To position the tick labels, see SetXTickLabelPos.

See SetXAxisPosition for positioning the X axis.

SetXTimeFormat

SetXTimeFormat — Set date/time formatting string for X labels

Synopsis

```
$plot->SetXTimeFormat($xtf)
```

Description

SetXTimeFormat sets the formatting string for X tick and data labels when 'time' formatting mode for X labels is in effect. Use SetXLabelType to select the formatting mode for labels. The formatting string is used with the PHP strftime to format labels as date/time strings.

Note

This function is retained for compatibility, but use of SetXLabelType is preferred.

Parameters

$xtf
Formatting string for X labels, used with strftime(). For example, if the label value is 1104534000 (which is the time_t representation of 6:00 PM on the last day of 2004), '%Y-%m-%d.%H:%M:%S' results in '2004-12-31.18:00:00', and '%d %b %Y' results in '31 Dec 2004'.

Notes

This applies to X tick labels, and also to X data labels unless overridden by SetXDataLabelType.

To use date/time formatting, the label values must be Unix time_t values (number of seconds since Unix epoch).

Unlike SetPrecisionX, SetXTimeFormat does not automatically enable the correct label formatting mode. You must call SetXLabelType('time') to use date/time formatting of labels.

The default time format is '%H:%M:%S', showing hours, minutes, and seconds (and ignoring any date information).

History

Starting with PHPlot-5.0.6, the time format can be set with SetXLabelType instead.

The default time format was undefined prior to PHPlot-5.0rc3.

SetXTitle

SetXTitle — Sets the X axis title, and optionally its position

Synopsis

```
$plot->SetXTitle($xtitle, [$xpos])
```

Description

SetXTitle sets the text to be displayed as the X axis title. Optionally, it also sets the position of the title and the axis itself: below the graph (the usual place), above the graph, both, or neither.

Parameters

$xtitle
>   The text string to use for the X axis title. The string can contain multiple lines, separated by newlines (in PHP: "\n").

$xpos
>   Optional position for the X axis and title. Use one of the following strings for the position:

Position	Description
plotdown	X axis below the plot
plotup	X axis above the plot
both	One X axis above, and one below
none	No X axis, no X axis title

>   The default is 'plotdown'.

Notes

By default, there is no X axis title. If SetXTitle is called with an empty string as the title, the default behavior is restored. This includes not leaving space on the graph for the title.

SetXTitleColor

SetXTitleColor — Set the color of the X Title

Synopsis

```
$plot->SetXTitleColor($color)
```

Description

SetXTitleColor sets the color of the X title (as set with SetXTitle).

Parameters

$color
    Color value to use. See Section 3.5, "Colors" for more on color values.

Notes

Use this function if you want the X title to have a different color than the main title. See Section 3.7.1, "Titles" for more about plot titles.

By default, the X title defaults to use the same color as the main plot title. The main plot title color is set with SetTitleColor, and it defaults to black.

History

This function was added in PHPlot-5.2.0. Through PHPlot-5.1.3, the main, X, and Y titles always used the same color.

SetYAxisPosition

SetYAxisPosition — Move the Y axis

Synopsis

`$plot->SetYAxisPosition($pos)`

Description

`SetYAxisPosition` sets the position of the Y axis.

Parameters

$pos
> The X position in world coordinates for the Y axis. (World coordinates are the coordinate space of your data points.) If the value is omitted or an empty string, the default behavior is restored.

Notes

The given position is truncated (towards 0) to an integer value.

The default axis position differs for vertical and horizontal plots. For vertical plots, the Y axis position defaults to the left side of the plot. For horizontal plots, the Y axis position defaults to X=0, provided X=0 is within the range of the graph. If X=0 is not within the range of the graph, the Y axis position will default to the edge with the smallest absolute X value. This means the Y axis will be on the left side of the graph if all values of X are greater than zero, and on the right side of the graph if all values of X are less than zero. (For log scale plots, however, the default Y axis position is X=1.)

History

Through PHPlot-5.3.0, the argument was required, and there was no way to restore the default behavior (because the argument value was always converted to an integer). Starting with PHPlot-5.3.1, the argument may be given as an empty string, or omitted, to restore the default behavior.

Through PHPlot-5.1.3, there was no special handling for horizontal plots. The Y axis always defaulted to the left side of the plot. When plotting negative data with horizontal plots, it was usually necessary to use `SetYAxisPosition(0)` to force the bars to start from X=0.

SetYDataLabelAngle

SetYDataLabelAngle — Set the text angle for Y data labels

Synopsis

```
$plot->SetYDataLabelAngle($ydla)
```

Description

`SetYDataLabelAngle` sets the text angle for Y data labels. If using TrueType fonts, any angle can be used. If using built-in GD fonts, only 0 degree and 90 degree text can be used.

Parameters

$ydla
    Desired angle for label text, in degrees.

Notes

This function applies to both Y axis data labels and Y data value labels. See Section 3.6, "Labels" for more information on data labels.

By default, Y data labels are drawn at 0 degrees. (This is different from X data labels, which default to the angle set for X tick labels.)

History

This function was added in PHPlot-5.1.0. Through PHPlot-5.0.7, Y data labels were always drawn at 0 degrees.

SetYDataLabelPos

SetYDataLabelPos — Position and control Y data labels

Synopsis

```
$plot->SetYDataLabelPos($ydlp)
```

Description

`SetYDataLabelPos` determines if and where Y data labels are drawn. For horizontal plots, these are Y axis data labels, which display the label strings from your data array. The labels can be drawn at the left side of the plot (left of the Y axis), on the right side, in both positions, or neither. For vertical plots, these are Y data value labels, displaying the value of the data point within the plot area.

Parameters

$ydlp
A string indicating the desired position for the Y data labels:

Position	Description
plotleft	Data labels left of the plot. This is for horizontal plots.
plotright	Data labels right of the plot. This is for horizontal plots.
both	Data labels both left and right of the plot. This is for horizontal plots.
plotin	Data value labels within the plot area. For bar charts, this displays the value above (or below) each bar. For stacked bar charts, this displays only the bar total labels and not the bar segment labels. For other plot types, see notes below.
plotstack	Data value labels above (or below) each bar, and below the top of each bar segment. This is valid for vertical stacked bar plots only. It turns on both bar total labels (as with 'plotin') and bar segment labels.
none	No data labels or data value labels

Notes

With vertical plots, this function controls the Y data value labels. For examples, see Example 5.19, "Bar Chart with Data Value Labels", Example 5.20, "Stacked Bars with Y Data Value Labels", and Example 5.33, "Linepoints Plot with Data Value Labels". With horizontal plots, this function controls the Y axis data labels. For examples, see Example 5.27, "Horizontal Bar Chart" and Example 5.28, "Horizontal Stacked Bar Chart", where the labels are enabled by default and displayed to the left of the Y axis.

The default position for Y data labels (for horizontal plots) is left of the plot. However, PHPlot only enables the data labels if SetYDataLabelPos was used to position them, or if SetYTickLabelPos was not used to enable the tick labels and the data labels are not all empty.

The default position for Y data value labels (for vertical plots) is 'none', meaning no labels. For plot types 'bars' and 'stackedbars', Y data value labels are drawn (if enabled) at fixed positions within or above/below the bars. For plot types 'lines', 'points', 'linepoints', and 'squared', Y data labels are drawn (if enabled) above the data points by default.

The position can be changed (see Section 4.5.4, "Label Tuning"), but PHPlot does not attempt to prevent interference between the labels and other plot elements. Y data value labels are not available with other plot types.

For tick labels, see SetYTickLabelPos.

The bar segment labels, if enabled ('plotstack'), are drawn inside the bars and may not be very visible if dark colors are used for the bar fill. Bar segment labels will be omitted for segments which are too narrow.

Y data value labels will be drawn below the bars for negative values. This only applies to vertical bar charts. Stacked bar charts are not allowed to have negative values.

The Y data label text angle is set with SetYDataLabelAngle. The Y data label text format can be controlled with SetYDataLabelType or SetYLabelType.

History

Through PHPlot-5.2.0, data value labels were only available for bars and stackedbars plot types. Starting with PHPlot-5.3.0, data value labels are also implemented for lines, points, linepoints, and squared plot types.

Horizontal plot types were added in PHPlot-5.1.2 and PHPlot-5.1.3, and the Y Data Labels were extended to include the Y axis data labels for horizontal plots in addition to the data value labels for vertical plots.

The data value label feature for stacked bar graphs was added in PHPlot-5.1.1.

The data value label feature for bar graphs was added to PHPlot-5.0rc3.

SetYDataLabelType

SetYDataLabelType — Set formatting type for Y data labels

Synopsis

```
$plot->SetYDataLabelType($type, [...])
```

Description

SetYDataLabelType sets the formatting type for Y data labels. By default, data labels are formatted the same as tick labels. If [SetYLabelType](#) is not used, then there is no special formatting for either label type, so the labels are output as-is. Available format types are 'data', 'time', 'printf', and 'custom'.

'data' formatting formats the labels as floating point numbers, with digits grouped into thousands (3 digit groups), and with user-defined precision Grouping separator characters can be set with [SetNumberFormat](#). The precision (number of digits after the decimal point) can be set as an additional argument to SetYDataLabelType. A prefix and suffix string can also be specified.

'time' formatting formats the labels as date/time values, with the format string specified as an additional argument to SetYDataLabelType.

'printf' formatting formats the labels using the standard sprintf function, with the format string specified as an additional argument to SetYDataLabelType.

'custom' formatting formats the labels using a caller-provided function, with an optional pass-through argument. This provides the maximum flexibility in formatting labels.

Parameters

There is one required argument, $type. Other arguments depend on the value of the $type argument.

$type
   A string indicating the desired formatting mode: 'data', 'time', 'printf', or 'custom'. Or, an empty string meaning revert to no formatting.

For type 'data', there are three optional arguments:

$precision
   The formatting precision, or number of decimal places (optional). If omitted, the default is 1.

$prefix
   A prefix string to be placed before the formatted label values. This could be used for a currency symbol, for example. The default is an empty string.

$suffix
   A suffix string to be placed after the formatted label values. This could be used for a currency symbol, for example. The default is an empty string.

For type 'time', there is one optional argument:

$format
   Formatting string, used with strftime(). For example, '%Y-%m-%d' results in formatting a time_t value as a year, month, and day numbers. If omitted, the default is '%H:%M:%S' (hours, minutes, and seconds).

For type 'printf', there is one optional argument:

$format
> Formatting string, used with `sprintf()`. If omitted, the default value of '%e' uses scientific notation with default field sizes.

For type 'custom', there is one required argument and one optional argument:

$callback
> A callback function to format the label. This is either the name of a function (as a string), or a two-element array with an object instance and method name. (Refer to the PHP documentation for more information on the callback type.) The callback will be called with two arguments: the value of the label to be formatted, and the pass-through argument (see next).

$callback_arg
> A pass-through argument for the callback function. If omitted, NULL is used.

Notes

This function applies to both Y axis data labels and Y data value labels. See Section 3.6, "Labels" for more information on data labels.

The default formatting mode is to do no special formatting of the labels. Strings will be output as-is, and numbers will be output using PHP's default formatting. If you need to change label formatting back to the default, or to override a format type you set for tick labels and have no formatting for data labels, use SetYDataLabelType without arguments, or with an empty string argument.

When using a custom label formatting function, do not assume the labels are formatted in any particular order, or only once each.

Examples

See SetXLabelType.

History

This function was added in PHPlot-5.1.0. Through PHPlot-5.0.7, data labels and tick labels always used the same formatting, as set with SetYLabelType.

SetYLabelAngle

SetYLabelAngle — Set the text angle for Y tick labels

Synopsis

```
$plot->SetYLabelAngle($yla)
```

Description

`SetYLabelAngle` sets the text angle for Y tick labels. If using TrueType fonts, any angle can be used. If using built-in GD fonts, only 0 degree and 90 degree text can be used.

Parameters

$yla
    Desired angle for label text, in degrees.

Notes

The default text angle for Y labels is 0 degrees, for horizontal text.

This does not apply to Y data labels. For those, see SetYDataLabelAngle.

SetYLabelType

SetYLabelType — Set formatting type for Y tick labels

Synopsis

```
$plot->SetYLabelType($type, [...])
```

Description

`SetYLabelType` sets the formatting type for Y tick labels, and the default formatting type for Y data labels. (If [SetYDataLabelType](#) is never called, `SetYLabelType` effectively sets the formatting type for both Y tick labels and Y data labels.) By default, there is no special formatting, so the labels are output as-is. Available format types are 'data', 'time', 'printf', and 'custom'.

'data' formatting formats the labels as floating point numbers, with digits grouped into thousands (3 digit groups), and with user-defined precision Grouping separator characters can be set with [SetNumberFormat](#). The precision (number of digits after the decimal point) can be set with [SetPrecisionY,](#) or as an additional argument to SetYLabelType. A prefix and suffix string can also be specified.

