

# ChartDirector Ver 2.5

---

PHP Programmer's Manual

Advanced Software Engineering Limited

[www.advsofteng.com](http://www.advsofteng.com)

Copyright © 2002

ASE doc code: cdphp2504

# License Agreement

You should carefully read the following terms and conditions before using the ChartDirector software. Your use of the ChartDirector software indicates your acceptance of this license agreement. Do not use the ChartDirector software if you do not agree with the license agreement.

## Disclaimer of Warranty

The ChartDirector software and the accompanying files are distributed and licensed “as is”. Advanced Software Engineering Limited disclaims all warranties, either express or implied, including, but not limited to implied warranties of merchantability and fitness for a particular purpose. Should the ChartDirector software prove defective, the licensee assumes the risk of paying the entire cost of all necessary servicing, repair, or correction and any incidental or consequential damages. In no event will Advanced Software Engineering Limited be liable for any damages whatsoever (including without limitation damages for loss of business profits, business interruption, loss of business information and the like) arising out of the use or the inability to use the ChartDirector software even if Advanced Software Engineering Limited has been advised of the possibility of such damages.

## Copyright

The ChartDirector software is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The ChartDirector software is licensed, not sold. Title to the ChartDirector software shall at all times remain with Advanced Software Engineering Limited.

## Trial Version

The trial version of the ChartDirector software will produce acknowledgement messages in chart images generated by it.

Subjected to the conditions in this license agreement:

- You may use the unmodified trial version of the ChartDirector software without charge.
- You may redistribute the unmodified trial version of the ChartDirector software, provided you do not charge for it.
- You may embed the unmodified trial version of the ChartDirector software (or part of it), in a product and distribute the product, provided you do not charge for the product.

If you do not want the acknowledgement messages appearing in the charts, or you want to embed the ChartDirector software (or part of it) in a product that is not free, you must purchase a commercial license to use the ChartDirector software from Advanced Software Engineering Limited. Please refer to Advanced Software Engineering’s web site at [www.advsofteng.com](http://www.advsofteng.com) for details.

## Credits

The ChartDirector software is developed using code from the Independent JPEG Group and the FreeType team. Any software that is derived from the ChartDirector software must include the following text in its documentation. This applies to both the trial version of the ChartDirector software and to the commercial version of the ChartDirector software.

This software is based in part on the work of the Independent JPEG Group

This software is based in part of the work of the FreeType Team

## Ordering Information

Please refer to Advanced Software Engineering's web site at [www.advsofteng.com](http://www.advsofteng.com).

## Support Contact

Please refer to Advanced Software Engineering's web site at [www.advsofteng.com](http://www.advsofteng.com).

# Table of Contents

<b>Introduction.....</b>	<b>14</b>
Welcome to ChartDirector.....	14
Programming Languages.....	17
Supported Platforms .....	17
<b>Installation.....</b>	<b>18</b>
<b>Windows Installation .....</b>	<b>18</b>
PHP Version Compatibility on Windows.....	18
Installing ChartDirector for PHP on Windows.....	18
Installing Sample Scripts on Windows .....	19
Installing the ChartDirector License on Windows .....	19
Using ChartDirector for PHP on Windows.....	20
<b>Linux/FreeBSD/Solaris Installation .....</b>	<b>20</b>
Version Compatibility on Linux/FreeBSD/Solaris .....	20
Installing ChartDirector for PHP on Linux/FreeBSD/Solaris .....	21
Viewing the Sample PHP Scripts .....	22
Installing the ChartDirector License on Linux/FreeBSD/Solaris .....	22
Using ChartDirector for PHP on Linux/FreeBSD/Solaris .....	23
<b>Getting Started.....</b>	<b>24</b>
The First Project.....	24
Incorporating Charts in Web Pages .....	26
ChartDirector Object Model Overview .....	26
<b>Pie Chart.....</b>	<b>28</b>
Simple Pie Chart .....	28
3D Pie Chart.....	29
Side Label Layout .....	30
Circular Label Layout .....	31
Pie Chart with Legend.....	33
Background Coloring.....	34
Metallic Background Coloring.....	36
Coloring and Wallpaper.....	37
Text Style and Colors.....	39
Sectors with Borders .....	41
3D Angles.....	42

3D Depth.....	43
3D Shadow Mode.....	44
Layout Angle and Direction .....	45
<b>XY Chart.....</b>	<b>48</b>
Simple Bar Chart.....	48
3D Bar Chart .....	49
Multi-Color Bar Chart .....	50
Stacked Bar Chart.....	52
Multi-Bar Chart .....	53
Depth Bar Chart .....	55
Bar Labels .....	56
Bar Gap.....	58
Positive Negative Bars .....	59
Horizontal Bar Chart.....	61
Dual Horizontal Bar Charts.....	63
Simple Line Chart.....	66
Enhanced Line Chart .....	67
3D Line Chart.....	69
Multi-Line Chart.....	70
Symbol Line Chart.....	72
Trend Line Chart .....	74
Arbitrary XY Line Chart .....	76
Discontinuous Lines .....	78
Scatter Chart.....	81
Scatter Trend Chart.....	83
Simple Area Chart.....	85
3D Area Chart .....	86
Line Area Chart .....	87
Stacked Area Chart.....	89
3D Stacked Area Chart .....	90
Depth Area Chart .....	92
High-Low-Open-Close Chart.....	93
Candle Stick Chart .....	96
Combination Chart .....	99
Marks and Zones .....	101
Marks and Zones (2) .....	103
Dual Y-Axis .....	105
Dual X-Axis .....	107

Text Style and Colors .....	109
Background and Wallpaper .....	111
Log Scale Axis .....	113
Y-Axis Scaling .....	114
Tick Density.....	116
<b>Using Data Sources with ChartDirector .....</b>	<b>118</b>
Using ChartDirector in Web Pages .....	118
Strategies in Passing Data to ChartDirector .....	118
Database Sample Code Overview .....	119
Perform Query in the ChartDirector PHP Page .....	120
Pass Data to ChartDirector as Query Parameters .....	124
Use Session Variables to Store the Chart.....	128
<b>Clickable Charts.....</b>	<b>129</b>
Operation Overview .....	129
Simple Clickable Charts .....	130
Javascript Clickable Charts .....	137
Custom Clickable Objects .....	141
Database Clickable Charts.....	146
Using Different Handlers for Different Data Points .....	150
Making Other Objects Clickable .....	151
<b>ChartDirector API Reference .....</b>	<b>152</b>
<b>Important Notes.....</b>	<b>152</b>
Notes for Perl Developers .....	152
Notes for C++ Developers.....	152
<b>International Character Sets Support .....</b>	<b>154</b>
What is Unicode and What is UTF8? .....	154
Writing Code in UTF8.....	154
ISO-8859-1 (Latin1) Western European Character Set.....	154
Displaying International Characters .....	154
<b>Data Types .....</b>	<b>155</b>
Color Specification .....	155
Font Specification.....	158
Alignment Specification .....	160
No Value Specification .....	160
<b>Data and Text Formatting.....</b>	<b>161</b>

Parameter Substitution.....	161
Numeric Formatting.....	163
<b>Draw Objects .....</b>	<b>164</b>
Box.....	164
TextBox.....	166
Line .....	170
<b>BaseChart .....</b>	<b>172</b>
destroy .....	173
setSize .....	174
setBackground .....	174
setBorder.....	175
setWallpaper .....	175
setBgImage .....	176
setTransparentColor .....	176
addTitle .....	177
addTitle2 .....	177
addLegend .....	178
getDrawArea .....	179
addDrawObj .....	180
addText .....	180
addLine .....	181
setColor.....	182
setColors.....	182
setColors2 .....	182
getColor.....	183
dashLineColor .....	183
patternColor .....	184
patternColor2 .....	184
gradientColor.....	185
gradientColor2.....	186
layout .....	187
makeChart.....	187
makeChart2.....	188
makeChart3.....	189
getHTMLImageMap .....	189
<b>LegendBox.....</b>	<b>191</b>
addKey .....	192

setText .....	193
setKeySize .....	193
getImageCoor2 .....	194
getHTMLImageMap .....	195
<b>PieChart .....</b>	<b>197</b>
PieChart Constructor .....	199
setPieSize .....	199
set3D .....	200
setStartAngle .....	200
setLabelFormat .....	201
setLabelStyle .....	202
setLabelLayout .....	202
setLineColor .....	204
setLabelPos .....	205
setData .....	206
sector .....	206
<b>Sector .....</b>	<b>207</b>
setExplode .....	207
setLabelFormat .....	208
setLabelStyle .....	208
setLabelLayout .....	209
setLabelPos .....	209
setColor .....	210
getImageCoor .....	211
getLabelCoor .....	211
<b>XYChart .....</b>	<b>212</b>
XYChart Constructor .....	214
xAxis .....	215
xAxis2 .....	215
yAxis .....	216
yAxis2 .....	216
syncYAxis .....	217
setYAxisOnRight .....	218
swapXY .....	218
setPlotArea .....	219
addBarLayer .....	220
addBarLayer2 .....	220



addBarLayer3.....	221
addLineLayer .....	222
addLineLayer2 .....	223
addScatterLayer.....	224
addTrendLayer.....	225
addTrendLayer2.....	225
addAreaLayer.....	226
addAreaLayer2.....	227
addHLOCLayer .....	228
addHLOCLayer2 .....	228
addCandleStickLayer .....	229
<b>PlotArea .....</b>	<b>230</b>
setBackground .....	230
setBackground2 .....	231
setGridColor .....	231
setGridWidth .....	232
<b>BaseAxis .....</b>	<b>232</b>
setLabelStyle.....	233
setLabelGap.....	233
setTitle.....	234
setTitlePos .....	234
setColors .....	235
setWidth .....	236
setTickLength.....	236
setTickLength2.....	237
addMark .....	237
addZone .....	238
<b>XAxis .....</b>	<b>239</b>
setLabels.....	240
setLinearScale .....	241
setLinearScale2 .....	242
addLabel .....	243
setIndent .....	243
<b>YAxis .....</b>	<b>244</b>
setLinearScale .....	245
setLogScale .....	246

setLogScale2 .....	246
setAutoScale .....	247
setTickDensity .....	249
setTopMargin .....	249
setLabelFormat .....	250
<b>Mark.....</b>	<b>250</b>
setValue .....	251
setMarkColor .....	251
setLineWidth .....	252
setDrawOnTop .....	252
<b>Layer.....</b>	<b>252</b>
set3D .....	253
setLineWidth .....	254
setBorderColor .....	254
setDataCombineMethod .....	255
setLegend .....	256
addDataSet .....	256
getDataSet .....	257
setXData .....	258
setXData2 .....	259
getXCoor .....	259
getYCoor .....	260
setDataLabelFormat.....	261
setDataLabelStyle .....	261
addCustomDataLabel .....	262
setAggregateLabelFormat.....	263
setAggregateLabelStyle .....	264
addCustomAggregateLabel.....	265
getImageCoor .....	266
getImageCoor2 .....	267
getHTMLImageMap .....	267
<b>BarLayer.....</b>	<b>268</b>
setBarGap .....	270
<b>LineLayer .....</b>	<b>270</b>
setGapColor .....	272
setImageMapWidth .....	272

<b>ScatterLayer .....</b>	<b>273</b>
setGapColor .....	274
setImageMapWidth .....	274
<b>TrendLayer.....</b>	<b>275</b>
setImageMapWidth .....	276
<b>AreaLayer.....</b>	<b>277</b>
<b>HLOCLayer .....</b>	<b>278</b>
setDataGap .....	279
<b>CandleStickLayer .....</b>	<b>280</b>
setDataGap .....	281
<b>DataSet.....</b>	<b>281</b>
setDataName .....	282
setDataColor .....	282
setUseYAxis2.....	283
setLineWidth .....	284
setDataSymbol.....	284
setDataSymbol2.....	285
setDataSymbol3.....	286
setDataLabelFormat.....	286
setDataLabelStyle .....	287
<b>DrawArea .....</b>	<b>288</b>
DrawArea Constructor.....	290
destroy .....	290
setSize .....	291
getWidth.....	291
getHeight.....	292
setBgColor .....	292
pixel.....	292
getPixel .....	293
line .....	293
hline .....	294
vline.....	294
arc.....	295
rect.....	295
polygon.....	296

surface .....	297
sector .....	298
cylinder .....	299
circle .....	300
fill .....	301
fill2 .....	301
text .....	302
text2 .....	303
text3 .....	304
text4 .....	304
close .....	305
merge .....	306
tile .....	306
load .....	307
loadGIF .....	307
loadPNG .....	308
loadJPG .....	308
loadWMP .....	309
out .....	309
outGIF .....	309
outGIF2 .....	310
outPNG .....	310
outPNG2 .....	311
outJPG .....	311
outJPG2 .....	312
outWMP .....	312
outWMP2 .....	313
setPaletteMode .....	313
setDitherMethod .....	314
setTransparentColor .....	315
setAntiAliasText .....	315
setInterlace .....	316
setColorTable .....	316
getARGBColor .....	317
dashLineColor .....	317
patternColor .....	318
patternColor2 .....	319
gradientColor .....	320

gradientColor2.....	321
<b>TTFText .....</b>	<b>322</b>
getWidth .....	322
getHeight.....	323
getLineHeight.....	323
getLineDistance .....	323
draw .....	324
<b>ChartDirector API Index .....</b>	<b>325</b>

# Introduction

## Welcome to ChartDirector

ChartDirector is a powerful software graphics library for creating professionally looking charts in PNG, JPEG, WBMP or alternative GIF format.

The key features of ChartDirector are as follows.

- **Server Oriented – Fast and Efficient**

ChartDirector is specially designed to operate on the server side. It handles multiple concurrent requests quickly, while its light weight design consumes only a small amount of system resources.

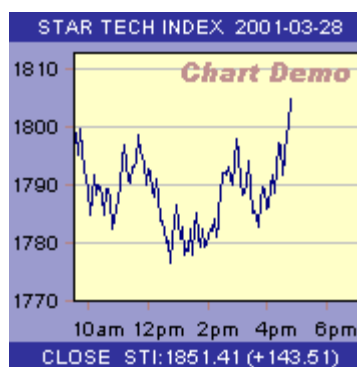
- **Flexibility – Allows you to design the chart you want**

ChartDirector models charts as objects hierarchies, consisting of title objects, legend objects, axis objects, data set objects, etc. Their properties are controllable through the ChartDirector API, providing you tremendous flexibility to design the charts you want.

ChartDirector employs a layering architecture. You may combine different type of layers on to creating any arbitrary “combo” charts. By supporting alpha transparency coloring, bottom layers remain visible even if covered by top layers.

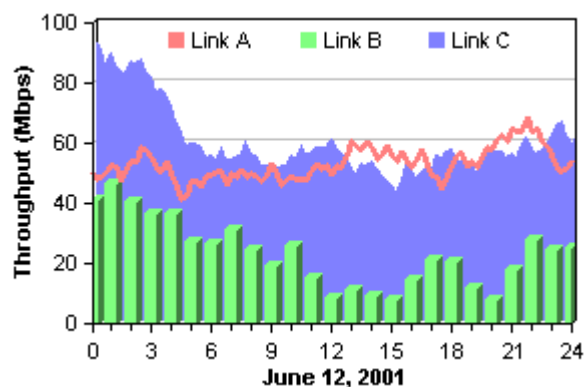
ChartDirector even includes a general-purpose graphics API, so you can add annotations, custom text, shapes and images to decorating your charts the chart in custom ways. It is possible to use ChartDirector as a general-purpose graphics library.

### Sample Charts



Simple yet elegant design. Make possible by the versatile object-oriented ChartDirector API.

With a dimension of 180 x 180 pixels, its file size is just 1414 bytes.