'time' formatting formats the labels as date/time values, using a format specifier set by [SetYTimeFormat](#) or using an additional argument to SetYLabelType.

'printf' formatting formats the labels using the standard `sprintf` function, with the format string specified as an additional argument to SetYLabelType.

'custom' formatting formats the labels using a caller-provided function, with an optional pass-through argument. This provides the maximum flexibility in formatting labels.

Parameters

There is one required argument, $type. Other arguments depend on the value of the $type argument.

$type
   A string indicating the desired formatting mode: 'data', 'time', 'printf', or 'custom'. Or, an empty string meaning revert to no formatting.

For type 'data', there are three optional arguments:

$precision
   The formatting precision, or number of decimal places (optional). If omitted, the value set with [SetPrecisionY](#) is used, or if that was never called then the default is 1.

$prefix
   A prefix string to be placed before the formatted label values. This could be used for a currency symbol, for example. The default is an empty string.

$suffix
   A suffix string to be placed after the formatted label values. This could be used for a currency symbol, for example. The default is an empty string.

For type 'time', there is one optional argument:

$format
> Formatting string, used with `strftime()`. For example, '%Y-%m-%d' results in formatting a `time_t` value as a year, month, and day numbers. If omitted, the value set with [SetYTimeFormat](#) is used, or if that was never called then the default is '%H:%M:%S' (hours, minutes, and seconds).

For type 'printf', there is one optional argument:

$format
> Formatting string, used with `sprintf()`. If omitted, the default value of '%e' uses scientific notation with default field sizes.

For type 'custom', there is one required argument and one optional argument:

$callback
> A callback function to format the label. This is either the name of a function (as a string), or a two-element array with an object instance and method name. (Refer to the PHP documentation for more information on the callback type.) The callback will be called with two arguments: the value of the label to be formatted, and the pass-through argument (see next).

$callback_arg
> A pass-through argument for the callback function. If omitted, NULL is used.

Notes

The default formatting mode is to do no special formatting of the labels. Strings will be output as-is, and numbers will be output using PHP's default formatting. If you need to change label formatting back to the default, use SetYLabelType without arguments, or with an empty string argument.

A side effect of [SetPrecisionY](#) is to call this function SetYLabelType and set the format type mode to 'data'. Note that [SetYTimeFormat](#) does not have a corresponding side effect on the format type.

When using a custom label formatting function, do not assume the labels are formatted in any particular order, or only once each.

Examples

See [SetXLabelType](#).

History

Through PHPlot-5.0.7, this function set the format type for both Y tick labels and Y data labels. Starting with PHPlot-5.1.0, a new function [SetYDataLabelType](#) was added to allow separate control of tick and data labels. `SetYLabelType` now sets the format type for Y tick labels, and the default format type for Y data labels.

New label format types 'printf' and 'custom' were added at PHPlot-5.0.6, as well as all arguments after the first. In PHPlot-5.0.5 and earlier, you must use SetYTimeFormat and SetPrecisionY to set the formatting parameters. Starting with PHPlot-5.0.6, you have the choice of using those, or providing additional arguments to SetYLabelType. Also added was the ability to add a prefix and suffix to 'data' formatted labels. In PHPlot-5.0.5 and earlier, there was an undocumented class variable *data_units_text* that was applied as a suffix to 'data' mode labels, for both X and Y. This continues to work, but is deprecated.

Starting with PHPlot-5.0.6, you can use an empty string or no argument at all to reset to the default of no formatting.

Through PHPlot-5.0rc3, when the formatting mode is 'data' the thousands grouping separator was always a comma, and a period was used as a decimal point. Starting with 5.0.4, PHPlot attempts to get the correct values for your locale. You can set the separator characters yourself instead with [SetNumberFormat](#).

SetYScaleType

SetYScaleType — Select linear or logarithmic scale

Synopsis

```
$plot->SetYScaleType($st)
```

Description

`SetYScaleType` sets the scale type along the Y axis to be either linear (the default) or logarithmic.

Parameters

$st
> A string specifying the scale type: 'linear' or 'log'.

Notes

No Y value may be less than or equal to 0 with logarithmic Y scale.

Support for logarithmic scales in PHPlot is limited. One problem is that tick interval is fixed for the entire range of data, which is inappropriate for logarithmic scales where the data spans more than one magnitude.

The default X and Y scale types are linear.

SetYTickCrossing

SetYTickCrossing — Set crossing length of Y tick marks

Synopsis

```
$plot->SetYTickCrossing($yc)
```

Description

`SetYTickCrossing` sets the length by which the Y tick marks cross the Y axis or plot border (depending on the tick position set with [SetYTickPos](#)) pointing inwards.

Parameters

$yc
    Desired X tick crossing length in pixels.

Notes

The default tick crossing length is 3 pixels.

See figure under [SetXTickCrossing](#).

SetYTickIncrement

SetYTickIncrement — Set the step between Y ticks

Synopsis

```
$plot->SetYTickIncrement([$ti])
```

Description

`SetYTickIncrement` sets the interval between Y tick marks. You can use either this function or [SetNumYTicks](but not both) to control the tick mark spacing.

Parameters

$ti
    Desired tick interval, in world coordinates. If the value is omitted or an empty string, the default behavior is restored.

Notes

If neither `SetYTickIncrement` nor [SetNumYTicks](link) is used, the tick mark interval is calculated as 1/10th of the Y data range.

SetYTickLabelPos

SetYTickLabelPos — Position the Y tick labels

Synopsis

```
$plot->SetYTickLabelPos($ytlp)
```

Description

`SetYTickLabelPos` determines where (and if) the Y tick labels are drawn. The labels can be drawn on the left side of the plot, on the right side of the plot, in both positions, at the Y axis (even if it is in the middle of the plot), or not drawn at all.

Parameters

$ytlp
> A string indicating the desired position for the Y tick labels:

Position	Description
plotleft	Tick labels on the left side of the plot
plotright	Tick labels on the right side of the plot
both	Tick labels on both left and right sides of the plot
yaxis	Tick labels at Y axis (even if the axis is in the middle of the plot)
none	No tick labels

Notes

The default position for the Y tick labels is chosen to avoid overlapping tick and data labels. For vertical plots, there is no conflict (because data labels are drawn along X), so the tick label position defaults to 'plotleft'. For horizontal plots, the Y tick labels will default to 'none' if the Y data labels have been enabled using SetYDataLabelPos. If neither SetYTickLabelPos nor SetYDataLabelPos are used, PHPlot will enable just data labels if they are non-empty, and otherwise it will enable just tick labels, and position them to the left of the plot.

This applies only to tick labels. For data labels, see SetYDataLabelPos. You may want the tick marks to be in the same position as the tick labels. To position the tick marks, see SetYTickPos.

See SetYAxisPosition for positioning the Y axis.

SetYTickLength

SetYTickLength — Set outer length of Y tick marks

Synopsis

```
$plot->SetYTickLength($yln)
```

Description

`SetYTickLength` sets the length of the Y tick marks pointing outwards from the plot. For example, for tick marks on a Y axis on the left side of the plot, this is the length from the axis to the left.

Parameters

$yln
    Desired Y tick length in pixels.

Notes

The default tick length is 5 pixels.

See figure under SetXTickCrossing.

SetYTickPos

SetYTickPos — Position the Y tick marks

Synopsis

```
$plot->SetYTickPos($tp)
```

Description

SetYTickPos determines where (and if) the Y tick marks are drawn. The tick marks can be drawn on the left side of the plot, on the right side the plot, in both positions, at the Y axis (even if it is in the middle of the plot), or not drawn at all.

Parameters

$tp

A string indicating the desired position for the Y tick marks:

Position	Description
plotleft	Tick marks on the left side of the plot
plotright	Tick marks on the right side of the plot
both	Tick marks on both left and right sides of the plot
yaxis	Tick marks at Y axis (even if the axis is in the middle of the plot)
none	No tick marks

Notes

The default position for the Y tick marks is on the left side of the plot.

This applies only to tick marks. You may want the tick labels to be in the same positions as the tick marks. To position the tick labels, see SetYTickLabelPos.

See SetYAxisPosition for positioning the Y axis.

SetYTimeFormat

SetYTimeFormat — Set date/time formatting string for Y labels

Synopsis

```
$plot->SetYTimeFormat($ytf)
```

Description

SetYTimeFormat sets the formatting string for Y tick and data labels when 'time' formatting mode for Y labels is in effect. (Y data labels are only available with bar charts and stacked bar charts.) Use [SetYLabelType](#) to select the formatting mode for labels. The formatting string is used with the PHP strftime to format labels as date/time strings.

Note

This function is retained for compatibility, but use of [SetYLabelType](#) is preferred.

Parameters

$ytf
Formatting string for Y labels, used with strftime(). For example, if the label value is 1104534000 (which is the time_t representation of 6:00 PM on the last day of 2004), '%Y-%m-%d.%H:%M:%S' results in '2004-12-31.18:00:00', and '%d %b %Y' results in '31 Dec 2004'.

Notes

This applies to Y tick labels, and also to Y data labels unless overridden by [SetYDataLabelType](#).

To use date/time formatting, the label values must be Unix time_t values (number of seconds since Unix epoch).

Unlike [SetPrecisionY](#), SetYTimeFormat does not automatically enable the correct label formatting mode. You must call SetYLabelType('time') to use date/time formatting of labels.

The default time format is '%H:%M:%S', showing hours, minutes, and seconds (and ignoring any date information).

History

Starting with PHPlot-5.0.6, the time format can be set with [SetYLabelType](#) instead.

The default time format was undefined prior to PHPlot-5.0rc3.

SetYTitle

SetYTitle — Sets the Y axis title, and optionally its position

Synopsis

```
$plot->SetYTitle($ytitle, [$ypos])
```

Description

SetYTitle sets the text to be displayed as the Y axis title. Optionally, it also sets the position of the title and the axis itself: on the left side of the graph (the usual place), on the right side, both, or neither.

Parameters

$ytitle
> The text string to use for the Y axis title. The string can contain multiple lines, separated by newlines (in PHP: "\n").

$ypos
> Optional position for the Y axis and title. Use one of the following strings for the position:

Position	Description
plotleft	Y axis on the left side of the plot
plotright	Y axis on the right side of the plot
both	One Y axis on the left, and one on the right
none	No Y axis, no Y axis title

> The default is 'plotleft'.

Notes

By default, there is no Y axis title. If SetYTitle is called with an empty string as the title, the default behavior is restored. This includes not leaving space on the graph for the title.

SetYTitleColor

SetYTitleColor — Set the color of the Y Title

Synopsis

```
$plot->SetYTitleColor($color)
```

Description

SetYTitleColor sets the color of the Y title (as set with [SetYTitle](#)).

Parameters

$color
    Color value to use. See [Section 3.5, "Colors"](#) for more on color values.

Notes

Use this function if you want the Y title to have a different color than the main title. See [Section 3.7.1, "Titles"](#) for more about plot titles.

By default, the Y title defaults to use the same color as the main plot title. The main plot title color is set with [SetTitleColor](#), and it defaults to black.

History

This function was added in PHPlot-5.2.0. Through PHPlot-5.1.3, the main, X, and Y titles always used the same color.

Part III. Developer's Guide to PHPlot

This part of the PHPlot manual is about PHPlot internals, and various technical details which are of interest mostly to developers of PHPlot itself.

Chapter 7. PHPlot Plot Layout

This figure illustrates the plot layout, starting with PHPlot-5.0.5 when the margin calculations were rewritten. Use the following notes to help understand the figure.

- A vertical plot is shown. The X and Y coordinate system is the same for horizontal plots, but the roles are reversed: Y is the independent variable, and X is the dependent variable.

- The safe_margin (box with X) is a gap of 5 pixels used to separate elements.

- The main title, X titles, and Y titles are optional. If these titles are not set, the space allocated to them collapses, including the safe_margin gap. This is shown in the figure with an arrow.

- Similarly, the X tick labels, X axis data labels (for vertical plots), Y tick labels, and Y axis data labels (for horizontal plots) are optional. If these labels are not used, the space allocated to them collapses, including the safe_margin.

- For vertical plots, X axis data labels and X tick labels occupy the same space. For horizontal plots, Y axis data labels and Y tick labels occupy the same space. Normally only one of these should be present; if both are on for some reason they will overlay.

- The base_margin (pink box with plus sign) is the space between the image edge and the outermost graphics or text. If there is no image border, or the image border is no wider than 2 pixels (which is always the case through PHPlot-5.1.1), then the base_margin is set to the same as the safe_margin value (5 pixels). A wider image border increases the base_margin; for example an 8 pixel image border results in a base_margin of 11 pixels.

- The top, left, right, and bottom margins consist of base_margin plus whatever space is needed for titles and ticks. There is a minimum value for these margins - 2 times safe_margin plus base_margin. Even if there are no titles at all on a side, the margin on that side will be no less than this minimum margin. This keeps the axis or plot area edge from coming too close to the image edge. The calculated plot area margins shown can be overridden by using SetPlotAreaPixels or SetMarginsPixels.