Another chart design using the layering architecture of ChartDirector. Developers can combine multiple layers to make arbitrary combo charts. [300 x 200 pixels, 2217 bytes]

- **Interactive – Clickable charts ideal for providing “drill-down” capabilities**

The ChartDirector API generates image maps for all charts types. These image maps are ideal for providing “drill-down” capabilities on your charts.

For example, in a bar chart, the image map can distinguish which bar segment the user has clicked. This allows your system to perform different actions based on what bar segment the user has clicked.

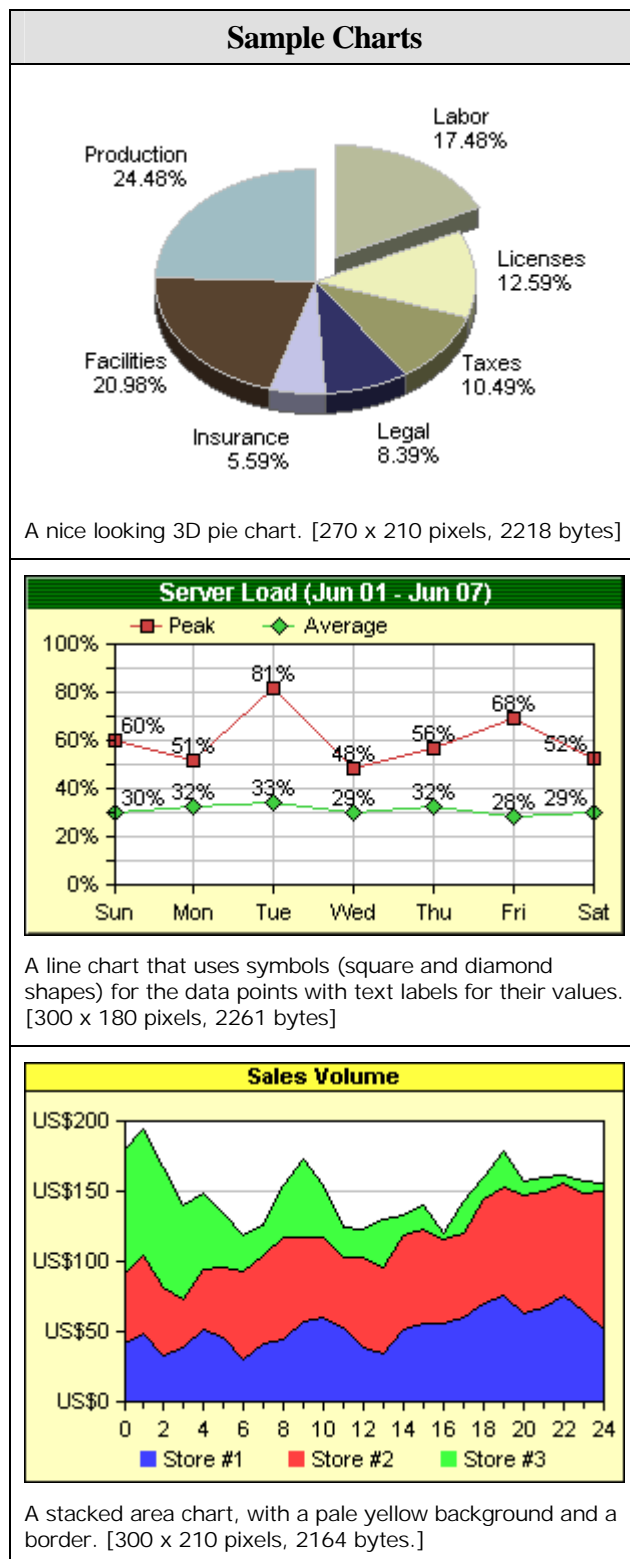
ChartDirector can even generate image maps for legend keys, custom text box and labels, so you can add custom buttons or hypertext links on the charts.

- **Internet/Intranet Friendly – Fast download time and maximum browser compatibility**

ChartDirector produces charts that are very small in file size - a key benefit for web applications. A typical 200 x 200 chart is only 500 - 2K bytes. If you find it hard to believe, visit our [chart gallery](#) to see them yourselves.

With the small file sizes, you can include many charts in your web page without worrying about download times.

ChartDirector charts are in PNG, JPEG, WBMP and alternative GIF formats, which is compatible with virtually all browsers and viewers, including those for mobile computing devices. No Java or ActiveX control is required on the browser side. You can even cut and paste ChartDirector charts onto email or word documents.



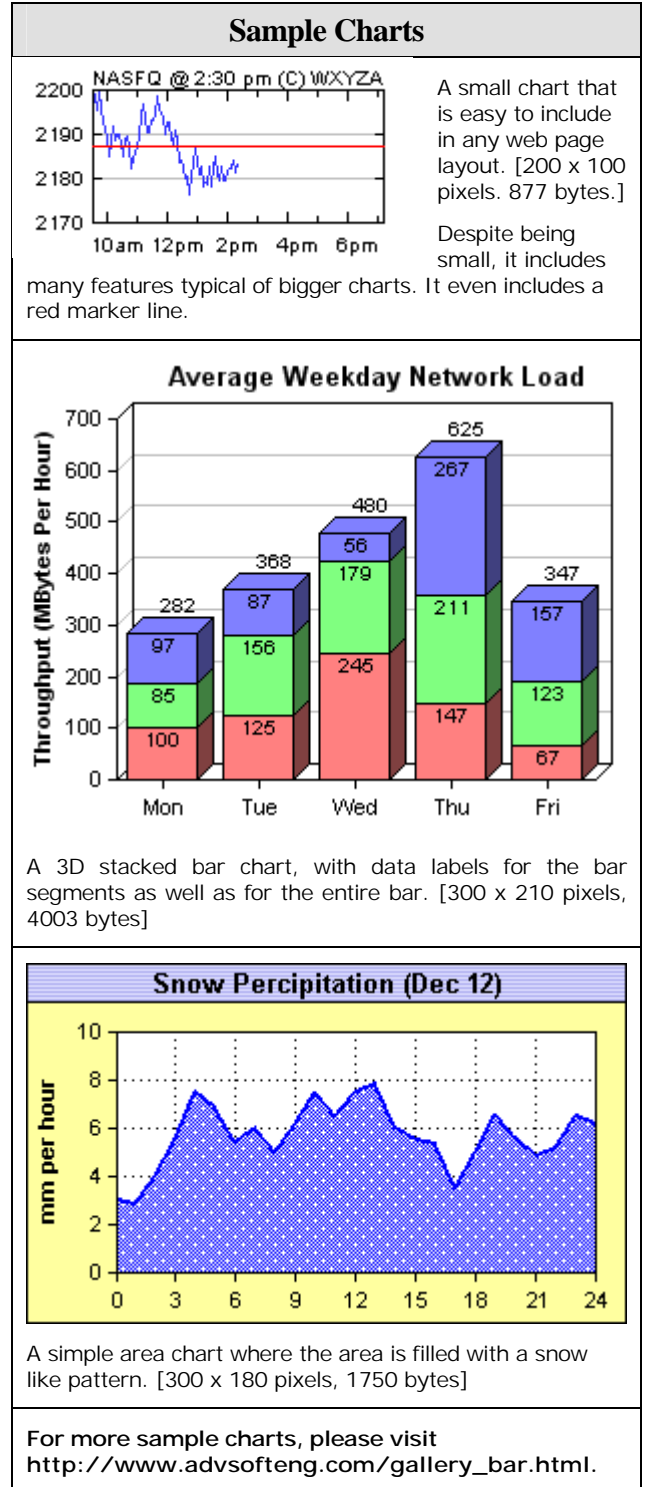
- **Ease of Use**

ChartDirector's object oriented API is intuitive to learn and use. Most charts can be produced with just a few lines of code.

To make even easier to use, ChartDirector includes comprehensive documentation, with step-by-step tutorials and numerous sample code.

- **Unicode support**

By supporting Unicode (or UTF8) text strings, ChartDirector can draw charts with characters from multiple languages.





## Programming Languages

The ChartDirector core is written using C++, so it is very fast and efficient. Language bindings are used to bind the ChartDirector API to other languages. The language bindings available include:

- ASP / COM / Visual Basic / VBScript / JScript
- .NET (C#, VB.NET, Managed C++, JScript.NET)
- PHP
- Perl
- Python
- C++

## Supported Platforms

ChartDirector is supported on Windows 2000/NT/XP/ME/98, Linux, FreeBSD and Solaris (Sparc).

---

# Installation

## Windows Installation

This section describes how to install ChartDirector for PHP under Windows. For the Linux/FreeBSD/Solaris version, please refer to the [Linux/FreeBSD/Solaris Installation](#) section.

### PHP Version Compatibility on Windows

ChartDirector is designed to be compatible with the following PHP versions:

- PHP Ver 4.0.4pl1
- PHP Ver 4.0.5
- PHP Ver 4.0.6
- PHP Ver 4.1.0
- PHP Ver 4.1.1
- PHP Ver 4.2.0
- PHP Ver 4.2.1

ChartDirector does not support PHP versions earlier than PHP 4.0.4pl1.

As of the writing of this documentation, PHP Ver 4.2.1 is the latest PHP version.

### Installing ChartDirector for PHP on Windows

ChartDirector for PHP on Windows is distributed as a zip file. To install ChartDirector:

- Create an empty subdirectory and unzip the ChartDirector distribution zip file in it.
- The ChartDirector for PHP module and its supporting DLLs are in the “lib” subdirectory. Copy all files in the “lib” subdirectory to the PHP extension directory in your system.

The PHP extension directory is determined by the “extension\_dir” variable in the “php.ini” file. Please refer to your PHP documentation for more information.

If your PHP extension directory consists of a relative path name, please make sure you know where it is relative to. It could be relative to the PHP executable, Apache executable, or some other directory, depending on your system configuration. To avoid confusion, you may want to modify your PHP extension directory to use an absolute path name instead.

- ChartDirector for PHP will dynamically load the ChartDirector PHP extension when needed. However, for some PHP systems, dynamic loading of extensions may be disabled for administrative or technical reasons.

Also, if your PHP is running as an Apache module on Windows, PHP will automatically disable dynamic loading of extensions. It is because Apache on Windows uses multi-threading, and PHP does not support dynamic loading of extensions under multi-threading.

If dynamic loading of PHP extensions is disabled in your system, you can pre-load the ChartDirector PHP extension by including one of the following lines in your PHP configuration file (“php.ini”) depending on your PHP version.

```
    ;for PHP version 4.0.4pl1
    extension=phpchartdir404.dll
or
    ;for PHP version 4.0.5 or 4.0.6
    extension=phpchartdir405.dll
or
    ;for PHP version 4.1.0 or 4.1.1, 4.1.2 and 4.2.0
    extension=phpchartdir410.dll
or
    ;for PHP version 4.2.1 (or above?)
    extension=phpchartdir421.dll
```

## Installing Sample Scripts on Windows

ChartDirector for PHP comes with a number of sample PHP scripts under the “phpdemo” directory. They are good tutorials and demonstrations on how to use ChartDirector.

To view the sample PHP scripts, you need to copy them to your web server directory. You may then point your browser to the starting page of the sample scripts, which is called “index.php”.

If ChartDirector is installed correctly, you should see sample charts being generated by the sample PHP programs as you click through the links in the “index.php” page.

If for some reason, you cannot see the charts, please click on the “check installation” link on the “index.php” page. This will show the error messages, which may be useful to diagnose the problem.

## Installing the ChartDirector License on Windows

If you have purchased a license to use ChartDirector, you should have a license code delivered to you via email and postal mail.

To install the license code, follow the steps below:

- Save the license code in an ASCII text file, and name the file "chartdir.lic". The ASCII text file should contain just one line, which is the license code. A sample license file is available at <http://www.advsofteng.com/chartdir.lic> for your reference.
- Put the license file in the same directory as "chartdir.dll", which should be in the directory where the ChartDirector for PHP module is installed.
- Make sure the web server "anonymous user" has sufficient privileges to read the license file.
- If you are using PHP as an Apache module, you may need to restart your Apache server for the license to take effect.

## Using ChartDirector for PHP on Windows

Any PHP script that uses ChartDirector need to include the ChartDirector script "phpchartdir.php". Please copy the "phpchartdir.php" (located in the "lib" subdirectory) to your PHP scripts directory. In your PHP scripts, please include the following statement:

```
require_once("phpchartdir.php");
```

You may also put the "phpchartdir.php" in other subdirectory. In this case, you need to include the path name in the "require\_once" statement. Please refer to the PHP documentation for more details on how to use "require\_once".

## Linux/FreeBSD/Solaris Installation

This section describes how to install ChartDirector for PHP under Linux/FreeBSD/Solaris. For the Windows version, please refer to the [Windows Installation](#) section.

## Version Compatibility on Linux/FreeBSD/Solaris

ChartDirector is designed to be compatible to the following PHP versions:

- PHP Ver 4.0.4pl1
- PHP Ver 4.0.5
- PHP Ver 4.0.6
- PHP Ver 4.1.0
- PHP Ver 4.1.1
- PHP Ver 4.2.0
- PHP Ver 4.2.1

As of the writing of this documentation, PHP Ver 4.2.1 is the latest PHP version.

## Installing ChartDirector for PHP on Linux/FreeBSD/Solaris

ChartDirector for PHP on Linux/FreeBSD/Solaris is distributed as a tar.gz file. To install ChartDirector:

- Extract the files from the tar.gz file by using:

```
gunzip <chartdir_file_name>.tar.gz
tar xvf <chartdir_file_name>.tar
```

- The ChartDirector for PHP module and its supporting shared objects and data are in the “lib” subdirectory. Copy everything in the “lib” subdirectory (including any subdirectory it contains) to the PHP extension directory in your system. Assuming you have changed to the “ChartDirector” directory after extracting the files, the command to do this is:

```
cp -R lib/* <php_ext_dir>
```

where <php\_ext\_dir> is the PHP extension directory in your system.

The PHP extension directory is determined by the “extension\_dir” variable in the “php.ini” file. Please refer to your PHP documentation for more information.

If your PHP extension directory consists of a relative path name, please make sure you know where it is relative to. It could be relative to the PHP executable, Apache executable, or some other directory, depending on your system configuration. To avoid confusion, you may want to modify your PHP extension directory to use an absolute path name instead.

## Verifying if Dynamic Loading of PHP Extensions is Enabled

ChartDirector for PHP will dynamically load the ChartDirector PHP extension when needed. However, for some PHP systems, dynamic loading of extension may be disabled for administrative or technical reasons.

Also, if your PHP is running as an Apache 2.0 module, PHP will automatically disable dynamic loading of extensions. It is because Apache 2.0 uses multi-threading, and PHP does not support dynamic loading of extensions under multi-threading.

If dynamic loading of PHP extensions is disabled in your system, you can pre-load the ChartDirector PHP extension by including one of the following lines in your PHP configuration file (“php.ini”) depending on your PHP version.

- If you are using PHP as a CGI, or as an Apache 1.x module, please use one of the following lines:

```
;for PHP version 4.0.4p11
extension=phpchartdir404.dll
or
;for PHP version 4.0.5 or 4.0.6
extension=phpchartdir405.dll
or
;for PHP version 4.1.0 or 4.1.1, 4.1.2 and 4.2.0
extension=phpchartdir410.dll
or
```

```
;for PHP version 4.2.1 (or above?)
extension=phpchartdir421.dll
```

- If you are using PHP as an Apache 2.x module, you would need to use the multi-threaded version of the ChartDirector PHP extensions. Please use one of the following lines:

```
;for PHP version 4.0.4pl1
extension=phpchartdir404mt.dll
or
;for PHP version 4.0.5 or 4.0.6
extension=phpchartdir405mt.dll
or
;for PHP version 4.1.0 or 4.1.1, 4.1.2 and 4.2.0
extension=phpchartdir410mt.dll
or
;for PHP version 4.2.1 (or above?)
extension=phpchartdir421mt.dll
```

## Viewing the Sample PHP Scripts

ChartDirector for PHP comes with a number of sample PHP scripts under the “phpdemo” directory. They are good tutorials and demonstrations on how to use ChartDirector.

To view the sample PHP scripts, you need to copy them to your web server directory. You may then point your browser to the starting page of the sample scripts, which is called “index.php”.

Note that the sample PHP scripts assume you are running PHP as an Apache module. If you are running PHP as a CGI script, or is using PHP with other kind web server, you may need to modify the sample scripts by adding “#!<path\_to\_php\_interpreter>” to the first line of “.php” file.

If ChartDirector is installed correctly, you should see sample charts being generated by the sample PHP programs as you click through the links in the “index.php” page.

If for some reason, you cannot see the charts, please click on the “check installation” link on the “index.php” page. This will show the error messages, which may be useful to diagnose the problem.

## Installing the ChartDirector License on Linux/FreeBSD/Solaris

If you have purchased a license to use ChartDirector, you should have a license code delivered to you via email and postal mail.

To install the license code, follow the steps below:

- Save the license code in an ASCII text file, and name the file “chartdir.lic”. The ASCII text file should contain just one line, which is the license code. A sample license file is available at <http://www.advsofteng.com/chartdir.lic> for your reference.
- Put the license file in the same directory as “libchartdir.so”, which should be in the directory where the ChartDirector for PHP module is installed.

- Make sure the web server "anonymous user" has sufficient privileges to read the license file.
- If you are using PHP as an Apache module, you may need to restart your Apache server for the license to take effect.

## Using ChartDirector for PHP on Linux/FreeBSD/Solaris

Any PHP script that uses ChartDirector need to include the ChartDirector script "phpchartdir.php". Please copy the "phpchartdir.php" (located in the "lib" subdirectory) to your PHP scripts directory. In your PHP scripts, please include the following statement:

```
require_once("phpchartdir.php");
```

You may also put the "phpchartdir.php" in other subdirectory. In this case, you need to include the path name in the "require\_once" statement. Please refer to the PHP documentation for more details on how to use "require\_once".

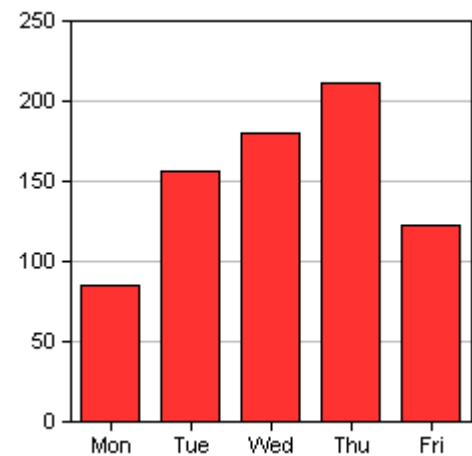
# Getting Started

## The First Project

To get a feeling of the ChartDirector, and to verify the ChartDirector development environment is set up properly, we will begin by building a very simple chart.

In this first project, we will draw a simple bar chart as shown on the right.

(Note that the trial version of ChartDirector will include a small acknowledgement message at the bottom of chart. The message will disappear in the licensed version of ChartDirector.)



(The following program is available as “phpdemo/simplebar.php”).

```
<?php
include("phpchartdir.php");

#The data for the bar chart
$data = array(85, 156, 179.5, 211, 123);

#The labels for the bar chart
$labels = array("Mon", "Tue", "Wed", "Thu", "Fri");

#Create a XYChart object of size 250 x 250 pixels
$c = new XYChart(250, 250);

#Set the plotarea at (30, 20) and of size 200 x 200 pixels
$c->setPlotArea(30, 20, 200, 200);

#Add a bar chart layer using the given data
$c->addBarLayer($data);

#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
```



```
?>
```

The code is explained below :

- `include( "phpchartdir.php" );`

This line includes the ChartDirector module so that you can use it.

- `$c = new XYChart(250, 250);`

The first step in creating any chart in ChartDirector is to create the appropriate chart object. In this example, we create an XYChart object that represents a chart 250 pixels wide and 250 pixels high. The XYChart object is used in ChartDirector for any chart that has an x axis and y axis, including the bar chart that we are drawing.

- `$c->setPlotArea(30, 30, 200, 200);`

The second step in creating a bar chart is to specify where exactly should we draw the bar chart. This is by specifying the rectangle that contains the bar chart. The rectangle is specified by using the (x, y) coordinates of the top-left corner, together with its width and height.

For this simple bar chart, we will use the majority of the chart area to draw the bar chart. We will leave some margin to allow for the text labels on the axis. In the above code, the top-left corner is set at (30, 30), and both the width and height is set at 200 pixels. Since the entire chart is 250 x 250 in size, there will be 20 to 30 pixels margin for the text labels.

Note that all ChartDirector charts use a coordinate system that is customary on computer screen. The x axis is the horizontal axis from left to right, while the y axis is the vertical axis from top to bottom. The origin (0, 0) is at the top-left corner.

For more complex charts which may contain titles, legend box and other things, we can use this method (and other methods) to design the exact layout of the entire chart.

- `$c->addBarLayer($data);`

The above line adds a bar chart layer to the XYChart. In ChartDirector, all charts that has an x axis and y axis are all represented as layers in the XYChart. An XYChart can contain a lot of layers. You can create “combination charts” easily (e.g. a chart containing both a line chart overlapped with a bar chart) by combining different layers on the same chart.

In the above line of code, the argument is an array of numbers representing the data points.

- `$c->xAxis->setLabels($labels);`

The above line of code sets the labels on the x-axis. The first method xAxis retrieves the x-axis object inside the XYChart object. The second method setLabels binds the text labels to the x-axis object. The argument to the setLabels method is an array of text string.

- `header("Content-type: image/png");  
print($c->makeChart2(PNG));`

Up to step 4, the chart is completed. We need to output it somehow. In our simple project, we output the chart directly to a web page in PNG format.

Apart from PNG, the ChartDirector supports other file formats (e.g. JPEG, alternative GIF, WBMP) and output methods (e.g. output to a file using [makeChart](#)).

## Incorporating Charts in Web Pages

In any real application, the chart image is to be displayed inside a HTML web page. To do this, you just need to write the web page normally, and use an <IMG> tag to include the chart in the HTML page.

For an ordinary image, typically you enter the path name of the image in the <IMG> tag. For charts generated dynamically using ChartDirector, you can enter the URL that generates the chart in the <IMG> tag.

For example:

```
<HTML>
<BODY>

<h1>Hello World!</h1>
<p>Hi, this is my first web page with ChartDirector charts.</p>

<IMG SRC=" http://aaa.bbb.ccc.ddd/phpdemo/simplebar.php">

More HTML elements .....

</BODY>
</HTML>
```

You can include multiple charts in the same page by using multiple <IMG> tags.

## ChartDirector Object Model Overview

This section will give a very brief high level overview of the ChartDirector object model.

At the top level, ChartDirector classifies charts into two main classes – [PieChart](#) and [XYChart](#). Both are derived from the superclass [BaseChart](#).

The BaseChart represents features common to all ChartDirector charts, such as the chart background, legend box, titles and custom text boxes. It also contains a [DrawArea](#) tool, which allows you to add arbitrary custom drawings to the chart.

The PieChart class, as its name implies, represents pie charts. PieChart objects contain [Sector](#) objects, which represent the sectors in the pie charts.

The XYChart class represents all chart types that have x and y axes. Each XYChart object contains a [PlotArea](#) object, where the chart is actually drawn. An XYChart also contains two [XAxis](#) objects and two [YAxis](#) objects. These four axes are located at the 4 sides of the PlotArea.

The XAxis and YAxis classes are subclasses of the [BaseAxis](#), which represents the features common to both types of axis.

The actual chart types in an XYChart are implemented as layers. The layers that are supported are listed in the following table. An XYChart can contain many layers, so combination charts (e.g. combining line chart and bar chart) can be created easily.

Layer	Description
<a href="#">BarLayer</a>	Represent all style of bar charts, such as simple bar chart, multi-bar chart, stacked bar chart, etc. Bar charts can be vertical or horizontal.
<a href="#">LineLayer</a>	Represent line charts.
<a href="#">ScatterLayer</a>	Represent scatter charts.
<a href="#">TrendLayer</a>	Represent trend line charts. A trend line the best fit straight line computed using the least square method.
<a href="#">AreaLayer</a>	Represent area charts. Like bar charts, area charts support stacked data.
<a href="#">HLOCLayer</a>	Represent high-low-open-close charts.
<a href="#">CandleStickLayer</a>	Represent candlestick charts.

All of the layers above are subclass of the [Layer](#) class. The Layer class represents features that are common to all layer types.

Each chart layer contains one or more [DataSet](#) objects. A DataSet object represents a data series. Most chart layers support multiple data sets and can combine the data series in various ways. For example, in a bar chart, multiple data series can be combined to form a “stacked bar chart” or “multi-bar chart”.

For more details on the above ChartDirector objects and various other ChartDirector objects, please refer to the [ChartDirector API Reference](#).

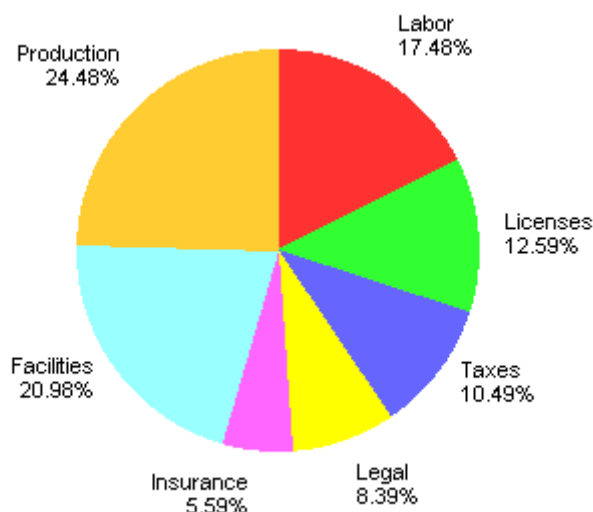
# Pie Chart

In this chapter we will discuss how to create pie charts using the ChartDirector API through a number of examples. Details of the ChartDirector API can be found in the chapter [ChartDirector API Reference](#).

## Simple Pie Chart

In this example, a very simple pie chart is created demonstrating the basic steps in creating pie charts.

- Create a pie chart object using “new [PieChart](#)”.
- Specify the center and radius of the pie using the [setPieSize](#) method.
- Specify the data used to draw the pie using the [setData](#) method.
- Generate the chart using the [makeChart2](#) method.



(The following program is available as “phpdemo/simplepie.php”).

```
<?php
include("phpchartdir.php");

#The data for the pie chart
$data = array(25, 18, 15, 12, 8, 30, 35);

#The labels for the pie chart
$labels = array("Labor", "Licenses", "Taxes", "Legal", "Insurance",
    "Facilities", "Production");

#Create a PieChart object of size 360 x 300 pixels
$c = new PieChart(360, 300);

#Set the center of the pie at (180, 140) and the radius to 100 pixels
$c->setPieSize(180, 140, 100);
```

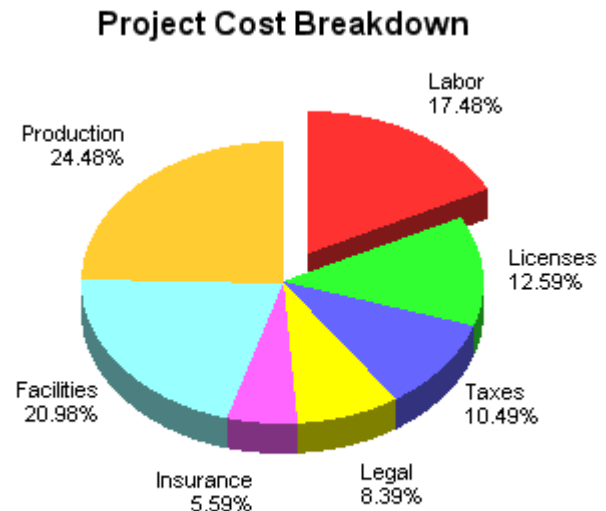
```
#Set the pie data and the pie labels
$c->setData($data, $labels);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## 3D Pie Chart

This example extends the previous simple pie example by introducing three more features of the ChartDirector:

- Add a title to the chart using the [addTitle](#) method
- Draw the pie in 3D using the [set3D](#) method
- Explode a sector using the [setExplode](#) method



(The following program is available as “phpdemo/threedpie.php”).

```
<?php
include("phpchartdir.php");

#The data for the pie chart
$data = array(25, 18, 15, 12, 8, 30, 35);

#The labels for the pie chart
$labels = array("Labor", "Licenses", "Taxes", "Legal", "Insurance",
    "Facilities", "Production");

#Create a PieChart object of size 360 x 300 pixels
$c = new PieChart(360, 300);

#Set the center of the pie at (180, 140) and the radius to 100 pixels
$c->setPieSize(180, 140, 100);

#Add a title to the pie chart
$c->addTitle("Project Cost Breakdown");

#Draw the pie in 3D
$c->set3D();
```

```
#Set the pie data and the pie labels
$c->setData($data, $labels);

#Explode the 1st sector
$sectorObj = $c->sector(0);
$sectorObj->setExplode();

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## Side Label Layout



This example illustrates an alternative sector label layout method supported by ChartDirector.

In previous examples, sector labels are positioned circularly around the pie. In the current example, the labels are positioned on the left and right sides of the chart. In ChartDirector terminology, the former method is called “circular label layout”, while the latter method is called “side label layout”.

“Circular label layout” usually uses less space and is the default layout method. However, if the pie chart contains a lot of small sectors, the labels may overlap with each others. It is because there may be insufficient space on the pie perimeter to position the labels.

“Side label layout” has the advantages that it can avoid label overlapping unless under the most extreme cases (e.g. even the whole chart does not have enough space for all the labels). In “side label layout”, labels will automatically shift up and down to avoid overlapping.

You can choose which label layout method to use by using the [setLabelLayout](#) method.