- The main title is positioned relative to the top of the image. Starting with PHPlot-5.0.5, the X and Y titles are positioned relative to the plot area. PHPlot-5.0.4 and earlier positioned the X and Y titles relative to the image edges. The difference should not be visible with automatic margins, but if the margins are set larger, or the plot area smaller, then the X and Y titles will move inwards with the newer releases.

- It is possible to position X ticks and X tick labels to xaxis, and Y ticks and Y tick labels to yaxis, so the ticks and/or labels will float to the axis positions rather than always appear on the edges of the plot area. Space will be allocated for the corresponding margin only if the axis happens to fall exactly at the data limits for that side. This may cause problems if the axis is close but not quite at the edge; no margin space will be allocated on that side for the labels, and the labels may overlap the title or fall off the edge of the image.

ljb 2010–05–29
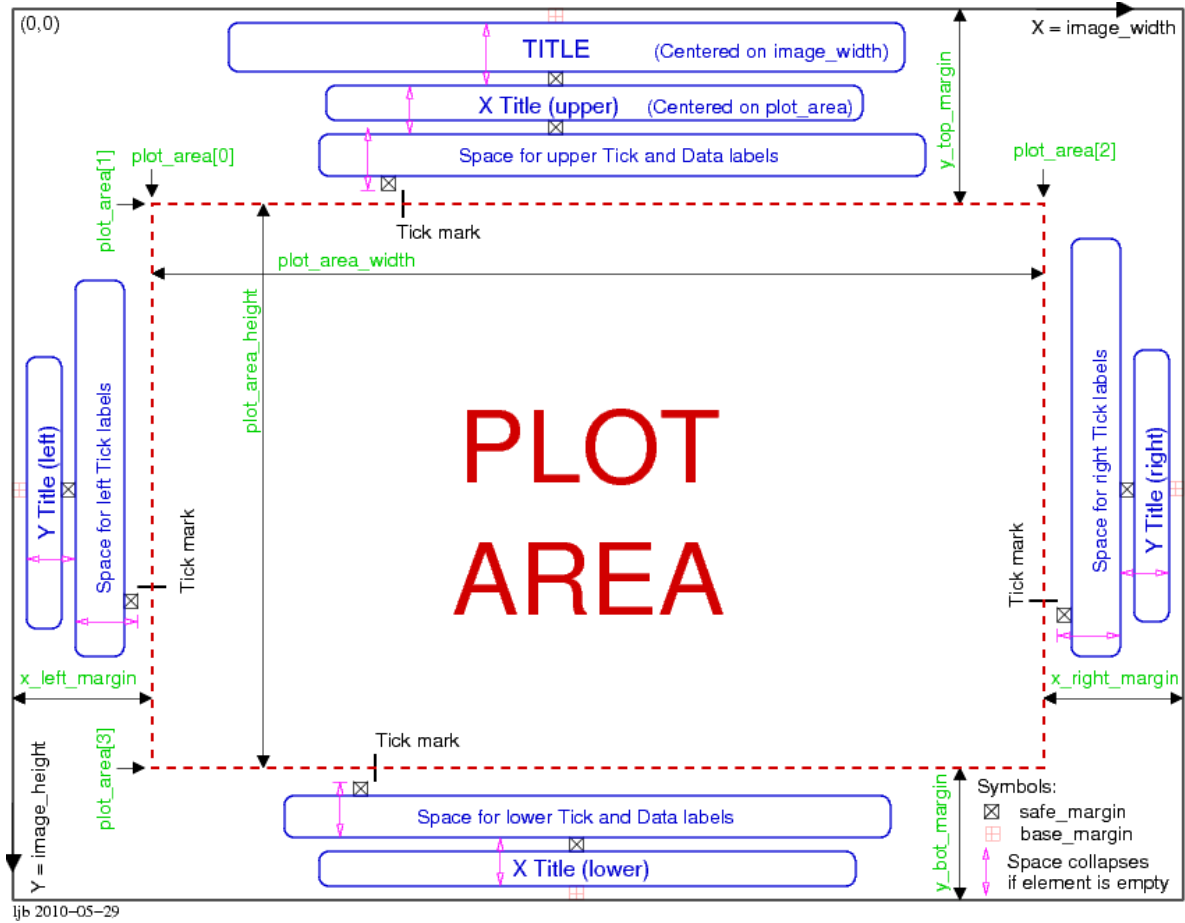
Chapter 8. PHPlot Legend Layout

This figure illustrates the layout of the legend, as drawn by DrawLegend(). Two of six possible arrangements of the colorboxes and text are shown. See SetLegendStyle for more information.

Through PHPlot-5.2.0, the width of the colorboxes was always char_w, the width of one character (or the width of the upper case letter "E") in the legend text font. Starting with PHPlot-5.3.0, this can be adjusted using a class variable. See Section 4.5.3, "Tuning the Legend" for more.
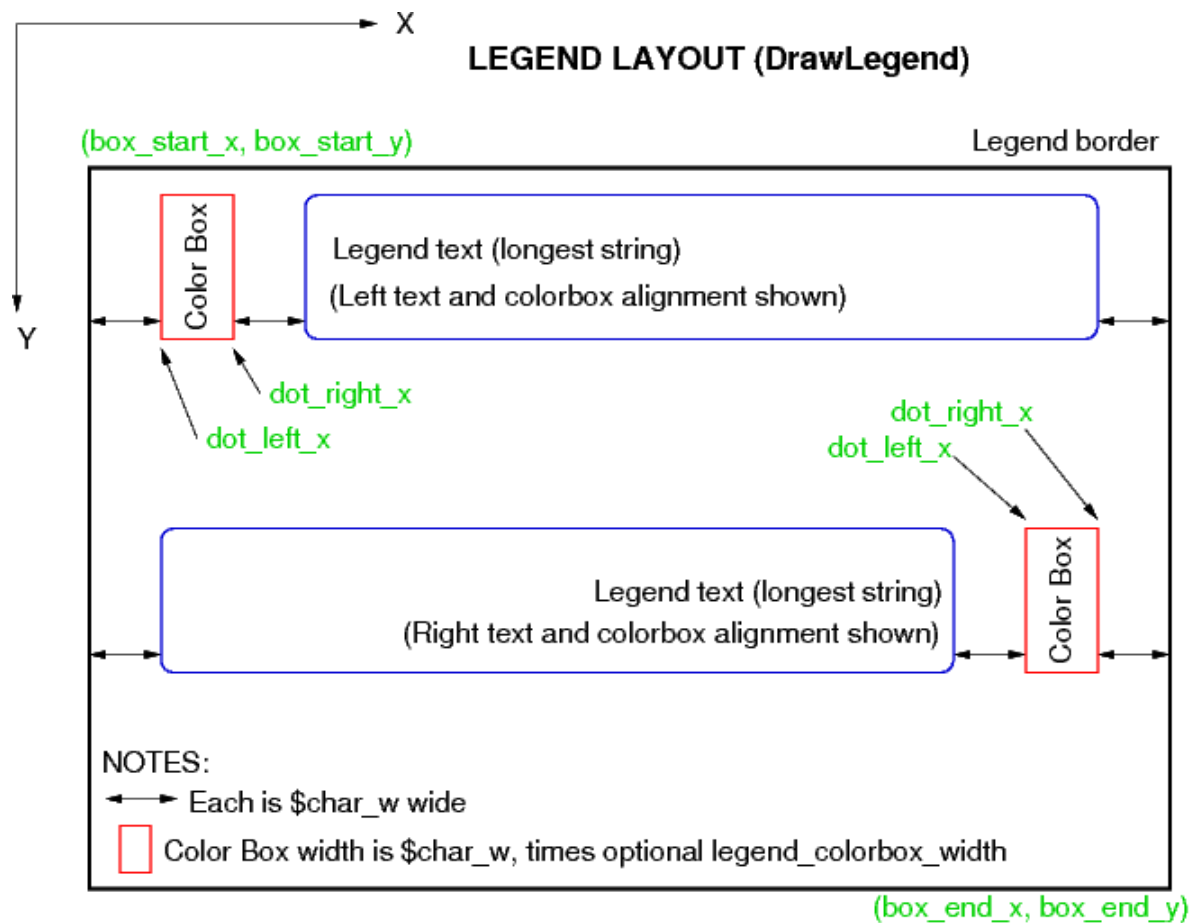
Chapter 9. PHPlot Class Internal Functions

This chapter documents PHPlot internal functions. These functions are intended to be used only by PHPlot itself.

Note

Starting with PHPlot-5.1.0, most of the internal functions are declared as `protected`, which limits their visibility to other member functions and inherited or parent classes. Some internal functions are still `public`, usually because they are needed for testing PHPlot. However, all functions documented in this chapter should be treated as private, for use only by PHPlot. If you feel you have a need to use one of these functions from outside PHPlot (or an inherited class), please report this via the available PHPlot support mechanisms.

array_merge_php4($array1,$array2)
　　This non-member function was removed at PHPlot-5.0.4.

array_pad_array(&$arr, $size, $arr2=NULL)
　　This non-member function was removed at PHPlot-5.0.4 and replaced with the class member function pad_array.

CalcAxisPositions()
　　Calculates the X and Y axis positions in world coordinates. These can be supplied by the user, in which case they are only changed if they are outside the data range. If axis positions are not supplied by the user, CalcAxisPositions applies defaults as described in SetXAxisPosition and SetYAxisPosition. Called by DrawGraph. Note: This code was moved out of CalcTranslation at PHPlot-5.0.5.

CalcBarWidths($stacked, $verticals)
　　Calculates values for `bars` and `stackedbars` plot types. It calculates the width of the bars and the margins around and between them. An argument was added in PHPlot-5.1.2 to support horizontal plots. Arguments were changed in PHPlot-5.3.0 to explicitly select stacked or grouped bars, and vertical or horizontal plots. This is called by the bar chart drawing functions DrawBars, DrawStackedBars, DrawHorizBars, and DrawHorizStackedBars. (Through PHPlot-5.1.2 this was called by DrawGraph before a bars or stackedbars plot.)

CalcGridSettings()
　　Applies defaults to the X and Y grid settings if not user specified. This was added in PHPlot-5.1.2 when the previous, static defaults (X grid off, Y grid on) needed to be dependent on the plot type. Called by DrawGraph.

CalcMargins($maximize)
　　Calculates the size of the four margins around the plot area: x_left_margin, x_right_margin, y_top_margin, and y_bot_margin. It does this by trying to determine how much space is needed for titles, labels, and tick marks. Starting with PHPlot-5.0.5, this is only called by DrawGraph, and it also calculates position offsets for titles and labels. It is called even in case of a user-supplied plot area (SetPlotAreaPixels or SetMarginsPixels was used). If the $maximize argument is true, then the plot area does not leave room for X or Y axis, labels, or titles; this is used for pie charts.

　　Through PHPlot-5.0.6, all 4 margins are either user-defined or all 4 are automatically calculated. Starting with PHPlot-5.0.7, the 4 margins can be independently set or defaulted to automatic. CalcMargins calculates values for all 4 margins, but only saves those that have not been set using either SetMarginsPixels or SetPlotAreaPixels. Note that other than the overall plot title, elements are drawn relative to the plot area, which is calculated based on the actual margins. If the top margin is increased, for example, the plot title stays at the top of the image, but top tick marks and labels move down against the graph.

CalcMaxDataLabelSize($which = 'x')
> Calculates the size of the biggest X or Y data label. For 'x' it returns the height along the Y axis of the tallest data label. For 'y' it returns the width along the sides of the widest data label. This is used to allocate space for margins. This was added to PHPlot-5.0.5. The argument supporting Y label width was added in PHPlot-5.1.2. Called by CalcMargins.

CalcMaxTickLabelSize($which)
> Calculates the size of the biggest tick label. The $which argument is 'x' or 'y' to indicate which labels to work with. For 'x', it returns the height along the Y axis; for 'y' it returns the width along the X axis. This is used to allocate space for margins. This was added to PHPlot-5.0.5. Calls CalcTicks to determine the tick value parameters. Called by CalcMargins.

CalcPlotAreaPixels()
> Calculates the pixel coordinates of the plot area. This was added to PHPlot-5.0.5 by moving the parts of the calculations out of SetPlotAreaPixels and SetMarginsPixels. Those two functions now simply record their arguments, and make no attempt to calculate any parameters. Called by DrawGraph, after CalcMargins is used to calculate margins.

CalcPlotAreaWorld()
> Calculates the world coordinate limits of the plot area. This was added to PHPlot-5.0.5 by moving the calculations out of SetPlotAreaWorld. Called by DrawGraph, after FindDataLimits is used to examine the data array values.

CalcTicks($which)
> Calculates the tick parameters. $which is 'x' or 'y'. Returns an array of 3 values: start, end, and interval. This was added to PHPlot-5.0.5, although it still uses the same method of just dividing the interval by 10 if there is no user-supplied interval or tick count. Called by CalcMaxTickLabelSize, DrawYTicks, and DrawXTicks.

CalcTranslation()
> Calculates the parameters for transforming world to pixel coordinates. This function calculates the scale (xscale, yscale) and origin (plot_origin_x, plot_origin_y) for X and Y translations, which are used by the xtr() and ytr() functions. Starting with PHPlot-5.0.5, this is only called by DrawGraph, as nothing else uses the parameters until the graph is ready to be drawn.

CheckDataArray()
> Checks that there is a valid data array for the plot, and calculates values that depend on the data type. Called very early by DrawGraph. This was added in PHPlot-5.1.3, moving the checking out of DrawGraph and adding the data_columns calculation.

CheckDataType($valid_types)
> Used to validate the data_type for a plot_type. This works like a specialized version of CheckOption. $valid_types contains the valid data type(s) for the current plot_type, separated by a comma and space if more than one is supported. If the current data_type is in the list, returns TRUE, else produces an error. This was added in PHPlot-5.1.2 to unify the way the plot drawing functions check the data type.

CheckDataValueLabels($label_control, &$x_adj, &$y_adj, &$h_align, &$v_align)
> Checks to see if data value labels should be drawn, based on $label_control (which is always `$this->y_data_label_pos`, since only vertical plot functions currently use this). Returns FALSE if the labels are off, else returns TRUE and sets 4 variables which are used by DrawDataValueLabel to position the label. $x_adj and $y_adj are pixel offsets for the label, and $h_align and $v_align set the text alignment. Called by plot type drawing functions such as DrawDots which do data value labels, other than bars and stackedbars. This was added in PHPlot-5.3.0.

CheckLabels()
> Fixes up the data and tick label position, angle, and format settings. This applies defaults to data label and tick label positions but avoids having them overlap unless the user deliberately positioned them that way. Also sets the default angle for X data labels, and the default formatting for X and Y data labels, to match the corresponding

settings for tick labels if they were not already set. This is for compatibility with PHPlot-5.0.7 and earlier, when there was only one control for both types of labels. Called by DrawGraph before calculating margins with CalcMargins. This was added in PHPlot-5.1.0.

CheckLabelsAllEmpty()

Returns TRUE if all of the labels in the data array are empty strings. This is used by CheckLabels to determine whether tick or axis data labels should default on if both are left off. This was added in PHPlot-5.1.2.

CheckOption($which_opt, $which_acc, $which_func)

Checks the validity of an option passed to a PHPlot member function. $which_opt is the string to check, and $which_acc is a string of acceptable choices (with a comma and then a space between choices). If the string to check is not found in the string of acceptable choices, a fatal error will be reported using DrawError. The error message will include $which_func which should be the name of the calling function (using the PHP __FUNCTION__ magic constant). Note that this is used to catch programming errors, not run-time or user errors. If the string to check is acceptable, it is returned down-cased and trimmed of leading and trailing spaces. Note: At PHPlot-5.0.5, this function was changed to require exactly ', ' (comma space) between acceptable choices, and reject empty strings for $which_opt and disallow partial matches.

CheckOptionArray($opt, $acc, $func)

Checks the validity of an option argument passed to a PHPlot member function that can accept its argument as either a string or an array of strings. $opt is the string or array to check, and $acc is a string of acceptable choices (with a comma and then a space between choices). If the $opt argument is supplied as a string, it is first converted to an array with one element. Then the elements of the array are each checked for validity. If any element of the array of strings to check is not found in the string of acceptable choices, a fatal error will be reported using DrawError. The error message will include $func which should be the name of the calling function (using the PHP __FUNCTION__ magic constant). Note that this is used to catch programming errors, not run-time or user errors. If all of the array elements to check are acceptable, the array is returned with each element down-cased and trimmed of leading and trailing spaces. Note that an array is always returned, even if the opt argument is a string. This was added in PHPlot-5.1.2.

CheckPointParams()

Adjusts the point_shapes and point_sizes arrays so they have the same size, and stores the size in a class variable. This handles processing deferred from SetPointShapes and SetPointSizes until graph drawing time. It must be called from internal functions that produce plots that use point shapes: currently DrawDots and DrawDotsError. This was added in PHPlot-5.1.0.

DecodeDataType()

Analyzes the data type and sets several member variables that other functions can use to understand how to process the data array. The variables it sets all have names starting with "datatype_" (refer to Chapter 10, *PHPlot Class Member Variables*). This is called by DrawGraph. It was added in PHPlot-5.1.2 (where it returned an array of 4 flags, and was called by several functions that needed to know the structure of the data array.) In PHPlot-5.1.3, it was changed to set member variables instead. It is now called just once, and the other functions reference the variables as needed.

DoCallback($reason, ...)

Call a callback (hook) function. $reason is the name given to the callback, for example `draw_titles` (meaning: call after drawing titles). The reasons are array indexes in the $callbacks class variable. Following that are zero or more arguments to pass to the callback, after the image resource and passthrough arguments. DoCallback does nothing if there is no callback registered for the given reason, otherwise it calls the callback function. See Section 4.3, "Callbacks" for more information on callbacks. Starting with PHPlot-5.1.3, DoCallback returns the value returned by the callback function, if any.

DrawArea($do_stacked = False)

Draws an area plot, or a stacked area plot (if the optional argument is True). Called by DrawGraph when the plot type is `area` or `stackedarea`. Stacked area plots were added in PHPlot-5.1.1; through PHPlot-5.1.0 this function did not have a parameter and was used only for area plots.

DrawBackground()
    Draws the image background, either an image file or solid fill or nothing. Called by DrawGraph.

DrawBar($x1, $y1, $x2, $y2, $data_color, $alt_color, $shade_top = TRUE, $shade_side = TRUE)
    Draws a single bar (or bar segment), with either shading or border. Four corner coordinates and 2 color indexes are provided. The first color $data_color is the bar fill color. The second color $alt_color is either the shading color or the border color, depending on whether shading is on or not (see GetBarColors). The last two arguments are flags used to suppress the top or side shading for certain cases of stacked bar segments. Called by DrawBars and DrawHorizBars to draw each bar. Called by DrawStackedBars and DrawHorizStackedBars to draw each bar segment. This was added in PHPlot-5.2.0, moving common code from those 4 functions.

DrawBars()
    Draws a bar chart plot. Called by DrawGraph when the plot type is bars. If the data type indicates a horizontal bar chart, calls DrawHorizBars instead.

DrawDataLabel($which_font, $which_angle, $x_world, $y_world, $which_color, $which_text, $which_halign = 'center', $which_valign = 'top', $x_adjustment=0, $y_adjustment=0)
    This function was removed in PHPlot-5.1.3 and replaced by DrawDataValueLabel.

DrawDataValueLabel($x_or_y, $x_world, $y_world, $text, $halign, $valign, $x_adjustment=0, $y_adjustment=0, $min_width=NULL, min_height=NULL)
    Draws a data value label (previously called "Y data label"). These are the above-bar or in-bar labels on bar and stackedbar charts.

    x_or_y is 'x' or 'y' to select the font, angle, and formatting type. The label is drawn at world coordinates (x_world, y_world) after device coordinate offset (x_adjustment, y_adjustment) is applied. Two arguments halign and valign specify the text alignment relative to the plotting point (see DrawText). The text argument is formatted with FormatLabel before drawing. The final arguments min_width and min_height are used to prevent labels from overlapping their allocated space. If either of these is supplied, the text is sized before drawing, and if it won't fit in the space the text is not drawn. This is used to suppress labels that are too wide to be drawn inside their bars, for example.

    This function was added in PHPlot-5.1.3 and replaced DrawDataLabel. It is called by the bar chart drawing functions including DrawBars and DrawStackedBars if these labels are enabled.

DrawDot($x_world, $y_world, $record, $color)
    Draws a single marker point ('dot') at the given X and Y world coordinates, using the given color. The $record parameter selects the marker shape and size using the arrays set up with SetPointSizes() and SetPointShapes(); they are not passed as arguments themselves. Called by those plotting routines that include dots: DrawDotsError and DrawDots.

DrawDots($paired = False)
    Draws a dot plot. Called by DrawGraph when the plot type is points, and by DrawLinePoints for the points portion of a linepoints plot. If the data type indicates an error bar plot, it calls DrawDotsError instead. (Through PHPlot-5.1.2, this was called directly from DrawGraph for both lines and linepoints plots, but not for error bar plots.) $paired is true for linepoints plots, to suppress drawing elements that would be duplicated by the 'lines' part of the plot. (This argument was added in PHPlot-5.1.3.)

DrawDotsError($paired = False)
    Draws a dot plot with error bars. Called by DrawDots (and indirectly from DrawLinePoints) when the data type indicates an error bar plot. (Through PHPlot-5.1.2 this was called directly by DrawGraph.) $paired is true for linepoints error plots, and defaults to false for points error plots. If true, it suppress drawing elements that would be duplicated by DrawLinesError. (This argument was added in PHPlot-5.1.3.)

DrawError($error_message, [$where_x], [$where_y])

Starting with PHPlot-5.0.5, this function is an alias for PrintError and is retained for compatibility. The $where_x and $where_arguments are now ignored. (Previously they positioned the error message on the image, but were never used.)

(Through PHPlot-5.0.4, DrawError wrote the error message to an image, wrote the image, and exited.)

DrawHorizBars()

Draws a horizontal bars plot. Called by DrawBars when the data type indicates a horizontal plot. This was added in PHPlot-5.1.2 (but was called by DrawGraph). In PHPlot-5.1.3 it was changed to be called by DrawBars.

DrawHorizStackedBars()

Draws a horizontal stacked bars plot. Called by DrawStackedBars when the data type indicates a horizontal plot. This was added in PHPlot-5.1.3.

DrawImageBorder()

Draws a border around the image, if enabled by SetImageBorderType. Called by DrawGraph.

DrawLegend(x,y,type)

Draws the plot legend. This includes the box, text labels, and color boxes. Called by DrawGraph, but only if legend text has been set using SetLegend.

DrawLinePoints()

Draws a linepoints plot, with or without error bars. Called by DrawGraph when the plot type is linepoints. It simply calls DrawLines and DrawDots, which handle both the plain and error bar cases. This was added in PHPlot-5.1.3 to provide a function specific to this plot type.

DrawLines($paired = False)

Draws a line plot. Called by DrawGraph when the plot type is lines, and by DrawLinePoints for the lines portion of a linepoints plot. If the data type indicates an error bar plot, it calls DrawLinesError instead. (Through PHPlot-5.1.2, this was called directly from DrawGraph for both lines and linepoints plots, but not for error bar plots.) $paired is true for linepoints plots, to suppress drawing elements that would be duplicated by the points part of the plot. (This argument was added in PHPlot-5.1.3.)

DrawLinesError($paired = False)

Draws a lines plot with error bars. Called by DrawLines (and indirectly from DrawLinePoints) when the data type indicates an error bar plot. (Through PHPlot-5.1.2 this was called directly by DrawGraph.) $paired is true for linepoints error plots, to suppress drawing elements that would be duplicated by DrawDotsError. (This argument was added in PHPlot-5.1.3.)

DrawOHLC($draw_candles, $always_fill = FALSE)

Draws one of the three types of OHLC (Open/High/Low/Close) plots. If $draw_candles is FALSE, draws a basic OHLC plot (plot type ohlc). If $draw_candles is TRUE and $always_fill is FALSE, draws a candlestick OHLC plot (plot type candlesticks). If $draw_candles is TRUE and $always_fill is TRUE, draws a filled candlestick OHLC plot (plot type candlesticks2). Called by DrawGraph when the plot type is one of those three types. This was added in PHPlot-5.3.0.

DrawPieChart()

Draws a pie chart plot. Called by DrawGraph with the plot type is pie.

DrawPlotAreaBackground()

Draws the plot area background, either an image file set with SetPlotAreaBgImage, or else a solid fill color selected by SetPlotBgColor if enabled with SetDrawPlotAreaBackground or else nothing. Called by DrawGraph.

DrawPlotBorder()

Draws the border around the plot area. This draws zero to four lines around the plot area (depending on SetPlotBorderType). It only draws the outline; other functions draw the tick marks and labels for the X axis and Y axis. Called by DrawGraph.

DrawSquared()
> Draws a squared (stepped lines) plot. Called by DrawGraph when the plot type is `squared`.

DrawStackedBars()
> Draws a stacked bars chart plot. Called by DrawGraph when the plot type is `stackedbars`. If the data type indicates a horizontal plot, calls DrawHorizStackedBars instead.