(The following program is available as “phpdemo/sidelabelpie.php”.)

```
<?php
include("phpchartdir.php");

#The data for the pie chart
```

```

$data = array(25, 18, 15, 12, 8, 30, 35);

#The labels for the pie chart
$labels = array("Labor", "Licenses", "Taxes", "Legal", "Insurance",
    "Facilities", "Production");

#Create a PieChart object of size 500 x 230 pixels
$c = new PieChart(500, 230);

#Set the center of the pie at (250, 120) and the radius to 100 pixels
$c->setPieSize(250, 120, 100);

#Add a title box using 14 points Times Bold Italic as font
$c->addTitle("Project Cost Breakdown", "timesbi.ttf", 14);

#Draw the pie in 3D
$c->set3D();

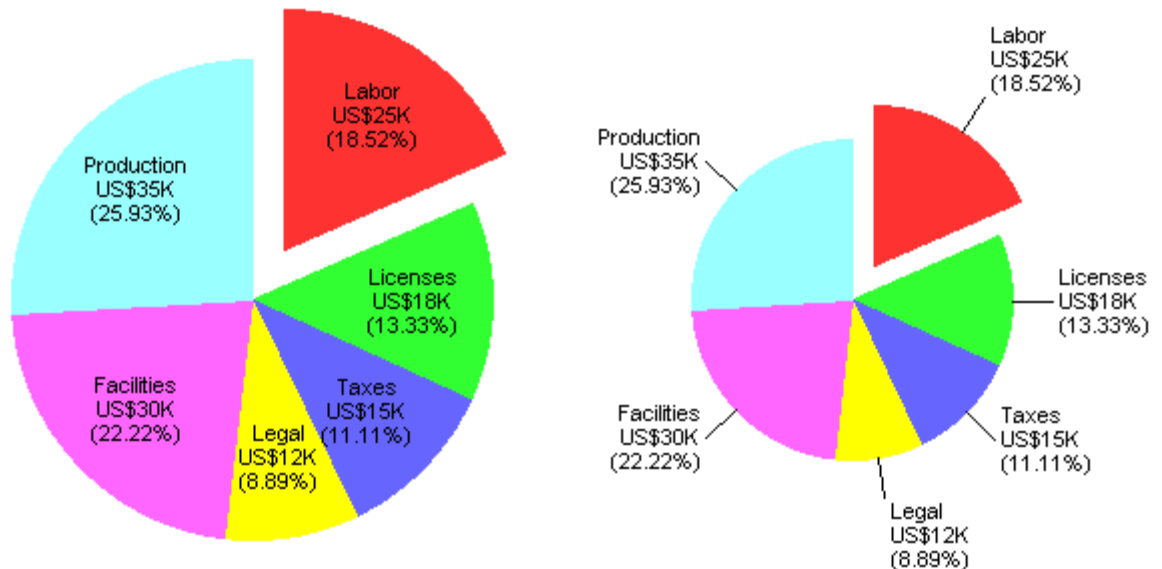
#Use the side label layout method
$c->setLabelLayout(SideLayout);

#Set the pie data and the pie labels
$c->setData($data, $labels);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## Circular Label Layout



This example demonstrates how to control the positions of sector labels in circular label layout, and to include join lines to connect the labels to the sector perimeter. The [setLabelPos](#) method is used to

achieve these effects. It also demonstrates how to specify the format of the label using the [setLabelFormat](#) method.

(The following program is available as “phpdemo/labelpie.php”.)

```
<?php
include("phpchartdir.php");

#The data for the pie chart
$data = array(25, 18, 15, 12, 30, 35);

#The labels for the pie chart
$labels = array("Labor", "Licenses", "Taxes", "Legal", "Facilities",
    "Production");

#Create a PieChart object of size 300 x 300 pixels
$c = new PieChart(300, 300);

if ($HTTP_GET_VARS["img"] == "0") {
#=====
#    Draw a pie chart where the label is on top of the pie
#=====

    #Set the center of the pie at (150, 150) and the radius to 120 pixels
    $c->setPieSize(150, 150, 120);

    #Set the label position to -40 pixels from the perimeter of the pie
    #(-ve means label is inside the pie)
    $c->setLabelPos(-40);

} else {
#=====
#    Draw a pie chart where the label is outside the pie
#=====

    #Set the center of the pie at (150, 150) and the radius to 80 pixels
    $c->setPieSize(150, 150, 80);

    #Set the sector label position to be 20 pixels from the pie. Use a join
    #line to connect the labels to the sectors.
    $c->setLabelPos(20, LineColor);

}

#Set the label format to three lines, showing the sector name, value, and
#percentage. The value 999 will be formatted as US$999K.
$c->setLabelFormat("{label}\nUS\${value}K\n({percent}%)");

#Set the pie data and the pie labels
$c->setData($data, $labels);

#Explode the 1st sector
$sectorObj = $c->sector(0);
$sectorObj->setExplode();

#output the chart in PNG format
```

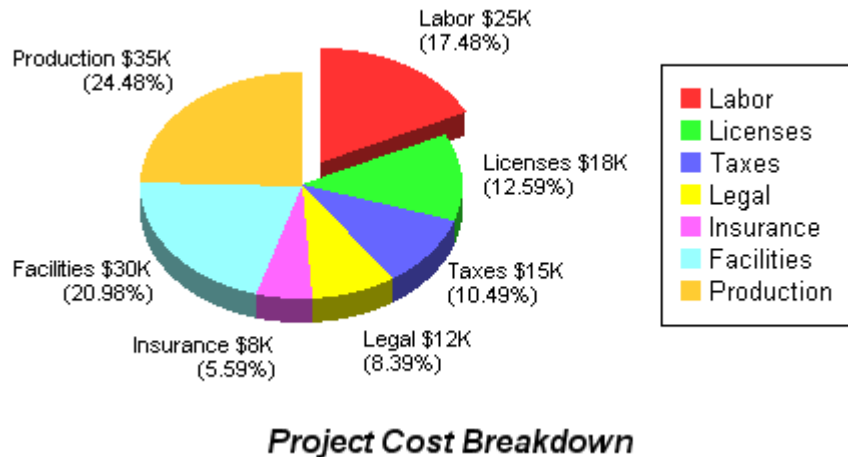


```
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## Pie Chart with Legend

This example demonstrates the following ChartDirector features:

- Add a legend box using the [addLegend](#) method
- Change the label format of the sectors using the [setLabelFormat](#) method
- Add a title to the bottom of the chart using the [addTitle2](#) method



(The following program is available as “phpdemo/legendpie.php”).

```
<?php
include("phpchartdir.php");

#The data for the pie chart
$data = array(25, 18, 15, 12, 8, 30, 35);

#The labels for the pie chart
$labels = array("Labor", "Licenses", "Taxes", "Legal", "Insurance",
    "Facilities", "Production");

#Create a PieChart object of size 450 x 240 pixels
$c = new PieChart(450, 240);

#Set the center of the pie at (150, 100) and the radius to 80 pixels
$c->setPieSize(150, 100, 80);

#Add a title at the bottom of the chart using Arial Bold Italic font
$c->addTitle2(Bottom, "Project Cost Breakdown", "arialbi.ttf");

#Draw the pie in 3D
$c->set3D();

#add a legend box where the top left corner is at (330, 40)
$c->addLegend(330, 40);

#modify the label format for the sectors to $nnnK (pp.pp%)
$c->setLabelFormat("{label} \${value}K\n({percent}%)" );

#Set the pie data and the pie labels
```

```

$c->setData($data, $labels);

#Explode the 1st sector
$sectorObj = $c->sector(0);
$sectorObj->setExplode();

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## Background Coloring



This example demonstrates the following ChartDirector features:

- Set the background color of the chart, and add a 3D border to the chart using the [setBackground](#) method of the PieChart object.
- Set the background color of the title box, using the [setBackground](#) method of the TextBox object that represents the title box. (This TextBox object is returned by the [addTitle](#) method.)
- Set the background color and add a 3D border to the sector labels, using the [setBackground](#) method of the TextBox object that represents the sector label prototype. (This TextBox object is returned by the [setLabelStyle](#) method.) The background color is set to the special constant [SameAsMainColor](#), which means it is the same as the fill color of the corresponding sector.
- Set the color of the join line and the sector border using the [setLineColor](#) method. The sector border color is set to the special constant [SameAsMainColor](#), which means it is the same as the fill color of the corresponding sector. As a result, the sectors appear to be borderless.

Note that in the above chart, there are a number of small sectors. ChartDirector automatically shifts the sector labels up and down to avoid overlapping.

This example demonstrates a technique that can improve sector label layout by adjusting the start angle of the first sector.

One common issue in pie charts is if they contain a lot of small sectors, and the data are sorted, the small sectors will be crowded together. Although the side layout method can avoid label overlapping by shifting the labels up and down, if there are too many small sectors, some labels may need to be shifted great distances.

In these cases, label layout can often be improved (that is, distributed more evenly, and shifted less) by choosing an appropriate “start angle”.

The “start angle” is the angle of the first sector in the pie. Subsequent sectors are layout in clockwise direction by default. You can control the start angle and the layout direction by using the [setStartAngle](#) method.

Label layout can be improved if the small sectors are near the horizontal axis. It is because the vertical distances between the sector labels are maximized for sectors that are near the horizontal axis. As a result, the extent of label overlapping is minimized.

If the data is in descending order (small sectors at the end), you may use a start angle of 135 degrees with clockwise sector layout. If the data is in ascending order (small sectors at the beginning), you may use a start angle of 45 degrees with clockwise sector layout. This will position the small sectors near the 90 degrees (horizontal) position.

In this example, the data is sorted in descending order, so we set the start angle to 135 degrees using the [setStartAngle](#) method to minimize label shifting. Note that the small sectors are on the right hand side of the pie. If we do not use set the start angle (in this case, the default angle of 0 will be used), the small sectors will be at the top of the pie.

(The following program is available as “phpdemo/bgpie.php”).)

```
<?php
include("phpchartdir.php");

#The data for the pie chart
$data = array(35, 30, 25, 7, 6, 5, 4, 3, 2, 1);

#The labels for the pie chart
$labels = array("Labor", "Production", "Facilities", "Taxes", "Misc",
    "Legal", "Insurance", "Licenses", "Transport", "Interest");

#Create a PieChart object of size 500 x 230 pixels
$c = new PieChart(500, 230);

#Set background color to light blue (0xccccff) with a 1 pixel 3D border
$c->setBackground(0xccccff, -1, 1);

#Add a title box using 1Times Bold Italic/14 points/blue (0x9999ff) as font
$titleObj = $c->addTitle("Project Cost Breakdown", "timesbi.ttf", 14);
$titleObj->setBackground(0x9999ff);
```

```

#Set the center of the pie at (250, 120) and the radius to 100 pixels
$c->setPieSize(250, 120, 100);

#Draw the pie in 3D
$c->set3D();

#Use the side label layout method
$c->setLabelLayout(SideLayout);

#Set the label box the same color as the sector with a 1 pixel 3D border
$labelStyleObj = $c->setLabelStyle();
$labelStyleObj->setBackground(SameAsMainColor, Transparent, 1);

#Set the border color of the sector the same color as the fill color. Set
#the line color of the join line to black (0x0)
$c->setLineColor(SameAsMainColor, 0x0);

#Set the start angle to 135 degrees may improve layout when there are many
#small sectors at the end of the data array (that is, data sorted in
#descending order). It is because this makes the small sectors position
#near the horizontal axis, where the text label has the least tendency to
#overlap. For data sorted in ascending order, a start angle of 45 degrees
#can be used instead.
$c->setStartAngle(135);

#Set the pie data and the pie labels
$c->setData($data, $labels);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## Metallic Background Coloring

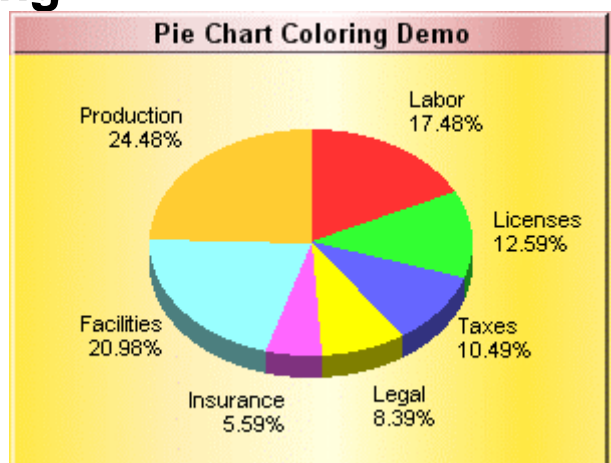
This example demonstrates the gradient coloring features of ChartDirector.

Gradient colors can be created using the [gradientColor](#) and [gradientColor2](#) methods.

ChartDirector comes with several gradient colors that have metallic look and feel. In this example, the goldGradient color is used for the chart background, while the redMetalGradient color is used for the title background.

Please refer to the description of [gradientColor2](#) for the pre-defined gradient colors that come with ChartDirector.

(The following program is available as “phpdemo/goldpie.php”).



```

<?php
include("phpchartdir.php");

#The data for the pie chart
$data = array(25, 18, 15, 12, 8, 30, 35);

#The labels for the pie chart
$labels = array("Labor", "Licenses", "Taxes", "Legal", "Insurance",
    "Facilities", "Production");

#Create a PieChart object of size 300 x 230 pixels
$c = new PieChart(300, 230);

#Set the background color of the chart to gold (goldGradient). Use a 2
#pixel 3D border.
$c->setBackground($c->gradientColor($goldGradient), -1, 2);

#Set the center of the pie at (150, 115) and the radius to 80 pixels
$c->setPieSize(150, 115, 80);

#Add a title box using 10 point Arial Bold font. Set the background color
#to red metallic (redMetalGradient). Use a 1 pixel 3D border.
$titleObj = $c->addTitle("Pie Chart Coloring Demo", "arialbd.ttf", 10);
$titleObj->setBackground($c->gradientColor($redMetalGradient), -1, 1);

#Draw the pie in 3D
$c->set3D();

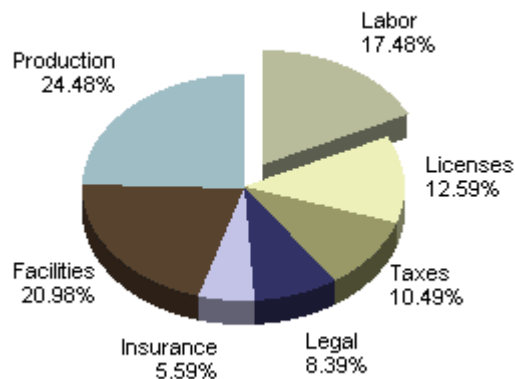
#Set the pie data and the pie labels
$c->setData($data, $labels);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

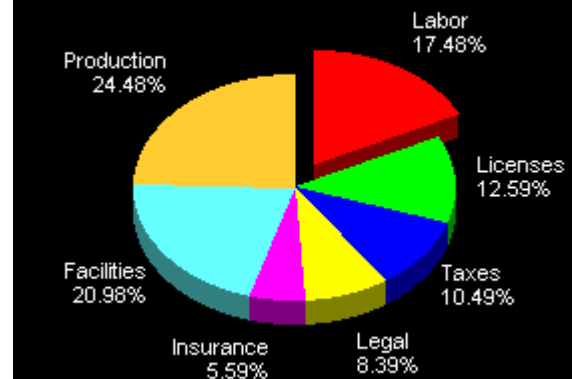
```

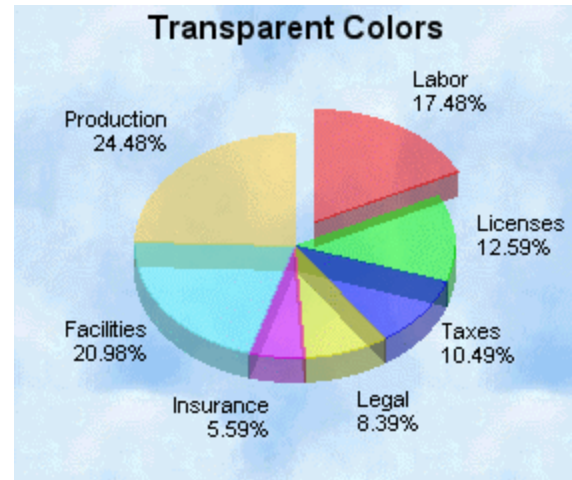
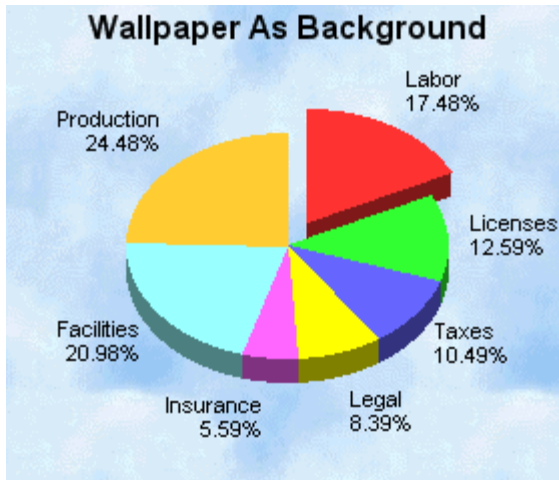
## Coloring and Wallpaper

Custom Colors



Dark Background Colors





This example demonstrates how to modify the coloring scheme using the [setColor](#) and [setColors](#) and [setColors2](#) methods, and apply a background image to the chart using the [setWallpaper](#) method.

(The following program is available as “phpdemo/colorpie.php”).)

```
<?php
include("phpchartdir.php");

#The data for the pie chart
$data = array(25, 18, 15, 12, 8, 30, 35);

#The labels for the pie chart
$labels = array("Labor", "Licenses", "Taxes", "Legal", "Insurance",
    "Facilities", "Production");

#Colors of the sectors if custom coloring is used
$colors = array(0xb8bc9c, 0xecf0b9, 0x999966, 0x333366, 0xc3c3e6,
    0x594330, 0xa0bdc4);

#Create a PieChart object of size 280 x 240 pixels
$c = new PieChart(280, 240);

#Set the center of the pie at (140, 120) and the radius to 80 pixels
$c->setPieSize(140, 120, 80);

#Draw the pie in 3D
$c->set3D();

#Set the coloring schema
if ($HTTP_GET_VARS["img"] == "0") {
    $c->addTitle("Custom Colors");
    #set the LineColor to light gray
    $c->setColor(LineColor, 0xc0c0c0);
    #use given color array as the data colors (sector colors)
    $c->setColors2(DataColor, $colors);
} else if ($HTTP_GET_VARS["img"] == "1") {
    $c->addTitle("Dark Background Colors");
    #use the standard white on black palette
    $c->setColors($whiteOnBlackPalette);
```

```

} else if ($HTTP_GET_VARS["img"] == "2") {
    $c->addTitle("Wallpaper As Background");
    $c->setWallpaper(dirname($PATH_TRANSLATED)."/bg.png");
} else {
    $c->addTitle("Transparent Colors");
    $c->setWallpaper(dirname($PATH_TRANSLATED)."/bg.png");
    #use semi-transparent colors to allow the background to be seen
    $c->setColors($transparentPalette);
}

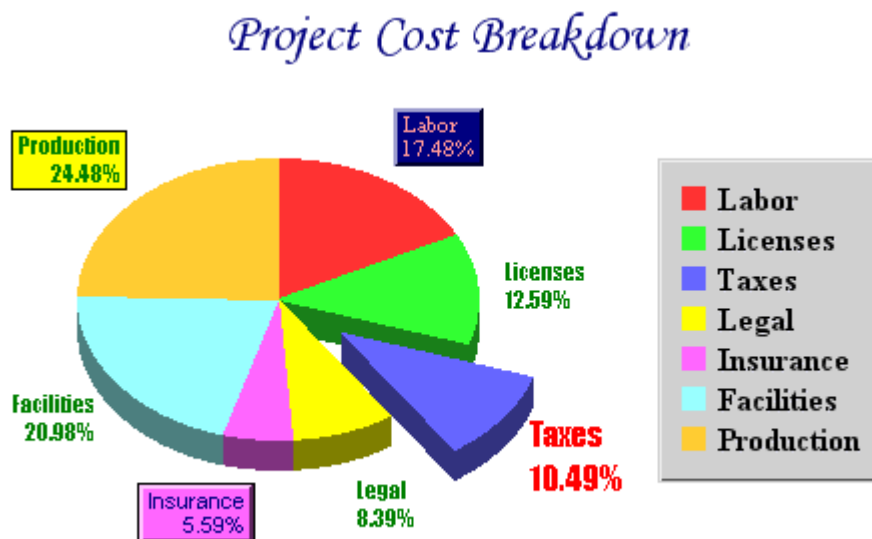
#Set the pie data and the pie labels
$c->setData($data, $labels);

#Explode the 1st sector
$sectorObj = $c->sector(0);
$sectorObj->setExplode();

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## Text Style and Colors



This example demonstrates how to control the text styles, text colors, background colors and text box border effects.

In the chart above, the title, the sector labels, and the legends all have different fonts, text colors, background colors, and border styles.

- The title text and font are specified using the [addTitle](#) method.

- The legend box font is specified using the [addLegend](#) method. The legend box background color and 3D border are specified using the [setBackground](#) method of the LegendBox object. (The LegendBox object is returned by the addLegend method.)
- The default sector label font is specified using the PieChart object's [setLabelStyle](#) method.
- The sector label font of individual sector is specified using the Sector object's [setLabelStyle](#) method.
- The sector label's background color, border color and 3D border effects are specified using the [setBackground](#) method of the TextBox object representing the sector label. (This TextBox object is returned by the Sector object's [setLabelStyle](#) method.)

(The following program is available as “phpdemo/fontpie.php”).

```
<?php
include("phpchartdir.php");

#The data for the pie chart
$data = array(25, 18, 15, 12, 8, 30, 35);

#The labels for the pie chart
$labels = array("Labor", "Licenses", "Taxes", "Legal", "Insurance",
    "Facilities", "Production");

#Create a PieChart object of size 480 x 300 pixels
$c = new PieChart(480, 300);

#Set the center of the pie at (150, 150) and the radius to 100 pixels
$c->setPieSize(150, 150, 100);

#Add a title to the pie chart using Monotype Corsiva ("mtcorsva")/20
#points/deep blue (0x000080) as font
$c->addTitle("Project Cost Breakdown", "mtcorsva.ttf", 20, 128);

#Draw the pie in 3D
$c->set3D();

#Add a legend box using 12 points Times New Romans Bold ("timesbd.ttf")
#font. Set background color to light grey (0xd0d0d0), with a 1 pixel 3D
#border.
$legendObj = $c->addLegend(340, 80, true, "timesbd.ttf", 12);
$legendObj->setBackground(0xd0d0d0, 0xd0d0d0, 1);

#Set the default font for all sector labels to Impact/8 points/dark green
#(0x008000).
$c->setLabelStyle("impact.ttf", 8, 0x8000);

#Set the pie data and the pie labels
$c->setData($data, $labels);

#Explode the 3rd sector
$sectorObj = $c->sector(2);
$sectorObj->setExplode(40);
```



```

#Use Impact/12 points/red as label font for the 3rd sector
$sectorObj = $c->sector(2);
$sectorObj->setLabelStyle("impact.ttf", 12, 0xff0000);

#Use Arial/8 points/deep blue as label font for the 5th sector. Add a
#background box using the sector fill color (SameAsMainColor), with a black
#(0x000000) edge and 2 pixel 3D border.
$sectorObj = $c->sector(4);
$labelStyleObj = $sectorObj->setLabelStyle("", 8, 0x80);
$labelStyleObj->setBackground(SameAsMainColor, 0x0, 2);

#Use Times New Romans/8 points/light red (0xff9999) as label font for the
#6th sector. Add a dark blue (0x000080) background box with a 2 pixel 3D
#border.
$sectorObj = $c->sector(0);
$labelStyleObj = $sectorObj->setLabelStyle("times.ttf", 8, 0xff9999);
$labelStyleObj->setBackground(0x80, Transparent, 2);

#Use Impact/8 points/deep green (0x008000) as label font for 7th sector.
#Add a yellow (0xFFFF00) background box with a black (0x000000) edge.
$sectorObj = $c->sector(6);
$labelStyleObj = $sectorObj->setLabelStyle("impact.ttf", 8, 0x8000);
$labelStyleObj->setBackground(0xffff00, 0x0);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

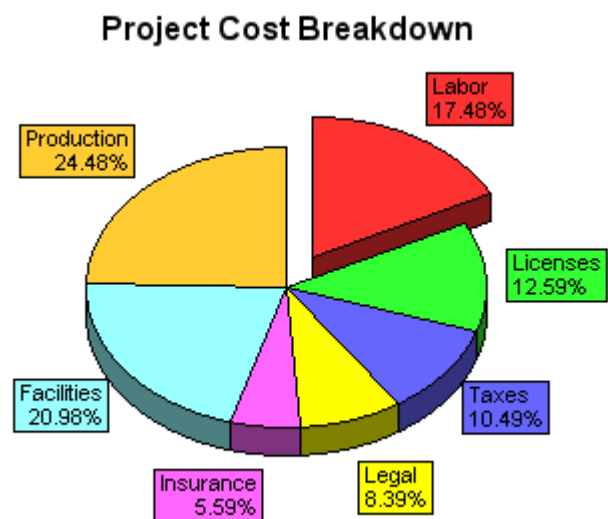
## Sectors with Borders

This example demonstrates how to add borders to the sectors, and to set the background and border colors for the sector label text boxes.

The sector borders are set using the [setLineColor](#) method.

The sector label background and border colors are set using the [setBackground](#) method of the `TextBox` object that represents the sector label prototype. (This `TextBox` object is returned by the [setLabelStyle](#) method.) The background color is set to [SameAsMainColor](#), which means the background color is the same as the corresponding sector fill color.

(The following program is available as “phpdemo/linepie.php”).



```

<?php
include("phpchartdir.php");

#The data for the pie chart
$data = array(25, 18, 15, 12, 8, 30, 35);

#The labels for the pie chart
$labels = array("Labor", "Licenses", "Taxes", "Legal", "Insurance",
    "Facilities", "Production");

#Create a PieChart object of size 360 x 280 pixels
$c = new PieChart(360, 280);

#Set the center of the pie at (180, 140) and the radius to 100 pixels
$c->setPieSize(180, 140, 100);

#Add a title to the pie chart
$c->addTitle("Project Cost Breakdown");

#Draw the pie in 3D
$c->set3D();

#Set the border color of the sectors to black (0x0)
$c->setLineColor(0x0);

#Set the background color of the sector label to the same color as the
#sector. Use a black border.
$labelStyleObj = $c->setLabelStyle();
$labelStyleObj->setBackground(SameAsMainColor, 0x0);

#Set the pie data and the pie labels
$c->setData($data, $labels);

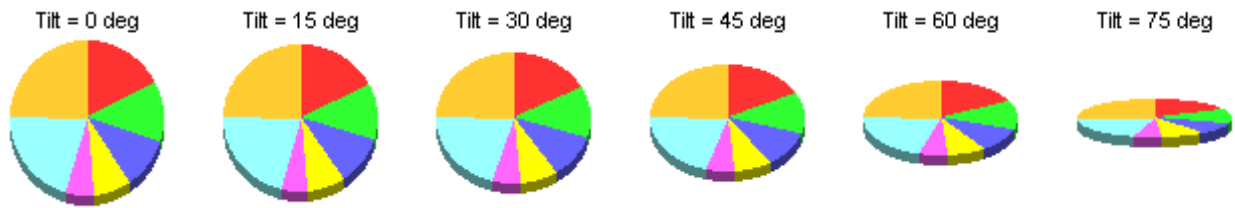
#Explode the 1st sector
$sectorObj = $c->sector(0);
$sectorObj->setExplode();

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## 3D Angles

This example illustrates the effects of different 3D angles. ChartDirector allows you to configure the 3D depth and angles using the [set3D](#) method. This example also demonstrates how to disable sector labels by setting their colors to Transparent using the [setLabelStyle](#) method.



(The following program is available as “phpdemo/3danglepie.php”).

```
<?php
include("phpchartdir.php");

#the tilt angle of the pie
$angle = $_HTTP_GET_VARS["img"] * 15;

#The data for the pie chart
$data = array(25, 18, 15, 12, 8, 30, 35);

#Create a PieChart object of size 100 x 110 pixels
$c = new PieChart(100, 110);

#Set the center of the pie at (50, 55) and the radius to 38 pixels
$c->setPieSize(50, 55, 38);

#Set the depth and tilt angle of the 3D pie (-1 means auto depth)
$c->set3D(-1, $angle);

#Add a title showing the tilt angle
$c->addTitle("Tilt = ".$angle." deg", "arial.ttf", 8);

#Set the pie data
$c->setData($data);

#Disable the sector labels by setting the color to Transparent
$c->setLabelStyle("", 8, Transparent);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## 3D Depth

This example illustrates the effects of different 3D depth. ChartDirector allows you to configure the 3D depth and angles using the [set3D](#) method.



(The following program is available as “phpdemo/3ddepthpie.php”).

```
<?php
include("phpchartdir.php");

#the tilt angle of the pie
$depth = $HTTP_GET_VARS["img"] * 5 + 5;

#The data for the pie chart
$data = array(25, 18, 15, 12, 8, 30, 35);

#Create a PieChart object of size 100 x 110 pixels
$c = new PieChart(100, 110);

#Set the center of the pie at (50, 55) and the radius to 38 pixels
$c->setPieSize(50, 55, 38);

#Set the depth of the 3D pie
$c->set3D($depth);

#Add a title showing the depth
$c->addTitle("Depth = ".$depth." pixels", "arial.ttf", 8);

#Set the pie data
$c->setData($data);

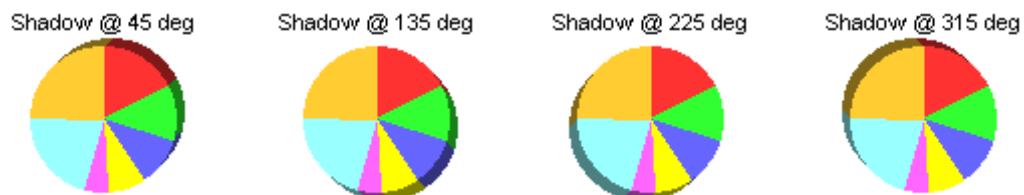
#Disable the sector labels by setting the color to Transparent
$c->setLabelStyle("", 8, Transparent);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## 3D Shadow Mode

The standard way to draw a pie chart in 3D is to view the chart from an inclined angle. Using this method, the surface of a 3D pie will become an ellipse.

The ChartDirector supports an alternative way to draw a pie chart in 3D, that is, to draw the 3D portion like a shadow. Using this method, the 3D pie will remain perfectly circular. See below for illustration.



Comparing the two 3D styles, the pie in the shadow 3D mode is bigger, remains perfectly circular and is not “distorted”. Some people think it presents information more accurately. However, some people think the standard 3D mode is more “natural” looking.

Which 3D mode is better is a matter of personal preference. The ChartDirector supports both so the developer can choose which mode suits his needs. The [set3D](#) method can be use to select which mode to use.

(The following program is available as “phpdemo/shadowpie.php”).

```
<?php
include("phpchartdir.php");

#the tilt angle of the pie
$angle = $HTTP_GET_VARS["img"] * 90 + 45;

#The data for the pie chart
$data = array(25, 18, 15, 12, 8, 30, 35);

#Create a PieChart object of size 100 x 110 pixels
$c = new PieChart(100, 110);

#Set the center of the pie at (50, 55) and the radius to 36 pixels
$c->setPieSize(50, 55, 36);

#Set the depth, tilt angle and 3D mode of the 3D pie (-1 means auto depth,
#"true" means the 3D effect is in shadow mode)
$c->set3D(-1, $angle, true);

#Add a title showing the shadow angle
$c->addTitle("Shadow @ ".$angle." deg", "arial.ttf", 8);

#Set the pie data
$c->setData($data);

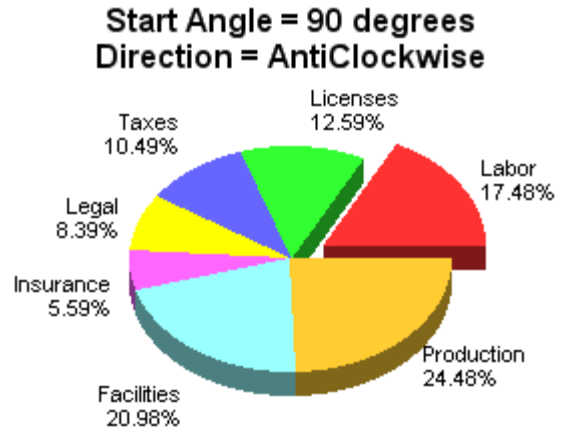
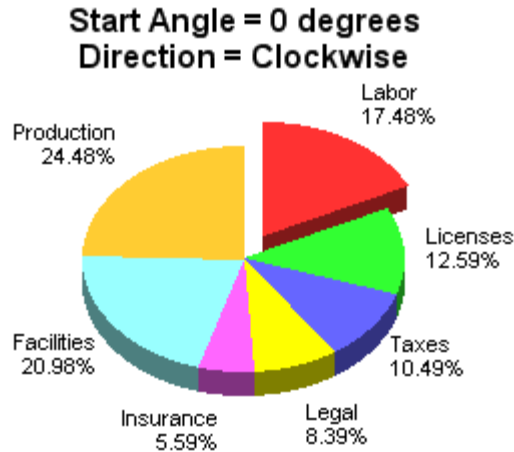
#Disable the sector labels by setting the color to Transparent
$c->setLabelStyle("", 8, Transparent);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## Layout Angle and Direction

By default, ChartDirector will layout the sectors in a pie chart starting from the positive y-axis and in the clockwise direction.