DrawText($which_font, $which_angle, $which_xpos, $which_ypos, $which_color, $which_text, $which_halign = 'left', $which_valign = 'bottom')
> Draws a string of text $which_text, at position ($which_xpos, $which_ypos). The font is selected with $which_font, which is one of the font arrays in the object. The text is drawn at angle $which_angle (built-in fonts can be used at 0 and 90 degrees only, TrueType at any angle). $which_color is a GD color index for the image. Text alignment relative to the (x,y) point is controlled with $which_halign (`center`, `left`, or `right`) and $which_valign (`center`, `bottom`, or `top`). Multi-line text strings are supported. This function accounts for the limitations and differences in GD text drawing routines for built-in and TrueType fonts. Called by numerous functions which place text on the plot.
>
> Starting with PHPlot-5.0.5, this function just calls ProcessText in text drawing mode. DrawText should be used by all internal PHPlot code that needs to draw text, and ProcessText should only be used by DrawText and SizeText.
>
> There are 7 font array variables in the PHPlot object. Through PHPlot-5.0.5, these were separate variables: generic_font, title_font, legend_font, x_label_font, y_label_font, x_title_font, and y_title_font. Starting with PHPlot-5.0.6, these were combined into a single class array variable `fonts`, indexed by the element name, for example $fonts['generic'].
>
> Starting with PHPlot-5.1.0, $which_font can be NULL or an empty string to use the *generic* font. This was intended to allow callbacks to avoid having to reference the internal class array variable which stores font information.

DrawThinBarLines()
> Draws a thin bar lines plot. This is sometimes called an impulse plot. Called by DrawGraph when the plot type is `thinbarline`. This function draws both vertical and horizontal variants of this plot type.

DrawTitle()
> Draws the main plot title as set with SetTitle. This is centered at the very top of the image. Called by DrawGraph.

DrawXAxis()
> Draws the X (horizontal) axis, including the axis line, tick marks and labels, and also draws the vertical grid lines. All of these except the axis line are done in DrawXTicks. Called by DrawGraph.

DrawXDataLabel($xlab, $xpos, $row=FALSE)
> Draws an axis data label for an X value. The labels are above or below the plot area or both, depending on the value set with SetXDataLabelPos. This also calls DrawXDataLine to draw a line from the label to the point, if enabled. This is used by vertical plots. Called by plot drawing routines for all plot types except `pie`: DrawArea, DrawBars, DrawDots, DrawDotsError, DrawThinBarLines, DrawLines, DrawLinesError, DrawSquared, and DrawStackedBars.

DrawXDataLine($xpos, $row)
> Draws X data lines, which are vertical lines from the bottom or top of the plot to the data points. This is enabled with SetDrawXDataLabelLines. The lines are drawn from the position (above, below, or both) of the X axis data labels, which are set with SetXDataLabelPos. Called by DrawXDataLabel.

DrawXTick($which_xlab, $which_xpix)
> Draws a single X value tick mark and its label. These can appear on the bottom of the graph, top of the graph, along the X axis (even if it is in the middle somewhere), on both sides, or nowhere, as set with SetXTickPos and SetXTickLabelPos. Called by DrawXTicks. This was added at PHPlot-5.0.5 by splitting the code out of DrawXTicks, for symmetry with DrawYTicks.

DrawXTicks()

Draws the vertical grid lines, the tick marks, and tick labels. Calls CalcTicks to calculate the tick parameters. Calls DrawXTick to draw each tick mark and its label. Called by DrawXAxis.

DrawXTitle()

Draws the X axis title. There can be zero, one, or two of them depending on the position parameter specified in SetXTitle. Calls DrawText to actually draw the title(s). Called by DrawGraph.

DrawYAxis()

Draws the Y (vertical) axis, including the axis line, tick marks and labels, and also draws the horizontal grid lines. All of these except the axis line are done in DrawYTicks. Called by DrawGraph.

DrawYDataLabel($ylab, $ypos)

Draws an axis data label for a Y value. The labels are along the Y axis or side of the plot, or both, depending on the value set with SetYDataLabelPos. This is used by horizontal plots, and was added in PHPlot-5.1.2. Called by horizontal plot drawing functions such as DrawHorizBars.

DrawYErrorBar($x_world, $y_world, $error_height, $error_bar_type, $color)

Draws an error bar for the point at world coordinates ($x_world, $y_world). $error_height is the height (in world coordinates) of the error bar: positive for the upper range error, negative for the lower range error. The $error_bar_type is set to tee or line by SetErrorBarShape. Called by DrawDotsError and DrawLinesError, twice for each point on the plot (upper range error and lower range error).

DrawYTick($which_ylab, $which_ypix)

Draws a single Y value tick mark and its label. These can appear on the left of the graph, right of the graph, along the Y axis (even if it is in the middle somewhere), on both sides, or nowhere, as set with SetYTickPos and SetYTickLabelPos. Called by DrawYTicks.

DrawYTicks()

Draws the horizontal grid lines, the tick marks, and tick labels. Calls DrawYTick to draw each tick mark and its label. Called by DrawYAxis.

DrawYTitle()

Draws the Y axis title. There can be zero, one, or two of them depending on the position parameter specified in SetYTitle. Calls DrawText to actually draw the title(s). Called by DrawGraph.

FindDataLimits()

Finds the limits of the data. Using the data_type and the data array, it goes through the points and determines the minimum and maximum X and Y values. It stores the min and max Y values for each row (plot line) in the class arrays data_min and data_max. (Before PHPlot-5.0.4, these were stored back into the data array with special index values MINY (-1) and MAXY (-2).) It also stores the overall min and max X and Y values as min_x, max_x, min_y, and max_y. It also stores the length of the longest data label in max_t. Starting with PHPlot-5.0.5, this is only called once by DrawGraph. (In PHPlot-5.0.4 and earlier, this was called from various places, with a flag data_limits_done to indicate it was called.)

FormatLabel($which_pos, $which_lab)

Formats a value for use as a tick or data label. This implements the format type selected with SetXLabelType ,SetYLabelType, SetXDataLabelType ,and SetYDataLabelType. By default it returns the value as-is, or it can format it as a floating point number or date/time value. Called by several functions that need to format label values. Separation of data and tick label formatting was available starting with PHPlot-5.1.0.

GetBarColors($row, $idx, &$vars, &$data_color, &$alt_color)

Gets the color indexes to be used for a bar plot. This is used by bar and stackedbar plot drawing functions, and accounts for a custom data color callback if defined. $row and $idx are the indexes for the current bar or bar segment. $vars is an array argument that maintains information across calls - the caller allocates an empty array, and this function updates it. The color index for bar filling is returned in $data_color. $alt_color is the color index to use for shading (if shading is on), else the color index to use for the bar border (if shading is off). (This dual

use argument corresponds to the usage of DrawBar.) This was added in PHPlot-5.2.0, moving common code into a function.

GetColorIndex($color)
Allocate a GD color index for a color given as a 4 component array (R,G,B,A). Returns a color index to be used in GD drawing functions. The color returned is the exact color requested if it already exists in the image, or if can be allocated. For palette images, if the color map is full, no new colors can be allocated, and this function will return an index to the closest existing color. For truecolor images, this function always returns an index for the exact requested color. This was added in PHPlot-5.2.0, replacing the second half of SetIndexColor.

GetColorIndexArray($color_array, $max_colors)
Allocates GD color indexes for each color in the array $color_array, which are given as 4 component arrays (R,G,B,A). Up to $max_colors colors will be allocated. (This is used to limit the number of allocated data colors to the number of data sets in the plot, for example.) GetColorIndex is used to allocate each color. Returns an array of GD color indexes to be used in GD drawing functions. This was added in PHPlot-5.3.1.

GetDarkColorIndex($color)
Allocate a GD color index for a darker shade of a color given as a 4 component array (R,G,B,A). Returns a color index to be used in GD drawing functions. The method used is to subtract 48 from each red, green, and blue component (without letting any go negative). The alpha component is not adjusted. The color returned is the exact color requested if it already exists in the image, or if can be allocated, else the closest color available. This is used for shadow colors (for example, in bar charts and pie charts). This was added in PHPlot-5.2.0, replacing the second half of SetIndexDarkColor.

GetDarkColorIndexArray($color_array, $max_colors)
Allocates GD color indexes for a darker shade of each color in the array $color_array, which are given as 4 component arrays (R,G,B,A). Up to $max_colors colors will be allocated. (This is used to limit the number of allocated dark-shade data colors to the number of data sets in the plot, for example.) GetDarkColorIndex is used to allocate each color. Returns an array of GD color indexes to be used in GD drawing functions. This was added in PHPlot-5.3.1.

GetDataColor($row, $idx, &$vars, &$data_color, $extra=0)
Gets the color index to be used for a data element (a point or line segment, for example). This is used by multiple plot drawing functions to get the data color, accounting for a custom data color callback if defined. $row and $idx are the indexes for the data point. $vars is an array argument that maintains information across calls - the caller allocates an empty array, and this function updates it. The color index to use is returned in $data_color. $extra contains extra information for a data color callback. This was added in PHPlot-5.2.0, moving common code into a function.

GetDataErrorColors($row, $idx, &$vars, &$data_color, &$error_color, $extra=0)
This is an extended version of GetDataColor which is used for error bar plots. It returns two color index values: $data_color for the data element, and $error_color for the error bar. This is used by error plot drawing functions, and accounts for a custom data color callback if defined. $row and $idx are the indexes for the data point. $vars is an array argument that maintains information across calls - the caller allocates an empty array, and this function updates it. $extra contains extra information for a data color callback. This was added in PHPlot-5.2.0, moving common code into a function.

GetDefaultTTFont()
Returns the default TrueType font name. If no default font has been set using SetDefaultTTFont, the first time this is called it will go through a list of likely sans-serif fonts, trying to find one that works. The first one that works, or the last one if none works, will be set as the default font. This was added in PHPlot-5.1.3, replacing the static initialization of default font.

GetImage($image_filename, &$width, &$height)
Reads an image file from $image_filename, stores the width and height (in pixels) in the $width and $height reference arguments, and returns a PHP GD image resource of the image. This is used by SetInputFile and tile_img.

Errors go to PrintError; there is no way for the script to recover. Possible errors include an image file type which is unsupported by PHP GD, or a corrupt image file. Note: This was added at PHPlot-5.0.4, by moving common code from the two calling functions.

GetImageBorderWidth()

This returns the image border width, as set with SetImageBorderWidth or as defaulted. It is used by CalcMargins to account for image border width, and by DrawImageBorder when drawing the image border. This was added in PHPlot-5.1.2.

GetLineSpacing($font)

Given a font array variable, returns the proper spacing in pixels between lines of text using that font. This works for both GD and TrueType fonts. See also SetLineSpacing. Used by ProcessTextGD, ProcessTextTTF, and DrawLegend. Note: This was added at PHPlot-5.0.6, with support for mixing TTF and GD fonts.

NeedDataDarkColors()

Allocates darker data colors, which are used for shading. This is called by graph drawing functions such as DrawPieChart if they need to use these colors for shading. This was added in PHPlot-5.2.0.

NeedErrorBarColors()

Allocates colors used for error bars. This is called by any graph drawing functions that is going to draw error bars. This was added in PHPlot-5.2.0.

number_format($number, $decimals=0)

Formats a floating point number, like PHP's number_format(), inserting a decimal separator and thousands groups separators. Unlike the PHP function, this uses variables in the PHPlot class to select the separators. The separators can be set with SetNumberFormat, or by default PHPlot will attempt to get locale-specific values. For example, 1234+(56/100) will be returned as "1,234.56" if the locale is "en_US", and as "1.234,56" if the locale is "de_DE". As a fall-back, if locale information is not available, '.' is used for decimal point, and ',' for thousands separator. This fall-back is equivalent to the behavior in PHPlot 5.0rc3 and earlier. This is used by FormatLabel when the formatting type is data, and also for the pie chart labels in DrawPieChart.

pad_array(&$arr, $size)

Pads an array $arr with copies of itself until it reaches the given size. If $arr is a scalar, it will first be converted to an array with one element. Then, if $arr has fewer than $size elements, elements of $arr starting from the first will be appended until it reaches $size elements. This only works on zero-based sequential integer indexed arrays. Called by PadArrays, SetPointShapes, and SetPointSizes. This replaced array_pad_array at PHPlot-5.0.4, however that had an unused 3rd argument, and worked on general indexed arrays.

PadArrays()

Pads the style arrays (line_widths, line_styles, data_colors, etc.) so they are all large enough to contain an entry for each data set or plot line. This uses pad_array. Called by DrawGraph before drawing anything.

PrintError($error_message)

Handles a fatal error within PHPlot. Starting with PHPlot-5.0.5 this and DrawError are identical. PrintError attempts to draw the error message $error_message into the image, and then output the image. This method is used because PHPlot is normally expected to output an image, and text output would not be displayed properly. (If no image resource was available, and the SetIsInline flag is not on, PHPlot will send a 500 Internal Server Error header.) After this, PrintError uses the PHP trigger_error() function to signal a user error. This is normally fatal to the script, unless caught. This will also result in the error message written to the error output stream, which typically ends up in a web server error log.

(Through PHPlot-5.0.4, PrintError wrote an error message to standard output and exited.)

ProcessText($draw_it, $font, $angle, $x, $y, $color, $text, $halign, $valign)

This function acts as a bridge, or switch, between the two functions SizeText and DrawText, which handle both GD and TTF text, and the functions which specifically handle GD text or TTF text. The arguments to this function are the same as DrawText except for an additional first argument $draw_it. If $draw_it is true, text is drawn. This

is used by DrawText. If $draw_it is false, only the bounding box size of the text is calculated and returned. This is used by SizeText. In text sizing mode, the x, y, color, halign, and valign arguments are ignored, as they are not needed when calculating the text bounding box size. This function is only called by SizeText and DrawText, and calls either ProcessTextTTF or ProcessTextGD.

ProcessTextGD($draw_it, $font, $angle, $x, $y, $color, $text, $h_factor, $v_factor)

Draws GD fixed-font text, or calculates the size of GD fixed-font text. This is only called by ProcessText after it determines that GD text is in use. If $draw_it is true, text is drawn; if $draw_it is false, only the bounding box size of the text is calculated and returned as a two-element array ($width, $height). Here $width is measured along the X axis, and $height along Y, regardless of the text angle. These are the size of an orthogonal bounding box that contains the text block. The $font argument is a PHPlot font array, which must reference a GD font. The $angle is 0 or 90 degrees, as GD text only supports those values. $x, $y are the reference point of the text $text, which is drawn in color $color. The text string can contain multiple lines, with a newline character between lines. The $h_factor and $v_factor arguments are translated from the alignment arguments supplied to DrawText or SizeText: 0, 0.5, or 1.0 If $draw_it is false, for text sizing mode, the x, y, color, h_factor and v_factor arguments are ignored.

Note: This was added at PHPlot-5.0.5. It was changed at PHPlot-5.0.6 to take a single font array argument, rather than 3 separate arguments for font number, width, and height.

ProcessTextTTF($draw_it, $font, $angle, $x, $y, $color, $text, $h_factor, $v_factor)

Draws TTF text, or calculates the size of TTF text. This is only called by ProcessText after it determines that TTF text is in use. If $draw_it is true, text is drawn; if $draw_it is false, only the bounding box size of the text is calculated and returned as a two-element array ($width, $height). Here $width is measured along the X axis, and $height along Y, regardless of the text angle. These are the size of an orthogonal bounding box that contains the text block. The $font argument is a PHPlot font array, which must reference a TTF font. The text is drawn at $angle degrees; unlike GD text TTF text can be drawn at any angle. $x, $y are the reference point of the text $text, which is drawn in color $color. The text string can contain multiple lines, with a newline character between lines. The $h_factor and $v_factor arguments are translated from the alignment arguments supplied to DrawText or SizeText: 0, 0.5, or 1.0 If $draw_it is false, for text sizing mode, the x, y, color, h_factor and v_factor arguments are ignored.

Note that the interpretation of the alignment for text at arbitrary angles may not be what you expect. Rotation of text happens before alignment, and alignment and positioning use the orthogonal bounding box of the text.

Note: This was added at PHPlot-5.0.5. It was changed in PHPlot-5.0.6 to take a single font array argument, rather than 2 separate arguments for font filename and size.

SetColorIndexes()

Allocates all the colors needed for a plot. Called by DrawGraph before drawing anything. This was added in PHPlot-5.2.0.

SetDashedStyle($which_ndxcol)

Sets the GD line style to select a dashed line, in preparation for drawing a dashed line. To understand how this works, it helps to look at the documentation for the PHP GD function ImageSetStyle() and also refer to SetDefaultDashedStyle. GD expects a line style to be specified as an array of pixel values, which is awkward to deal with. PHPlot uses a shorthand notation with integer values indicating pairs of the number of color, then transparent pixels. SetDefaultDashedStyle creates a string of PHP code to generate an array of pixel values the way GD wants them, but defers filling in the exact color to use. SetDashedStyle evaluates this string with $which_ndx_color set to the color to use for the current line. The result is an array of pixel values for GD's ImageSetStyle. Called by DrawXTicks and DrawYTicks if a dashed grid is selected, as well as other line drawing functions such as DrawLines if dashed lines are used.

SetDefaultFonts()

Selects all the default font values and sizes. See SetFont for details of the font element names and default values. Called by PHPlot, the class constructor, to initialize fonts in the plot object, and by SetUseTTF to restore the defaults when changing from or to TrueType font usage.

SetDefaultStyles()

Initializes default colors and styles for PHPlot objects. Mostly this calls the public member functions such as SetDataColors but without specifying an array of colors, which causes the member functions to select default values. Called by PHPlot, the class constructor, to initialize the plot object.

SetIndexColor($which_color, $alpha = 0)

This function was removed in PHPlot-5.2.0. It parsed a color specification, and allocated a GD color index. The first part is replaced by calling SetRGBColor directly, and the second part is implemented with GetColorIndex at graph drawing time.

SetIndexDarkColor($which_color, $alpha = 0)

This function was removed in PHPlot-5.2.0. It parsed a color specification, and allocated a GD color index for a slightly darker shade of the color. The first part is replaced by calling SetRGBColor directly, and the second part is implemented with GetDarkColorIndex at graph drawing time.

SetInputFile($which_input_file)

Sets an image file $which_input_file to be used as the background image for the graph. Also resets the graph size to the size of the image file. This is used by the constructor, PHPlot. Note: In earlier releases, this was considered an externally available function. After a PHPlot object was created with the constructor, SetInputFile could be used to resize it and set the background image. Although this still works, it is deprecated. SetInputFile should be considered an internal-use-only function. Users should set the background image file using the 4th argument of PHPlot when creating an instance of the object.

SetLabelType($mode, $args)

Sets the formatting used for tick and data labels. This implements SetXLabelType, SetYLabelType, SetXDataLabelType ,and SetYDataLabelType. $mode is either 'x', 'xd', 'y', or 'yd' and selects the axis to configure and type of label being drawn (x, y for tick labels; xd, yd for data labels). $args is an array of arguments, with $args[0] selecting the type of formatting (for example, `data`). Additional array elements depend on the formatting type. For more details, see the above-referenced functions. All arguments to those functions are combined into an array and passed to `SetLabelType` as $args. Separation of data and tick label formatting was available starting with PHPlot-5.1.0.

SetRGBColor($color_asked, $alpha = 0)

Converts a general color specification into a standard form as an array of 4 components: red, green, blue, and alpha, and returns the array. The 3 color components are integers in the range 0-255, and the alpha component is an integer in the range 0-127 (where 0 means opaque). The acceptable color specification forms are documented in Section 3.5.1, "Color Parameter Forms" and Section 4.2.4, "Color Parameter Form Extensions", and include color names, component arrays, and strings of the form #RRGGBB and #RRGGBBAA. The alpha argument provides a default value if the color specification does not include alpha; the default 0 makes the color opaque. This is used directly by all functions that accept a color specification. Use of alpha in the color specification, the default alpha argument, and the 4th component in the returned array were added in PHPlot-5.1.1.

SizeText($which_font, $which_angle, $which_text)

Calculates the size of a block of text. It works on both GD (fixed-font) and TTF text. $which_font is a PHPlot font array, $which_angle is the text angle in degrees, and $which_text is the text string. The text string can contain multiple lines, with a newline character between lines. This function just calls ProcessText in text sizing mode to do the work. It returns a two-element array with the text width and height. These are the width and height of an orthogonal bounding box (box aligned with the X and Y axes) which contains the rotated text block. Called by functions which need to determine text size for laying out plot elements, such as CalcMargins. This function replaced TTFBBoxSize at PHPlot-5.0.5.

tile_img($file, $xorig, $yorig, $width, $height, $mode)

Tiles an image file over another the current plot image. $file is the filename of the image to use as the tile. ($xorig, $yorig) are the origin point for the tiling, and ($width, $height) are the area to be tiled. These are used to tile just under the plot area versus the entire image. The $mode can be `centeredtile`, `tile`, or `scale`. Scale mode

scales the source image to fit the target area. Tile and centeredtile modes repeat the source image as needed to fit into the target area; the difference is that centeredtile offsets the tile start position by half its size, which works better for some tiles. Called by DrawBackground and DrawPlotAreaBackground if an image file is selected for the plot area or overall background.

truncate_array(&$array, $size)

This was added in PHPlot-5.2.0 and removed in PHPlot-5.3.1, when data color processing was changed to not truncate the color arrays.

TTFBBoxSize($size, $angle, $font, $string)

This function was removed at PHPlot-5.0.5. It was replaced by SizeText.

xtr($x_world)

Translates an X world coordinate value into a pixel coordinate value. This uses the scale and translation set up by CalcTranslation. See GetDeviceXY for a public interface.

ytr($y_world)

Translates a Y world coordinate value into a pixel coordinate value. This uses the scale and translation set up by CalcTranslation. See GetDeviceXY for a public interface.

Chapter 10. PHPlot Class Member Variables

This chapter provides information about the PHPlot class member variables.

All PHPlot class member variables are meant for internal use only, although they are declared as *public*. You may use these in applications, but they are subject to change in future releases without concern for backward compatibility.

Section 4.5, "Tuning Parameters" describes how some of these PHPlot variables can be used to adjust plot appearance. Although these are also internal variables, setting them from your script is generally safe, as they have relatively small effects on plots, and are less likely to change in future releases.

10.1. List of Member Variables

The table below lists the PHPlot class member variables. If the variable is declared at the top of the PHPlot class definition, the initial value is listed under *Initialized to Value*. If nothing is listed under *Initialized to Value*, this means the variable is defined only when needed, and is not declared at the top of the PHPlot class definition. The *Reference Function* column lists the member function(s) used to set the variable, if the variable can be set by the application, else the member function which calculates the variable, if there is one, else the member function(s) which use the variable, if there are only a few.

Variable Name:	Initialized to Value:	Reference Function:	Description:
actual_bar_width		CalcBarWidths	Calculated width of bars for bar charts
bar_adjust_gap		CalcBarWidths	Calculated bar gap
bar_extra_space	0.5	CalcBarWidths	Extra space between groups of bars (See Tuning Parameters for more)
bar_width_adjust	1	CalcBarWidths	Width of bar relative to space for one bar (See Tuning Parameters for more)
bg_color	'white'	SetBackgroundColor	Color (R,G,B,A) for image background
bgimg		SetBgImage	Background image filename
bgmode		SetBgImage	Background image tiling mode
browser_cache	FALSE	SetBrowserCache	Flag: Don't send cache suppression headers
callbacks	array(...)	SetCallback	Callback (hook) function information. Indexed by callback reason; value is an array of function name and pass-through argument if the callback is in use, else NULL.
color_array	'small'	PHPlot	Name of the initial color map
dashed_grid	TRUE	SetDrawDashedGrid	Flag: Draw dashed or solid grid lines?
dashed_style	'2-4'	SetDefaultStyles	Initial dashed pattern code
data		SetDataValues	The data array

Variable Name:	Initialized to Value:	Reference Function:	Description:
data_border_colors	'black'	SetDataBorderColors	Array of colors (R,G,B,A) for bar chart data borders.
data_colors		SetDataColors	Array of colors (R,G,B,A) for data lines/marks/bars/etc. See default_colors for initial value.
data_columns		CheckDataArray	Maximum number of dependent variable values (usually Y values, or pie slices) in the data array rows. (Added in PHPlot-5.1.3)
data_max		FindDataLimits	Array: Per row maximum Y value. (Before PHPlot-5.1.2 this was named data_maxy.)
data_min		FindDataLimits	Array: Per row minimum Y value. (Before PHPlot-5.1.2 this was named data_miny.)
data_type	'text-data'	SetDataType	Format of the data array
datatype_error_bars		DecodeDataType	Flag: data type has error bars. (Added in PHPlot-5.1.3)
datatype_implied		DecodeDataType	Flag: data type has implied X or Y. (Added in PHPlot-5.1.3)
datatype_pie_single		DecodeDataType	Flag: data type is one-column data for pie chart with one slice per row. (Added in PHPlot-5.1.3)
datatype_swapped_xy		DecodeDataType	Flag: data type has swapped X and Y values. (Added in PHPlot-5.1.3)
data_units_text	''	FormatLabel	Obsolete - suffix for 'data'-formatted labels
data_value_label_angle		CheckDataValueLabels	Angle (in degrees) for data value labels (See Tuning Parameters for more)
data_value_label_distance		CheckDataValueLabels	Distance (in pixels) for data value labels (See Tuning Parameters for more)
decimal_point		SetNumberFormat	Character to use for decimal point in formatted numbers
default_colors	array(...)	SetDataColors, SetErrorBarColors	The default color array, used to initialize data_colors and error_bar_colors. (Added in PHPlot-5.1.0)
default_dashed_style		SetDefaultDashedStyle	Formatted PHP code to build a dashed line pattern
default_ttfont		SetDefaultTTFont, GetDefaultTTFont	Default TrueType font file. (Through PHPlot-5.1.2, there was a static default 'benjamingothic.ttf'. After PHPlot-5.1.2, the default is dynamic.)