Both the start angle and the layout direction can be changed using the [setStartAngle](#) method.



One common application for the `setStartAngle` method is to ensure the small sectors lie near the horizontal axis for data series that are sorted. Please refer to the earlier example on “Background Coloring”.

(The following program is available as “`phpdemo/anglepie.php`”.)

```
<?php
include("phpchartdir.php");

#query string to determine the starting angle and direction
$angle = 0;
$clockwise = true;
if ($HTTP_GET_VARS["img"] != "0") {
    $angle = 90;
    $clockwise = false;
}

#The data for the pie chart
$data = array(25, 18, 15, 12, 8, 30, 35);

#The labels for the pie chart
$labels = array("Labor", "Licenses", "Taxes", "Legal", "Insurance",
    "Facilities", "Production");

#Create a PieChart object of size 280 x 240 pixels
$c = new PieChart(280, 240);

#Set the center of the pie at (140, 130) and the radius to 80 pixels
$c->setPieSize(140, 130, 80);

#Add a title to the pie to show the start angle and direction
if ($clockwise) {
    $c->addTitle("Start Angle = ".$angle." degrees\nDirection = Clockwise")
;
} else {
    $c->addTitle("Start Angle = ".$angle.
        " degrees\nDirection = AntiClockwise");
}
```

```
#Set the pie start angle and direction
$c->setStartAngle($angle, $clockwise);

#Draw the pie in 3D
$c->set3D();

#Set the pie data and the pie labels
$c->setData($data, $labels);

#Explode the 1st sector
$sectorObj = $c->sector(0);
$sectorObj->setExplode();

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

# XY Chart

In ChartDirector, XYChart refers to any chart that has x and y axes. These include bar charts, line charts, scatter charts, trend line charts, area charts, high-low-open-close charts and candlestick charts.

ChartDirector employs a layering architecture for XY charts. Each chart type is represented as a layer on the “plot area”. You may include multiple layers on the same plot area to create combination charts.

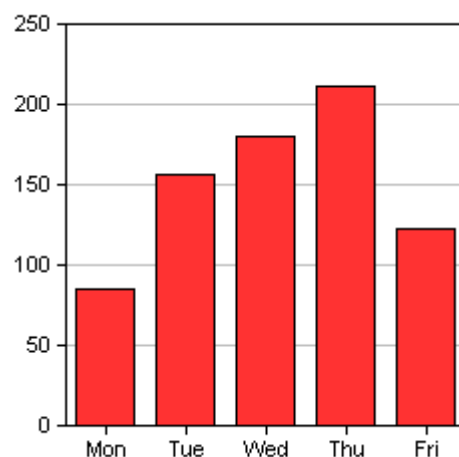
We will discuss how to create various XY charts using the ChartDirector API through a number of examples. Details of the ChartDirector API can be found in the chapter [ChartDirector API Reference](#).

## Simple Bar Chart

The simple bar chart has already been discussed in the chapter on Getting Started. For completeness of this chapter, we will repeat the project here.

This project demonstrates the following basic steps in creating a bar chart:

- Create a XYChart object using “new [XYChart](#)”.
- Specify the plot area of the chart using the [setPlotArea](#) method. The plotarea is the rectangle bounded by the x axis and y axis. You should leave some margin on the sides for axis labels and titles, etc.
- Specify the label on the x axis using the [setLabels](#) method of the x axis object.
- Add a bar chart layer and specify the data for the bar using the [addBarLayer](#) method.
- Generate the chart using the [makeChart2](#) method.



(The following program is available as “phpdemo/simplebar.php”).

```
<?php
include("phpchartdir.php");

#The data for the bar chart
$data = array(85, 156, 179.5, 211, 123);
```



```

#The labels for the bar chart
$labels = array("Mon", "Tue", "Wed", "Thu", "Fri");

#Create a XYChart object of size 250 x 250 pixels
$c = new XYChart(250, 250);

#Set the plotarea at (30, 20) and of size 200 x 200 pixels
$c->setPlotArea(30, 20, 200, 200);

#Add a bar chart layer using the given data
$c->addBarLayer($data);

#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

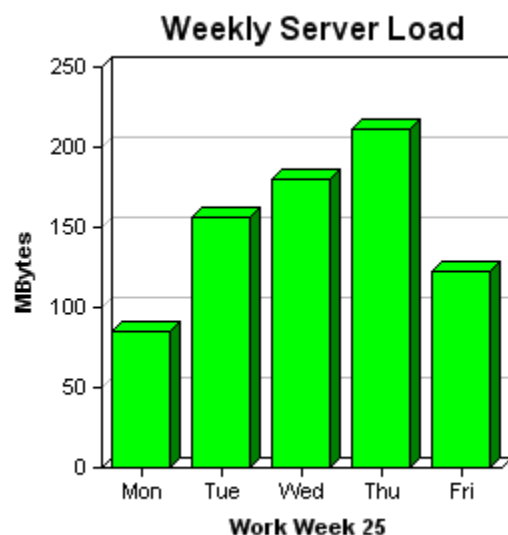
#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
}

```

## 3D Bar Chart

This example extends the previous simple bar example by introducing the following features of the ChartDirector:

- Draw the bars in 3D using the [set3D](#) method
- Add a title to the chart using the [addTitle](#) method
- Add a title to the x axis using the [setTitle](#) method of the x axis object
- Add a title to the y axis using the [setTitle](#) method of the y axis object



(The following program is available as “phpdemo/threedbar.php”).

```

<?php
include("phpchartdir.php");

#The data for the bar chart
$data = array(85, 156, 179.5, 211, 123);

#The labels for the bar chart
$labels = array("Mon", "Tue", "Wed", "Thu", "Fri");

#Create a XYChart object of size 300 x 280 pixels
$c = new XYChart(300, 280);

```

```
#Set the plotarea at (45, 30) and of size 200 x 200 pixels
$c->setPlotArea(45, 30, 200, 200);

#Add a title to the chart
$c->addTitle("Weekly Server Load");

#Add a title to the y axis
$c->yAxis->setTitle("MBytes");

#Add a title to the x axis
$c->xAxis->setTitle("Work Week 25");

#Add a bar chart layer with green (0x00ff00) bars using the given data
$barLayerObj = $c->addBarLayer($data, 0xff00);
$barLayerObj->set3D();

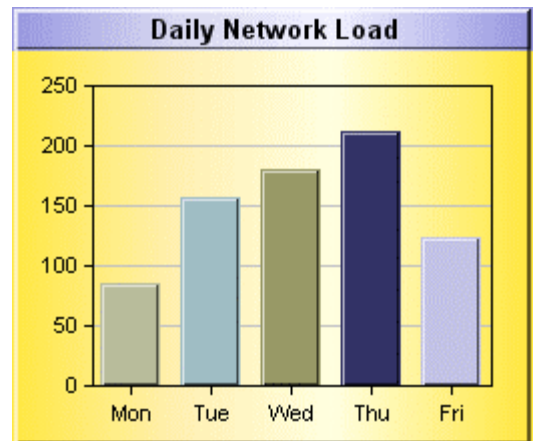
#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## Multi-Color Bar Chart

A multi-color bar chart is a bar chart where each bar has a different color. This example creates a multi-color bar chart and demonstrates the following ChartDirector features:

- Set the background color of the chart with a 3D border using the [setBackground](#) method of the XYChart object
- Set the background color of the title text box with a 3D border using the [setBackground](#) method of the TextBox object
- Create a multi-color bar layer using the [addBarLayer3](#) method
- Set a 3D border effect for the bars using the [setBorderColor](#) method of the Layer object.
- Create “gradient colors” using the [gradientColor2](#) method to create colors with golden and shiny look and feel. For more information on gradient colors, please refer to the section on [Color Specification](#).



(The following program is available as “phpdemo/colorbar.php”).)

```

<?php
include("phpchartdir.php");

#The data for the bar chart
$data = array(85, 156, 179.5, 211, 123);

#The labels for the bar chart
$labels = array("Mon", "Tue", "Wed", "Thu", "Fri");

#The colors for the bar chart
$colors = array(0xb8bc9c, 0xa0bdc4, 0x999966, 0x333366, 0xc3c3e6);

#Create a XYChart object of size 260 x 220 pixels
$c = new XYChart(260, 220);

#Set the background color of the chart to gold (goldGradient). Use a 2
#pixel 3D border.
$c->setBackground($c->gradientColor($goldGradient), -1, 2);

#Add a title box using 10 point Arial Bold font. Set the background color
#to blue metallic (blueMetalGradient). Use a 1 pixel 3D border.
$titleObj = $c->addTitle("Daily Network Load", "arialbd.ttf", 10);
$titleObj->setBackground($c->gradientColor($blueMetalGradient), -1, 1);

#Set the plotarea at (40, 40) and of 200 x 150 pixels in size
$c->setPlotArea(40, 40, 200, 150);

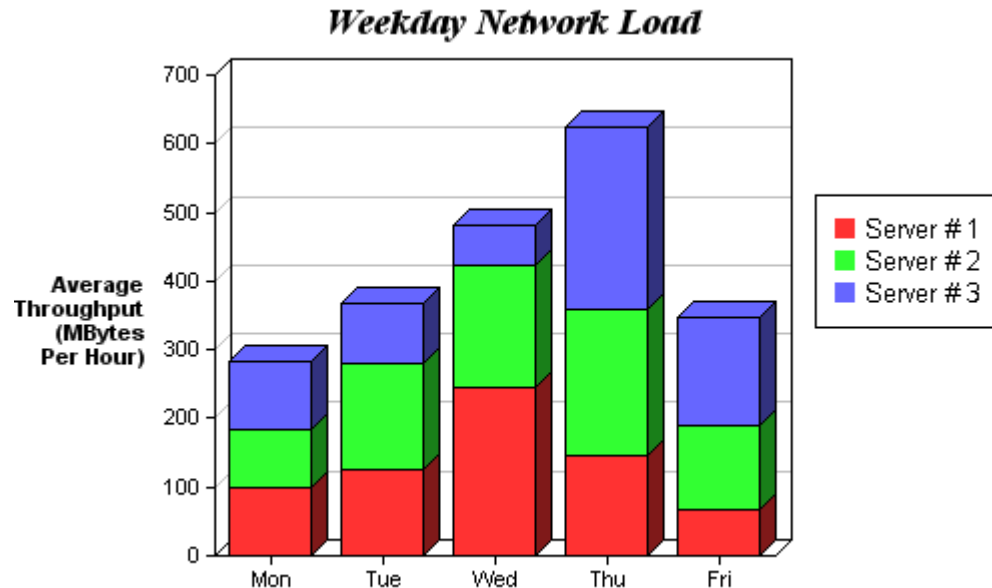
#Add a multi-color bar chart layer using the given data and colors. Use a 1
#pixel 3D border for the bars.
$barLayer3Obj = $c->addBarLayer3($data, $colors);
$barLayer3Obj->setBorderColor(-1, 1);

#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

# Stacked Bar Chart



This example illustrates the stacked bar chart, where multiple data sets are represented on the same bar chart layer by stacking the bar segments up. It demonstrates the following features of ChartDirector:

- Add a stacked bar layer using the [addBarLayer](#) method
- Add multiple data sets to the bar layer using the [addDataSet](#) method
- Add a legend to the chart using the [addLegend](#) method
- Add a title to the y-axis using the [addTitle](#) method, and draw the title upright using the [setFontAngle](#) method (the default for y-axis is to draw the title sideways – see previous examples). Note that the y-axis title can contain multiple lines. This is by including the line break character in the title.

(The following program is available as “phpdemo/stackedbar.php”).

```
<?php
include("phpchartdir.php");

#The data for the bar chart
$data0 = array(100, 125, 245, 147, 67);
$data1 = array(85, 156, 179, 211, 123);
$data2 = array(97, 87, 56, 267, 157);

#The labels for the bar chart
$labels = array("Mon", "Tue", "Wed", "Thu", "Fri");

#Create a XYChart object of size 500 x 320 pixels
$c = new XYChart(500, 320);

#Set the plotarea at (100, 40) and of size 280 x 240 pixels
```

```

$c->setPlotArea(100, 40, 280, 240);

#Add a legend box at (400, 100)
$c->addLegend(400, 100);

#Add a title to the chart using 14 points Times Bold Italic font
$c->addTitle("Weekday Network Load", "timesbi.ttf", 14);

#Add a title to the y axis. Draw the title upright (font angle = 0)
$titleObj = $c->yAxis->setTitle("Average\nThroughput\n(MBytes\nPer Hour)");
$titleObj->setFontAngle(0);

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Add a stacked bar layer and set the layer 3D depth to 8 pixels
$layer = $c->addBarLayer2(Stack, 8);

#Add the three data sets to the bar layer
$layer->addDataSet($data0, -1, "Server # 1");
$layer->addDataSet($data1, -1, "Server # 2");
$layer->addDataSet($data2, -1, "Server # 3");

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

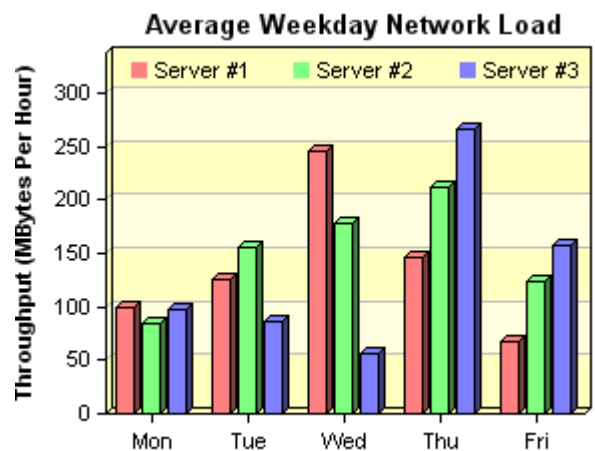
```

## Multi-Bar Chart

This example illustrates the multi-bar chart, where multiple data sets are represented on the same bar chart layer by drawing the bars side by side.

In addition, this example demonstrates the following ChartDirector features:

- Use two alternative colors as the plot area background by using the [setBackground](#) method of the PlotArea object.
- The y-axis is configured to leave a margin at the top for the legend keys. This is by using the [setTopMargin](#) method of the YAxis object.



(The following program is available as “phpdemo/multibar.php”).)