Variable Name:	Initialized to Value:	Reference Function:	Description:
done		DrawBackground, DrawImageBorder, DrawTitle	Array of flags for elements that must be drawn at most once. Flag is set TRUE when drawn. Indexes are: background, border, title. (Replaced separate variables in PHPlot-5.3.1.)
draw_broken_lines	FALSE	SetDrawBrokenLines	Flag: How to handle missing Y values
draw_plot_area_background	FALSE	SetDrawPlotAreaBackground	Flag: Draw the background of the plot area
draw_x_data_label_lines	FALSE	SetDrawXDataLabelLines	Flag: Draw X data label lines
draw_x_grid		SetDrawXGrid, CalcGridSettings	Flag: Draw X grid lines?
draw_y_grid		SetDrawYGrid, CalcGridSettings	Flag: Draw Y grid lines?
error_bar_colors		SetErrorBarColors	Array of colors (R,G,B,A) for error bars. See default_colors for initial value.
error_bar_line_width	1	SetErrorBarLineWidth	Thickness of error bar lines
error_bar_shape	'tee'	SetErrorBarShape	Shape (style) of error bars: line or tee
error_bar_size	5	SetErrorBarSize	Size of error bars
file_format	'png'	SetFileFormat	Image format: png, gif, jpg, wbmp
fonts		SetFontGD, SetFontTTF	Array of font information. (See notes for more)
grid_at_foreground	FALSE	DrawGraph	Flag: Draw grid on top of or behind the plot (See Tuning Parameters for more)
grid_color	'black'	SetGridColor	Color (R,G,B,A) to use for axes, plot area border, legend border, pie chart lines and text (not grid!)
group_frac_width	0.7	CalcBarWidths	Controls fraction of bar group space used for bar (See Tuning Parameters for more)
i_border	array(194, 194 ,194)	SetImageBorderColor	Color (R,G,B,A) for image border, if drawn
image_border_type	'none'	SetImageBorderType	Image border type
image_border_width		SetImageBorderWidth	Width of image border in pixels. Default depends on image_border_type. (Added in PHPlot-5.1.2)
image_height		PHPlot	Image height
image_width		PHPlot	Image width
img		PHPlot	Image resource
in_error		PrintError	Prevent recursion in error message image production

Variable Name:	Initialized to Value:	Reference Function:	Description:
is_inline	FALSE	SetIsInline	Don't sent headers
label_format	array(...)	SetXLabelType, SetYLabelType, SetXDataLabelType, SetYDataLabelType, SetXTimeFormat, SetYTimeFormat, SetPrecisionX, SetPrecisionY	Label format info. (See notes for more)
label_scale_position	0.5	SetLabelScalePosition	Pie chart label position factor
legend	''	SetLegend	Legend text array. Each index is a legend text line.
legend_colorbox_align		SetLegendStyle	Legend alignment of color boxes, left, right, or none
legend_colorbox_width		DrawLegend	Adjusts width of color boxes in the legend (See Tuning Parameters for more) (Added in PHPlot-5.3.0)
legend_text_align		SetLegendStyle	Legend style setting, left or right
legend_x_pos		SetLegendPixels, SetLegendWorld	Forced legend position
legend_xy_world		SetLegendPixels, SetLegendWorld	Flag indicating legend position was set in World coords
legend_y_pos		SetLegendPixels, SetLegendWorld	Forced legend position
light_grid_color	'gray'	SetLightGridColor	Color (R,G,B,A) for grid lines and X data lines
line_spacing	4	SetLineSpacing	Controls inter-line spacing of text
line_styles	array(...)	SetLineStyles	Plot line style(s)
line_widths	1	SetLineWidths	Plot line width(s)
locale_override		number_format	Flag to avoid importing locale info (See Tuning Parameters for more)
max_x		FindDataLimits	Overall max X value in the data array
max_y		FindDataLimits	Overall max Y value in the data array
min_x		FindDataLimits	Overall min X value in the data array
min_y		FindDataLimits	Overall min Y value in the data array
ndx_bg_color		SetColorIndexes	Color index of image background
ndx_data_border_colors		SetColorIndexes	Color index array for bar chart data borders
ndx_data_colors		SetColorIndexes	Color index array for plot data lines/ marks/bars/etc.
ndx_data_dark_colors		NeedDataDarkColors	Color index array for plot data, darker shade

Variable Name:	Initialized to Value:	Reference Function:	Description:
ndx_error_bar_colors		NeedErrorBarColors	Color index array for error bars
ndx_grid_color		SetColorIndexes	Color index for axes, plot area border, legend border, pie chart lines and text
ndx_i_border		SetColorIndexes	Color index for image border lines
ndx_i_border_dark		SetColorIndexes	Color index for image border lines, darker shade
ndx_light_grid_color		SetColorIndexes	Color index for grid lines and X data lines
ndx_plot_bg_color		SetColorIndexes	Color index of plot area background
ndx_text_color		SetColorIndexes	Color index for labels and legend text
ndx_tick_color		SetColorIndexes	Color index for tick marks
ndx_title_color		SetColorIndexes	Color index for main title
ndx_x_title_color		SetColorIndexes	Color index for X title (Added in PHPlot-5.2.0)
ndx_y_title_color		SetColorIndexes	Color index for Y title (Added in PHPlot-5.2.0)
num_data_rows		SetDataValues	Number of rows in the data array (number of points along X, or number of bar groups, for example)
num_recs		SetDataValues	Array with number of entries in each data row (including label and X if present)
num_x_ticks	"	SetNumXTicks	Forced number of X tick marks
num_y_ticks	"	SetNumYTicks	Forced number of Y tick marks
ohlc_min_width		DrawOHLC	Minimum half-width for elements in OHLC plots (candlestick bodies or tick pairs). Default is 2 pixels. (See Tuning Parameters for more) (Added in PHPlot-5.3.0)
ohlc_max_width		DrawOHLC	Maximum half-width for elements in OHLC plots (candlestick bodies or tick pairs). Default is 8 pixels. (See Tuning Parameters for more) (Added in PHPlot-5.3.0)
ohlc_frac_width		DrawOHLC	Scale factor for element widths in OHLC plots. Default is 0.3, meaning within min and max limits the elements will use 30% of the available space for half their width. (See Tuning Parameters for more) (Added in PHPlot-5.3.0)
output_file	"	SetOutputFile	Redirect to output file

Variable Name:	Initialized to Value:	Reference Function:	Description:
plot_area		CalcPlotAreaPixels	Array defining the calculated plot area. ([0],[1]) is the top left corner, ([2],[3]) is the bottom right corner.
plot_area_height		CalcPlotAreaPixels	Height of the plot area
plot_area_width		CalcPlotAreaPixels	Width of the plot area
plot_bg_color	'white'	SetPlotBgColor	Color (R,G,B,A) for plot area background
plot_border_type	'sides'	SetPlotBorderType	Where to draw plot borders. Can be scalar or array of choices.
plot_max_x		SetPlotAreaWorld, CalcPlotAreaWorld	Max X of the plot area in world coordinates
plot_max_y		SetPlotAreaWorld, CalcPlotAreaWorld	Max Y of the plot area in world coordinates
plot_min_x		SetPlotAreaWorld, CalcPlotAreaWorld	Min X of the plot area in world coordinates
plot_min_y		SetPlotAreaWorld, CalcPlotAreaWorld	Min Y of the plot area in world coordinates
plot_origin_x		CalcTranslation	X device coordinate of the plot area origin
plot_origin_y		CalcTranslation	Y device coordinate of the plot area origin
plot_type	'linepoints'	SetPlotType	Selected plot type
plotbgimg		SetPlotAreaBgImage	Plot area background image filename
plotbgmode		SetPlotAreaBgImage	Plot area background image tiling mode
plots	array(...)	DrawGraph, SetPlotType, FindDataLimits	Static array of plot type information (Added in PHPlot-5.3.0) (See notes for more)
point_counts		CheckPointParams	Size of point_shapes and point_sizes arrays (added in PHPlot-5.1.0)
point_shapes	array(...)	SetPointShapes	Marker shapes for point plots
point_sizes	array(...)	SetPointSizes	Marker sizes for point plots
print_image	TRUE	SetPrintImage	Flag: Automatic PrintImage after DrawGraph?
record_bar_width		CalcBarWidths	Area for each bar in a bar chart
records_per_group		SetDataValues	Maximum of num_recs[], max number of entries (including label and X if present) for all data rows
rgb_array		SetRGBArray	Array mapping color names to array of R, G, B values

Variable Name:	Initialized to Value:	Reference Function:	Description:
safe_margin	5		Fixed extra margin used in multiple places [(See Tuning Parameters for more)](#)
shading	5	SetShading	Drop shadow size for pie and bar charts
skip_bottom_tick	FALSE	SetSkipBottomTick	Skip bottom tick mark
skip_left_tick	FALSE	SetSkipLeftTick	Skip left tick mark
skip_right_tick	FALSE	SetSkipRightTick	Skip right tick mark
skip_top_tick	FALSE	SetSkipTopTick	Skip top tick mark
suppress_x_axis		SetDrawXAxis	Flag: Don't draw the X axis line. Unset (by default) means FALSE. (Added in PHPlot-5.3.0)
suppress_y_axis		SetDrawYAxis	Flag: Don't draw the Y axis line. Unset (by default) means FALSE. (Added in PHPlot-5.3.0)
text_color	'black'	SetTextColor	Color (R,G,B,A) for labels and legend text
thousands_sep		SetNumberFormat	Character to use to group 1000s in formatted numbers
tick_color	'black'	SetTickColor	Color (R,G,B,A) for tick marks
title_color	'black'	SetTitleColor	Color (R,G,B,A) for main title (and default for X and Y titles)
title_offset		CalcMargins	Y offset of main title position (Added in PHPlot-5.1.2)
title_txt	''	SetTitle	Main title text
total_records		SetDataValues	Total number of entries (rows times columns in each row) in the data array.
transparent_color		SetTransparentColor	Color (R,G,B,A) designated as transparent (Added in PHPlot-5.2.0)
ttf_path	'.'	SetTTFPath	TrueType font directory
use_ttf	FALSE	SetUseTTF	Default font type, True for TrueType, False for GD
x_axis_position	''	SetXAxisPosition	Position of X axis (in world coordinates)
x_axis_y_pixels		CalcTranslation	Device coordinate for the X axis
x_data_label_angle		SetXDataLabelAngle	X data label text angle (Added in PHPlot-5.1.0)
x_data_label_pos		SetXDataLabelPos	Position of X data labels. (Default was 'plotdown', but is now applied at graph drawing time.)
x_label_angle	0	SetXLabelAngle	X tick label text angle (and default for x_data_label_angle)

Variable Name:	Initialized to Value:	Reference Function:	Description:
x_label_axis_offset		CalcMargins	Label offset relative to plot area
x_label_bot_offset		CalcMargins	Label offset relative to plot area
x_label_top_offset		CalcMargins	Label offset relative to plot area
x_left_margin		CalcMargins	Calculated plot area margin - left side
x_right_margin		CalcMargins	Calculated plot area margin - right side
x_tick_cross	3	SetXTickCrossing	Length of X tick marks (inside plot area)
x_tick_inc	''	SetXTickIncrement	Step between X tick marks
x_tick_label_pos		SetXTickLabelPos	Position of X tick labels. (Default was 'plotdown', but is now applied at graph drawing time.)
x_tick_length	5	SetXTickLength	Length of X tick marks (outside plot area)
x_tick_pos	'plotdown'	SetXTickPos	Position of X tick marks
x_title_bot_offset		CalcMargins	Title offset relative to plot area
x_title_color		SetXTitleColor	Color (R,G,B,A) for X title (Added in PHPlot-5.2.0)
x_title_pos	'plotdown'	SetXTitle	X Axis title position
x_title_top_offset		CalcMargins	Title offset relative to plot area
x_title_txt	''	SetXTitle	X Axis title text
xscale		CalcTranslation	X scale factor for converting World to Device coordinates
xscale_type	'linear'	SetXScaleType	Linear or log scale on X
y_axis_position	''	SetYAxisPosition	Position of Y axis (in world coordinates)
y_axis_x_pixels		CalcTranslation	Device coordinate for the Y axis
y_bot_margin		CalcMargins	Calculated plot area margin - bottom
y_data_label_angle	0	SetYDataLabelAngle	Y data label text angle (Added in PHPlot-5.1.0)
y_data_label_pos		SetYDataLabelPos, CheckLabels	Position of Y labels above bars
y_label_angle	0	SetYLabelAngle	Y tick label text angle
y_label_axis_offset		CalcMargins	Label offset relative to plot area
y_label_left_offset		CalcMargins	Label offset relative to plot area
y_label_right_offset		CalcMargins	Label offset relative to plot area
y_tick_cross	3	SetYTickCrossing	Length of Y tick marks (inside plot area)
y_tick_inc	''	SetYTickIncrement	Step between Y tick marks

Variable Name:	Initialized to Value:	Reference Function:	Description:
y_tick_label_pos		SetYTickLabelPos, CheckLabels	Position of Y tick labels
y_tick_length	5	SetYTickLength	Length of Y tick marks (outside plot area)
y_tick_pos	'plotleft'	SetYTickPos	Position of Y tick marks
y_title_color		SetYTitleColor	Color (R,G,B,A) for Y title (Added in PHPlot-5.2.0)
y_title_left_offset		CalcMargins	Title offset relative to plot area
y_title_pos	'plotleft'	SetYTitle	Y Axis title position
y_title_right_offset		CalcMargins	Title offset relative to plot area
y_title_txt	''	SetYTitle	Y Axis title text
y_top_margin		CalcMargins	Calculated plot area margin - top
yscale		CalcTranslation	Y scale factor for converting World to Device coordinates
yscale_type	'linear'	SetYScaleType	Linear or log scale on Y

10.2. Member Variable Notes

This section contains details on some of the PHPlot class member variables listed in the previous section. Remember that all PHPlot class member variables are meant for internal use only.