```

<?php
include("phpchartdir.php");

#The data for the bar chart
$data0 = array(100, 125, 245, 147, 67);

```

```

$data1 = array(85, 156, 179, 211, 123);
$data2 = array(97, 87, 56, 267, 157);
$labels = array("Mon", "Tue", "Wed", "Thu", "Fri");

#Create a XYChart object of size 300 x 240 pixels
$c = new XYChart(300, 240);

#Add a title to the chart using 10 pt Arial font
$c->addTitle("          Average Weekday Network Load", "", 10);

#Set the plot area at (45, 25) and of size 240 x 180. Use two alternative
#background colors (0xffffc0 and 0xffffe0)
$plotAreaObj = $c->setPlotArea(45, 25, 240, 180);
$plotAreaObj->setBackground(0xffffc0, 0xffffe0);

#Add a legend box at (50, 20) using horizontal layout. Use 8 pt Arial font,
#with transparent background
$legendObj = $c->addLegend(50, 20, false, "", 8);
$legendObj->setBackground(Transparent);

#Add a title to the y-axis
$c->yAxis->setTitle("Throughput (MBytes Per Hour)");

#Reserve 20 pixels at the top of the y-axis for the legend box
$c->yAxis->setTopMargin(20);

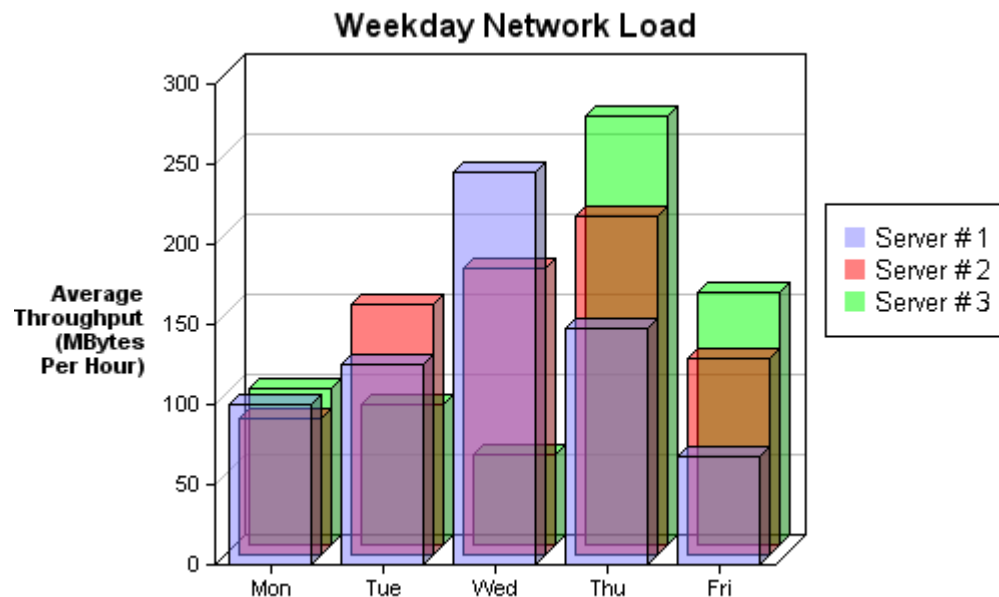
#Set the x axis labels
$c->xAxis->setLabels($labels);

#Add a multi-bar layer with 3 data sets
$layer = $c->addBarLayer2(Side, 3);
$layer->addDataSet($data0, 0xff8080, "Server #1");
$layer->addDataSet($data1, 0x80ff80, "Server #2");
$layer->addDataSet($data2, 0x8080ff, "Server #3");

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

# Depth Bar Chart



This example illustrates how you could display multiple data sets by using layers. The ChartDirector allows you to any arbitrary XY chart types for each layer. In this particular example, all layers are of bar chart type.

Note also the transparency feature of the ChartDirector. The bars are drawn in semi-transparent colors so that you can see through the bars.

(The following program is available as “phpdemo/depthbar.php”).

```
<?php
include("phpchartdir.php");

#The data for the bar chart
$data0 = array(100, 125, 245, 147, 67);
$data1 = array(85, 156, 179, 211, 123);
$data2 = array(97, 87, 56, 267, 157);

#The labels for the bar chart
$labels = array("Mon", "Tue", "Wed", "Thu", "Fri");

#Create a XYChart object of size 500 x 320 pixels
$c = new XYChart(500, 320);

#Set the plotarea at (100, 40) and of size 280 x 240 pixels
$c->setPlotArea(100, 40, 280, 240);

#Add a legend box at (405, 100)
$c->addLegend(405, 100);

#Add a title to the chart
$c->addTitle("Weekday Network Load");
```

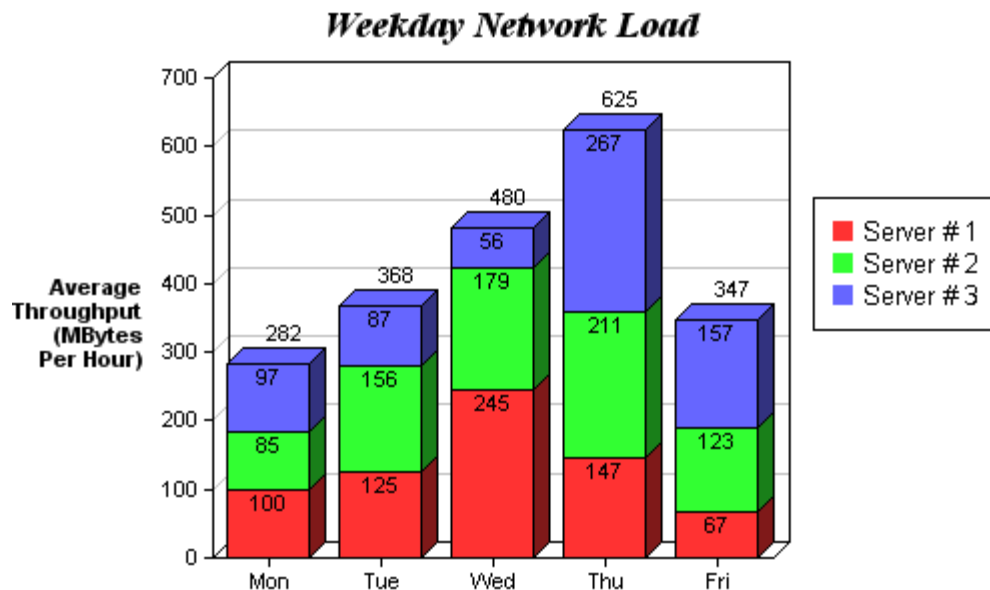
```
#Add a title to the y axis. Draw the title upright (font angle = 0)
$titleObj = $c->yAxis->setTitle("Average\nThroughput\n(MBytes\nPer Hour)");
$titleObj->setFontAngle(0);

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Add three bar layers, each representing one data set. The bars are drawn
#in semi-transparent colors.
$c->addBarLayer($data0, 0x808080ff, "Server # 1", 5);
$c->addBarLayer($data1, 0x80ff0000, "Server # 2", 5);
$c->addBarLayer($data2, 0x8000ff00, "Server # 3", 5);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## Bar Labels



This example illustrates how you could add data labels to the complete bars using the [setAggregateLabelStyle](#) method, as well as add labels to bar segments using the [setDataLabelStyle](#) method.

In addition to enabling and disabling the bar labels, you can control the style of the labels, their orientation (upright or sideways), their positions and data formats (e.g. number of decimal points) using the above ChartDirector API and two additional methods [setAggregateLabelFormat](#) and [setDataLabelFormat](#). Please refer to the [ChartDirector API Reference](#) for details.



When using bar labels with auto-scaling, remember to use the [setAutoScale](#) or [setTopMargin](#) method of the YAxis object to reserve some margin at the top of the plot area. This ensures bar labels are always drawn within the plot area.

(The following program is available as “phpdemo/labelbar.php”).

```
<?php
include("phpchartdir.php");

#The data for the bar chart
$data0 = array(100, 125, 245, 147, 67);
$data1 = array(85, 156, 179, 211, 123);
$data2 = array(97, 87, 56, 267, 157);

#The labels for the bar chart
$labels = array("Mon", "Tue", "Wed", "Thu", "Fri");

#Create a XYChart object of size 500 x 320 pixels
$c = new XYChart(500, 320);

#Set the plotarea at (100, 40) and of size 280 x 240 pixels
$c->setPlotArea(100, 40, 280, 240);

#Add a legend box at (400, 100)
$c->addLegend(400, 100);

#Add a title to the chart using 14 points Times Bold Italic font
$c->addTitle("Weekday Network Load", "timesbi.ttf", 14);

#Add a title to the y axis. Draw the title upright (font angle = 0)
$titleObj = $c->yAxis->setTitle("Average\nThroughput\n(MBytes\nPer Hour)");
$titleObj->setFontAngle(0);

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Add a stacked bar layer and set the layer 3D depth to 8 pixels
$layer = $c->addBarLayer2(Stack, 8);

#Add the three data sets to the bar layer
$layer->addDataSet($data0, -1, "Server # 1");
$layer->addDataSet($data1, -1, "Server # 2");
$layer->addDataSet($data2, -1, "Server # 3");

#Enable bar label for the whole bar
$layer->setAggregateLabelStyle();

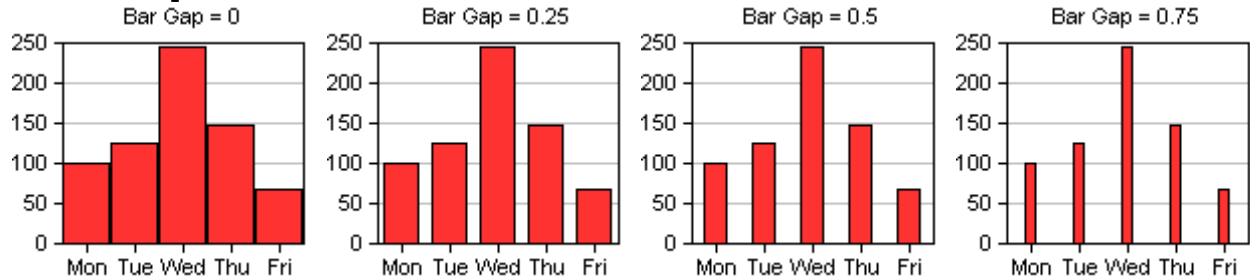
#Enable bar label for each segment of the stacked bar
$layer->setDataLabelStyle();

#Reserve 10% margin at the top of the plot area during auto-scaling. This
#ensures the data labels will not fall outside the plot area.
$c->yAxis->setAutoScale(0.1);

#output the chart in PNG format
header("Content-type: image/png");
```

```
print($c->makeChart2(PNG));
?>
```

## Bar Gap



This example illustrates the effects of manipulating the bar gap using the [setBarGap](#) method.

(The following program is available as “phpdemo/gapbar.php”).

```
<?php
include("phpchartdir.php");

$bargap = $_HTTP_GET_VARS["img"] * 0.25;

#The data for the bar chart
$data = array(100, 125, 245, 147, 67);

#The labels for the bar chart
$labels = array("Mon", "Tue", "Wed", "Thu", "Fri");

#Create a XYChart object of size 150 x 150 pixels
$c = new XYChart(150, 150);

#Set the plotarea at (25, 20) and of size 120 x 100 pixels
$c->setPlotArea(25, 20, 120, 100);

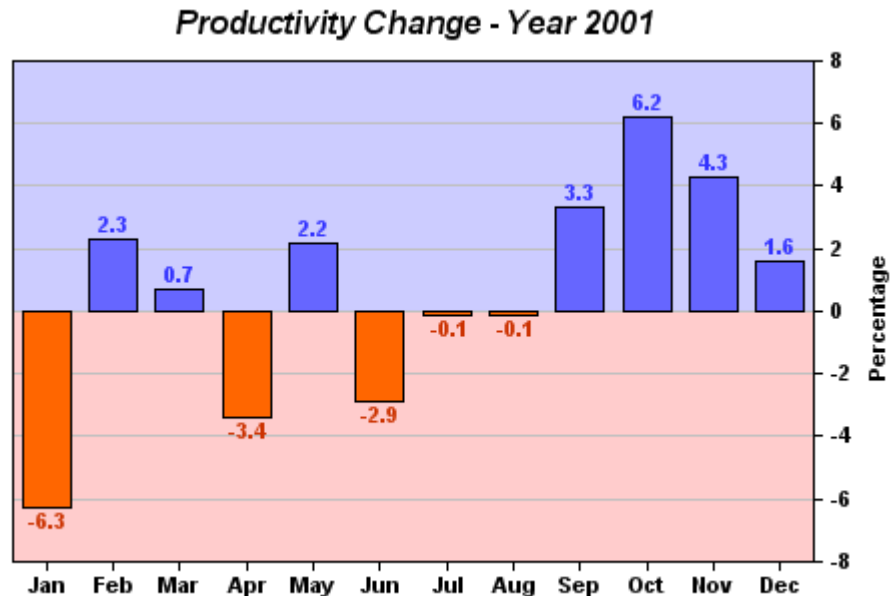
#Add a title to display to bar gap using 8 pts Arial font
$c->addTitle("          Bar Gap = ".$bargap, "arial.ttf", 8);

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Add a bar chart layer using the given data and set the bar gap
$barLayerObj = $c->addBarLayer($data);
$barLayerObj->setBarGap($bargap);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

# Positive Negative Bars



This example illustrates combining various ChartDirector features for designing nice looking charts.

The above bar chart actually consists of two bar chart layers. A simple loop is used to split the original data array into two arrays – one for positive data, and one for negative data. These two arrays then go to two bar chart layers of different colors. In this way, the positive and negative bars, and their data labels, can be of different colors.

The plot area background is colored using two custom rectangles. The positive side is a light blue rectangle, while the negative side is a pink rectangle.