10.2.1. fonts[]

The `fonts[]` array contains information about the fonts to use for text on the plot. The array keys are the element names (such as `title` or `legend`) as used in SetFont, SetFontGD, or SetFontTTF. The array values are arrays which contain information about the font to use for that element. The keys and values of the second-level arrays are:

Key	Value for TTF	Value for GD Font
ttf	True for a TrueType font	False for a GD font
font	Pathname of the font file	Font number: 1 through 5
size	Font point size	Not used
height	Height in pixels of an upper-case "E" in the font	Font height in pixels
width	Width in pixels of an upper-case "E" in the font.	Font width in pixels
spacing	Font's built-in inter-line spacing	Not used
line_spacing	User-requested inter-line spacing factor.	Same as for TTF

For TrueType fonts, the height and width can vary by character. The fonts array stores a height and width value for the font, but these are only used for sizing non-text plot elements (such as the legend color boxes). When PHPlot needs to know the drawn size of a string that will use TTF, it calculates the exact size of that specific string when drawn with the designated font.

GD fonts have fixed character width and height, so the values stored in the fonts array can be used to calculate text string sizes.

The `spacing` key stores the TrueType font's built-in inter-line spacing. Although TrueType fonts have this information internally, PHP cannot access it, so PHPlot calculates it by taking the height of the string "E\nE" and subtracting twice the height of the letter E.

The `line_spacing` key stores the user-specified line spacing adjustment for a text element, if any, from SetFont or one of the two related functions. It will be NULL if the spacing was not set for this element, meaning use the default line spacing. See SetLineSpacing for more information on how this is used.

Here is an example of part of a fonts array, for the title element:

```
$plot->fonts['title'] = array(
    'ttf' => FALSE,          // This element uses a GD font
    'font' => 2,             // Use GD font 2
    'height' => 13,          // Provided by GD
    'width' => 6,            // Provided by GD
    'line_spacing' => NULL,  // Use default line spacing
)
```

10.2.2. label_format[]

The `label_format[]` array contains information about how text labels should be formatted. This array has 4 entries, with keys 'x', 'y', 'xd', and 'yd'. The 'x' and 'y' entries are for tick labels, and the 'xd' and 'yd' entries are for data labels. (Note that PHPlot defaults data label formatting to match tick label formatting, but this is handled in CheckLabels, with the tick label format information copied to the data label format information if necessary.)

The value of each entry in `label_format` is an array containing formatting information. The arrays are empty by default, meaning there is no special formatting. If formatting has been enabled, for example with SetXLabelType, the arrays can contain the following keys and values:

Key	Used with type	Value
type		Formatting type: data, time, printf, or custom.
precision	data	The number of decimal positions to produce.
prefix	data	A prefix string to append to the label (for example, a currency sign).
suffix	data	A suffix string to append to the label (for example, a currency sign or percent sign). This replaces data_units_text (which still works too).
time_format	time	The date/time format string for the PHP strftime() function.
printf_format	printf	The format string for the PHP printf() function.
custom_callback	custom	The function (or array with object and method name) to call to format the label.
custom_arg	custom	An additional argument to pass to the custom_callback function.

10.2.3. plots[]

The `plots[]` array contains information about known plot types. Essentially, it tells FindDataLimits and DrawGraph how to produce the main portion of the plot. This was added in PHPlot-5.3.0. This is a static member variable in the PHPlot class, so it must be accessed as `PHPlot::$plots` and not through an instance of the class. It has protected visibility, so is not accessible from outside the class or derived classes.

The array keys are the plot types (for example: 'bars', 'linepoints'). These must not include upper case letters. The values are arrays which contain information about the plot type. The keys and values of the second-level arrays are:

Key	Value
draw_method	Name of the PHPlot class method (function) to call to draw the 'insides' of the plot. This is the only required entry. The function is responsible for drawing everything inside the plot area. (That is, everything except the plot titles, axis lines, tick marks and tick labels, grid lines, and legend.)
draw_arg	Optional argument(s) for the drawing method named by draw_method. If present, this must be an array. Each element of the array is passed to the draw_method function as a separate argument. If this is not specified, the drawing method is called with no arguments. This is generally used to 'overload' drawing methods so they handle multiple similar plot types.
draw_axes	A flag to indicate if the X axis and Y axis should be drawn. The default is TRUE. This is set to FALSE for pie charts.
abs_vals	Special data processing flag. If TRUE, this tells FindDataLimits that the plot type will take the absolute value of all dependent variable values in the data array. FindDataLimits needs this information so it knows how to find the minimum and maximum values in the data array. The default if omitted is FALSE.
sum_vals	Special data processing flag. If TRUE, this tells FindDataLimits that the plot type will sum up the dependent variable values in each row of the data array. FindDataLimits needs this information so it knows how to find the minimum and maximum values in the data array. The default if omitted is FALSE. If both abs_vals and sum_vals are TRUE, this means the plot type sums up the absolute values from the data array. (The stackedarea plot type does this.)

Appendix A. Change Log

This is the summary change log for the PHPlot Reference Manual. (For a more detailed list of changes, see the ChangeLog file in the manual's CVS repository. For changes to PHPlot itself, see the ChangeLog file inside the PHPlot distribution.)

2011-01-15
Released updated version of the manual to correspond to PHPlot-5.3.1.

2011-01-11
Added a new section in the Advanced chapter on placing multiple plots on an image, and added a new example of overlay plots.

2011-01-05
Added a short new section in the Advanced chapter showing how to extend the PHPlot class to customize default settings.

2010-12-26
New sub-section in Concepts chapter, Colors section, showing which function is used to set the color for each plot element.

2010-12-04
Released updated version of the manual to correspond to PHPlot-5.3.0.

2010-12-03
Document 2 new functions to control drawing of the axis lines.

2010-11-18
Restructuring. Too much advanced material was ending up in the basic programming chapter "Concepts". There is now a new chapter "Advanced", which includes the material on Callbacks (demoted from chapter to section), plus two sections previously in "Concepts": custom data color callback, and truecolor images. Also moved the Functions by Category chapter after Examples, not before.

Added new section "Tuning Parameters" in the Advanced chapter, to provide more information about class variables that can be set to adjust plot appearance.

2010-11-16
Documented 3 new plot types - OHLC financial plots, with examples.

2010-10-03
Released updated version of the manual to correspond to PHPlot-5.2.0.

2010-09-30
Changes for new color processing. Mostly internal functions and methods, but also two new public methods were added for controlling title colors.

2010-09-27
Document new behavior of stackedbar plots with negative values.

2010-09-25
Changes for new X and Y axis position defaults.

2010-09-13
Documentation was added and changes made for horizontal plot types and their new data types. (This was previously an experimental feature, documented in a text file included in with the PHPlot release.) Three new examples were added, showing horizontal plot types.

A terminology change was necessary due to the introduction of horizontal plots and dual use of Y data labels, which are now symmetrical with X data labels. Previously, *data labels* referred to labels from the data array which were displayed along the X axis or top edge, and *Y data labels* referred to the Y bar value labels displayed within the plot area. The manual now distinguishes these by type, not axis. *Axis data labels* are labels from the data array which are displayed along the independent variable axis, and *data value labels* show the value of a data point using the dependent variable. The general term *data labels* is used to refer to both types of labels. (For example, SetXDataLabelAngle applies to data labels - both axis data labels and data value labels.)

2010-09-08

Changes were made throughout the manual to support production of a PDF version. The changes should have only a small impact on the HTML version.

2010-08-30

Released updated version of the manual to correspond to PHPlot-5.1.3.

2010-08-27

Rewrote and updated sections related to TrueType font handling. Starting after PHPlot-5.1.2 this works differently and hopefully better.

2010-08-19

Removed the Extra Data Processing section in the Concepts chapter, and removed the `phplot_data.php` file from the installation steps. This file has been removed from the release. (Reference bug report 3048267.)

2010-07-29

Document the new data color customization callback with a new section in the Concepts chapter and two new examples.

2010-06-28

Released updated version of the manual to correspond to PHPlot-5.1.2.

2010-06-13

New section added to PHPlot Concepts chapter on using truecolor images. The reference and developer information were also updated, and a new example was added.

2010-05-30

Documented changes to image border and plot border functions.

2010-04-18

Added a new example to the examples chapter with a complete mini-application containing a web form and form handling script.

2010-04-03

Released updated version of the manual to correspond to PHPlot-5.1.1.

2010-03-25

Document new Y Data Labels for stacked bar charts.

Document new plot type 'stackedarea'.

2009-12-24

Released updated version of the manual to correspond to PHPlot-5.1.0.

2009-12-14

Document 4 new functions and changes to existing functions related to new controls on data labels versus tick labels.

Removed function list from Getting Started chapter, because it was out of date and duplicated material in another chapter.

2009-12-10

Added sections to the Concepts chapter about using TrueType fonts and special characters, because it didn't seem to have been documented anywhere else.

2009-11-18

Added a new section to the Callbacks chapter about using a callback for annotating a plot. This goes with a new example added to the Examples chapter.

2009-11-06

Added a new chapter to Part III Developer's Guide documenting the internal member class variables. The chapter on Callbacks has been moved from Part III up into Part I.

2009-10-12

Documented SetImageBorderType('none') and Set[XY]LabelType('').

2009-07-21

Updated SetNumberFormat to document a new hook variable to override locale, and fixed the example under SetXLabelType on rendering a Euro symbol.

2009-06-14

Released updated version of the manual to correspond to PHPlot-5.0.7.

PHPlot license is now LGPL-2.1.

Add information to SetRGBArray about using large and custom color maps.

Updates to Developer's Guide and Reference sections to document new feature allowing partial margin specifications in SetMarginsPixels and SetPlotAreaPixels.

2009-01-21

Released updated version of the manual to correspond to PHPlot-5.0.6.

2008-09-21

Changes were made for the next PHPlot release, including new label formatting types and mixed GD/TTF text types.

2008-07-22

Reference sections were changed to add a History subsection, detailing behavior in previous releases. This information was previously in the Notes subsections, but is not usually relevant to users of the current release.

2008-01-12

Update copyright year to 2008.

Updates to Part III, Developer's Guide for changes in internal functions used for text handling and margin/scale processing. Updated figure showing the plot layout.

Added new chapter to Developer's Guide on the callbacks feature, and added documentation of the 3 public functions used with callbacks to the Function Reference. The text is based on the `Callbacks` file which was included with PHPlot-5.0.4 (and will removed in the next release). Although this is now documented in the manual, the callbacks feature is still considered experimental.

2007-11-22

Added new section to Concepts chapter on error handling.

2007-10-19

    Released updated version of the manual to correspond to PHPlot-5.0.4, which should be released in the next few days.

    Added documentation for new function SetLegendStyle.

    Removed use of reference assignment (=&) in all inline examples, external examples, and reference material when creating a PHPlot object. Added note to PHPlot constructor reference explaining why.

    Added documentation for new function SetNumberFormat.

    Added new Part III - PHPlot Developer's Guide. This includes two layout figures and a list of internal functions. This is mostly intended as reference material for maintainers of PHPlot, but may be of use to others as well. More text may be added to this part in the future.

    Document the requirement that style arrays use zero-based sequential integer indexes only. This includes colors, point shapes/sizes, line widths, and line styles.

2006-12-01

    Released updated version of the manual to correspond to PHPlot-5.0rc3. Documented new Y data labels for bar charts and line/point suppression. Add a note on how to get borderless, unshaded bars (needed due to a recent bug fix).

    Updates were applied to the installation section to match the README included with 5.0rc3. Minimum PHP level corrected to 4.3.0, and noted that PHP 5 works, as this has now been tested. Deleted note about included examples and documentation; these have been removed from the release.

    Three examples which use random data where changed to use a fixed seed, so the results are repeatable.

    A new example was added, showing a bar chart with data labels.

2006-11-11

    Released manual dated 2006-11-11 to sourceforge.net. This is the first release of the manual on SourceForge. It corresponds to PHPlot-5.0rc2 with several patches applied.

2006-09-24

    Conversion of manual from DocBook-SGML to DocBook-XML. Reorganized, so the reference section is now alphabetical by function name, with a new chapter for category grouping of functions.

2005-02-27

    Limited release of draft manual dated 2005-02-27.

2005-02-06

    Limited release of draft manual dated 2005-02-06.