(The following program is available as “phpdemo/posnegbar.php”).

```
<?php
include("phpchartdir.php");

#The data for the bar chart
$data = array(-6.3, 2.3, 0.7, -3.4, 2.2, -2.9, -0.1, -0.1, 3.3, 6.2, 4.3,
  1.6);

#The labels for the bar chart
$labels = array("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
  "Sep", "Oct", "Nov", "Dec");

#Create a XYChart object of size 500 x 320 pixels
$c = new XYChart(500, 320);

#Add a title to the chart using Arial Bold Italic font
$c->addTitle("Productivity Change - Year 2001", "arialbi.ttf");

#Set the plotarea at (50, 30) and of size 400 x 250 pixels
```

```

$c->setPlotArea(50, 30, 400, 250);

#Split the original data array into two data arrays for the +ve and -ve
#data
$posData = array_pad(array(), count($data), 0);
$negData = array_pad(array(), count($data), 0);
for($i = 0; $i < count($labels); ++$i) {
    if ($data[$i] >= 0) {
        $posData[$i] = $data[$i];
        $negData[$i] = NoValue;
    } else {
        $negData[$i] = $data[$i];
        $posData[$i] = NoValue;
    }
}

#Add a blue (0x6666ff) bar chart layer using the positive data
$positiveLayer = $c->addBarLayer($posData, 0x6666ff);

#Use blue bar labels (0x3333ff) for the positive bar layer
$positiveLayer->setAggregateLabelStyle("arialbd.ttf", 8, 0x3333ff);

#Add a red (0xff6600) bar chart layer using the negative data
$negativeLayer = $c->addBarLayer($negData, 0xff6600);

#Use red bar labels (0xcc3300) for the negative bar layer
$negativeLayer->setAggregateLabelStyle("arialbd.ttf", 8, 0xcc3300);

#Set the labels on the x axis
$labelsObj = $c->xAxis->setLabels($labels);
$labelsObj->setFontStyle("arialbd.ttf");

#Reverse 10% margin on top and bottom of the plot area during auto-scaling
#to leave space for the bar labels
$c->yAxis->setAutoScale(0.1, 0.1);

#Draw the y axis on the right of the plot area
$c->setYAxisOnRight(true);

#Use Arial Bold as the y axis label font
$c->yAxis->setLabelStyle("arialbd.ttf");

#Add a title to the y axis
$c->yAxis->setTitle("Percentage");

#
#Draw background colors of the plot area. The positive side is pale blue,
#while the negative side is red. This is done by drawing two custom boxes
#at the plotarea background
#

#Need to layout first to determine the axis-scale, so that the zero point
#position can be determined.
$c->layout();

#Obtain the DrawArea object for custom drawing
$drawarea = $c->getDrawArea();

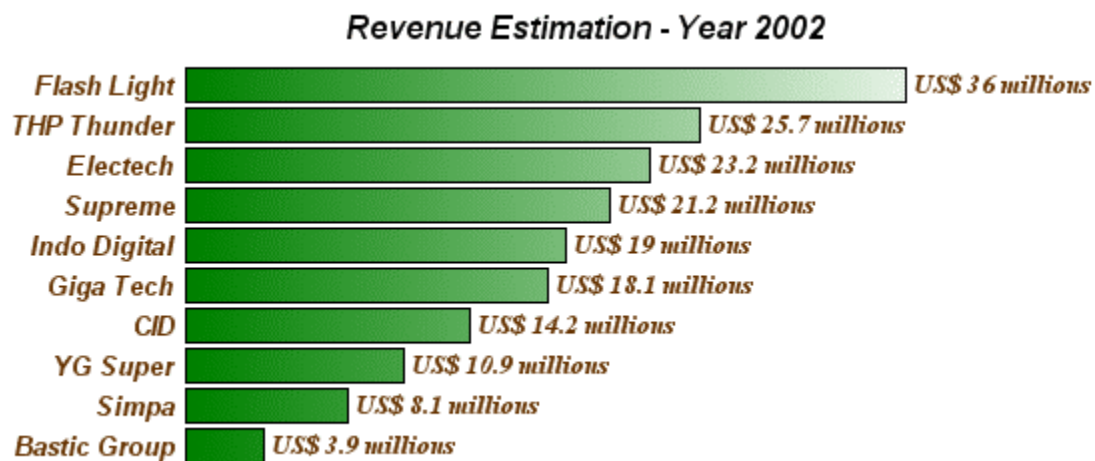
```

```
#Draw the positive rectangle background using pale blue (0xccccff)
$drawarea->rect(50, 30, 450, $positiveLayer->getYCoor(0), 0xccccff,
    0xccccff);

#Draw the negative rectangle background using pale red (0xffcccc)
$drawarea->rect(50, $negativeLayer->getYCoor(0), 450, 280, 0xffcccc,
    0xffcccc);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## Horizontal Bar Chart



This example illustrates how to draw horizontal bar charts by using the [swapXY](#) method. The swapXY method swaps the x-axis and y-axis, so the x-axis becomes vertical and the y-axis horizontal.

In the current version of ChartDirector, only bar chart layers support swapping x-axis and y-axis. After axis swapping, the bars will become horizontal.

This example also illustrates how ChartDirector features could be applied to design the charts you want.

For instance, the above chart does not have any visible axis, plot area rectangle, or grid lines. This is achieved by setting their colors Transparent.

Although the x-axis (vertical axis) is invisible, the x-axis labels remain visible. They become the bar names on the left, formatted using 10 point Arial Bold Italic font with deep red color.

The bar labels on the right shows the bar values in the format “US\$ nnn millions”. This is achieved by using the [setLabelFormat](#) method of the Layer object. The font used is 10 points Times Bold Italic with deep red color.

Notice that the color of the bars is actually a gradient color. It is dark green on the left and changes smoothly to white on the right. The gradient color is created using the [gradientColor](#) method.

(The following program is available as “phpdemo/hbar.php”).

```
<?php
include("phpchartdir.php");

#The data for the bar chart
$data = array(3.9, 8.1, 10.9, 14.2, 18.1, 19.0, 21.2, 23.2, 25.7, 36);

#The labels for the bar chart
$labels = array("Bastic Group", "Simpa", "YG Super", "CID", "Giga Tech",
    "Indo Digital", "Supreme", "Electech", "THP Thunder", "Flash Light");

#Create a XYChart object of size 600 x 250 pixels
$c = new XYChart(600, 250);

#Add a title to the chart using Arial Bold Italic font
$c->addTitle("Revenue Estimation - Year 2002", "arialbi.ttf");

#Set the plotarea at (100, 30) and of size 400 x 200 pixels. Set the
#plotarea border, background and grid lines to Transparent
$c->setPlotArea(100, 30, 400, 200, Transparent, Transparent, Transparent,
    Transparent, Transparent);

#Add a bar chart layer using the given data. Use a gradient color for the
#bars, where the gradient is from dark green (0x008000) to white (0xffffffff)
$layer = $c->addBarLayer($data, $c->gradientColor(100, 0, 500, 0, 0x8000,
    0xffffffff));

#Swap the axis so that the bars are drawn horizontally
$c->swapXY(true);

#Set the bar gap to 10%
$layer->setBarGap(0.1);

#Use the format "US$ xxx millions" as the bar label
$layer->setAggregateLabelFormat("US\$ {value} millions");

#Set the bar label font to 10 pts Times Bold Italic/dark red (0x663300)
$layer->setAggregateLabelStyle("timesbi.ttf", 10, 0x663300);

#Set the labels on the x axis
$textbox = $c->xAxis->setLabels($labels);

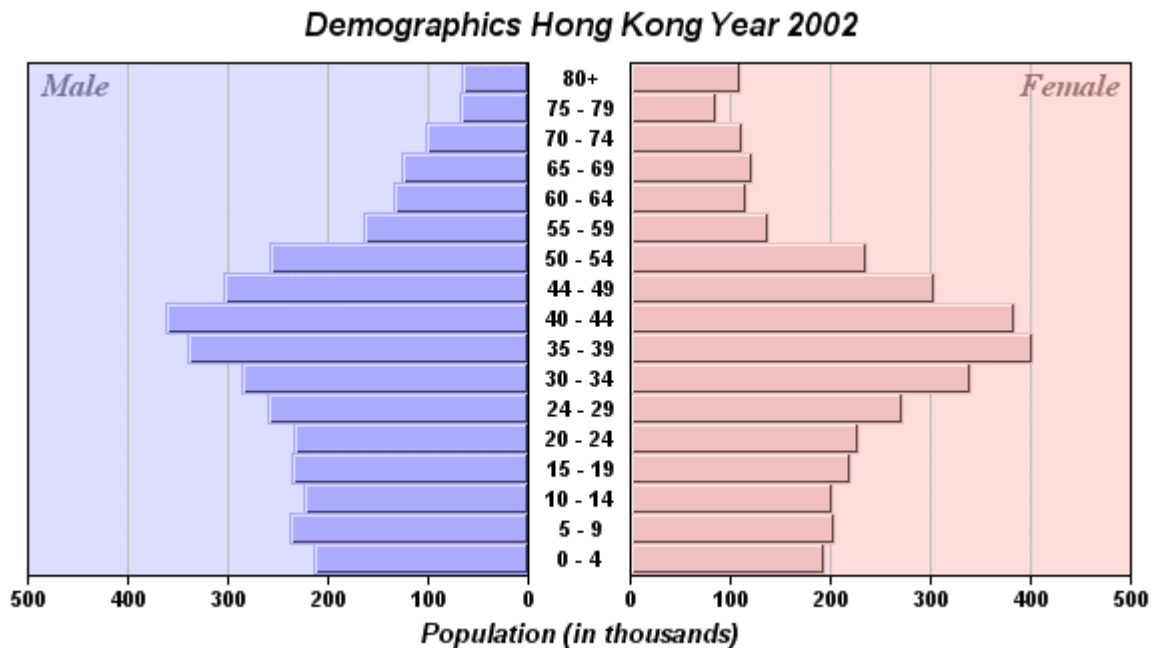
#Set the x axis label font to 10pt Arial Bold Italic
$textbox->setFontStyle("arialbi.ttf");
$textbox->setFontSize(10);

#Set the x axis to Transparent, with labels in dark red (0x663300)
$c->xAxis->setColors(Transparent, 0x663300);

#Set the y axis and labels to Transparent
$c->yAxis->setColors(Transparent, Transparent);
```

```
#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## Dual Horizontal Bar Charts



This example illustrates combining various ChartDirector features for designing more sophisticated charts.

The chart above actually consists of two separate bar charts. They are merge into a single image using the [merge](#) method of the DrawArea object.

The bars in the charts are configured with a 3D border using the [setBorderColor](#) method.

There are no gaps between adjacent bars. In fact, the 3D borders of the bars overlap (two adjacent bars shared a single 3D border). This is achieved by setting a slightly negative bar gap using the [setBarGap](#) method.

(The following program is available as “phpdemo/demographics.php”.)

```
<?php
include("phpchartdir.php");

#This demo chart shows the male and female population by age groups. It
#actually consists of two bar chart objects. They are merge into one chart
#using the ChartDirector API.

#The age groups
$labels = array("0 - 4", "5 - 9", "10 - 14", "15 - 19", "20 - 24",
```

```

    "24 - 29", "30 - 34", "35 - 39", "40 - 44", "44 - 49", "50 - 54",
    "55 - 59", "60 - 64", "65 - 69", "70 - 74", "75 - 79", "80+");

#The male population (in thousands)
$male = array(215, 238, 225, 236, 235, 260, 286, 340, 363, 305, 259, 164,
    135, 127, 102, 68, 66);

#The female population (in thousands)
$female = array(194, 203, 201, 220, 228, 271, 339, 401, 384, 304, 236,
    137, 116, 122, 112, 85, 110);

#=====
#   Draw the main chart, which contains the right bar chart
#=====

#Create a XYChart object of size 590 x 340 pixels
$c = new XYChart(590, 340);

#Add a title to the chart using Arial Bold Italic font
$c->addTitle("Demographics Hong Kong Year 2002", "arialbi.ttf");

#Set the plotarea at (320, 30) and of size 250 x 256 pixels. Use pink
#(0xffdddd) as the background. This is the plot area for the right bar
#chart. Later we will create another XYChart object and merge into the left
#side.
$c->setPlotArea(320, 30, 250, 256, 0xffdddd);

#Add a custom text label at the top right corner of the right bar chart
$textObj = $c->addText(570, 30, "Female", "timesbi.ttf", 12, 0xa07070);
$textObj->setAlignment(TopRight);

#Add the pink (0xf0c0c0) bar chart layer using the female data
$femaleLayer = $c->addBarLayer($female, 0xf0c0c0);

#Swap the axis so that the bars are drawn horizontally
$c->swapXY(true);

#Use a slightly negative bar gap so that the bars are packed tightly with
#the border of a bar overlaps with the borders its neighbours (that is, two
#bars sharing a common border)
$femaleLayer->setBarGap(-0.1);

#Set the border style of the bars to 1 pixel 3D border
$femaleLayer->setBorderColor(-1, 1);

#Add a Transparent line layer to the chart using the male data. As it is
#Transparent, only the female bar chart can be seen. We need both the male
#and female data in both the left and right charts, because we want their
#axis to have the same scale. In this example, we are using auto-scaling.
#If the data for the charts are different, the axis scale can be different.
#That's why we need to add both male and female to both charts, but make
#only one of them visible in each chart.
$c->addLineLayer($male, Transparent);

#Set the y axis label font to Arial Bold
$c->yAxis->setLabelStyle("arialbd.ttf");

```



```

#=====
#   Draw the left bar chart
#=====

#We change the data to negative, because the bars on the left bar chart
#start at the right side and extend to the left
for($i = 0; $i < count($labels); ++$i) {
    $male[$i] = -$male[$i];
    $female[$i] = -$female[$i];
}

#Create a XYChart object of size 280 x 300 pixels. This is the left bar
#chart.
$c2 = new XYChart(280, 300);

#Set the plotarea at (19, 0) and of size 250 x 256 pixels. Use pale blue
#(0xddddff) as the background.
$c2->setPlotArea(19, 0, 250, 256, 0xddddff);

#Add a custom text label at the top left corner of the left bar chart
$c2->addText(19, 0, "Male", "timesbi.ttf", 12, 0x7070a0);

#Add the pale blue (0xaaaaff) bar chart layer using the male data
$maleLayer = $c2->addBarLayer($male, 0xaaaaff);

#Swap the axis so that the bars are drawn horizontally
$c2->swapXY(true);

#Use a slightly negative bar gap so that the bars are packed tightly with
#the border of a bar overlaps with the borders its neighbours (that is, two
#bars sharing a common border)
$maleLayer->setBarGap(-0.1);

#Set the border style of the bars to 1 pixel 3D border
$maleLayer->setBorderColor(-1, 1);

#Add a Transparent line layer to the chart using the female data. This is
#to ensure both left and right charts are using the same data (both male
#and female data) so their auto-scaled axis will have the same scale.
$c2->addLineLayer($female, Transparent);

#Set the y-axis label format to ignore the negative sign. So even the data
#are negative, the axis will show positive numbers.
$c2->yAxis->setLabelFormat("{value|0.~~}");

#Set the y axis label font to Arial Bold
$c2->yAxis->setLabelStyle("arialbd.ttf");

#=====
#   Merge the left chart into the main chart
#=====

#Output the main chart into a DrawArea object for further manipulation
$drawarea = $c->makeChart3();

```

```
#merge the left chart into the main chart
$drawarea->merge($c2->makeChart3(), 0, 30, TopLeft, 0);

#=====
#    Add more text labels to the chart
#=====

#Add the labels for the age groups to the chart. These labels are located
#between the charts, so we handle them using custom text
for($i = 0; $i < count($labels); ++$i) {
    $textObj = $c->addText(295, $femaleLayer->getXCoor($i), $labels[$i],
        "arialbd.ttf");
    $textObj->setAlignment(Center);
}

#Add a horizontal axis title in the middle of the two charts. We also
#handle this using a custom text
$textObj = $c->addText(295, 315, "Population (in thousands)",
    "arialbi.ttf", 10);
$textObj->setAlignment(Center);

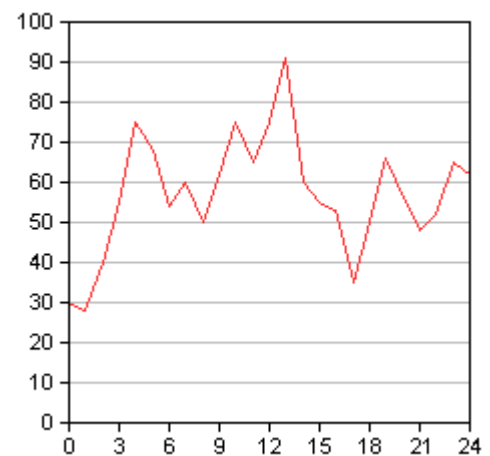
#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## Simple Line Chart

This project demonstrates the basic steps in creating a line chart. Note that the source code for this project is almost the same as that of the [Simple Bar Chart](#) example. Instead of using the [addBarLayer](#) method to create a bar chart layer, the [addLineLayer](#) method is used to create a line chart layer.

The followings are the basic steps in creating a line chart:

- Create an XYChart object using the [XYChart](#) method.
- Specify the plot area of the chart using the [setPlotArea](#) method. The plotarea is the rectangle bounded by the x-axis and the y-axis. You should leave some margin on the sides for axis labels and titles, etc.
- Specify the label on the x-axis using the [setLabels](#) method of the x-axis object.
- Add a line chart layer and specify the data to draw the line using the [addLineLayer](#) method.
- Generate the chart using the [makeChart2](#) method.



(The following program is available as “phpdemo/simpleline.php”).

```
<?php
include("phpchartdir.php");

#The data for the line chart
$data = array(30, 28, 40, 55, 75, 68, 54, 60, 50, 62, 75, 65, 75, 91, 60,
    55, 53, 35, 50, 66, 56, 48, 52, 65, 62);

#The labels for the line chart
$labels = array("0", "", "", "3", "", "", "6", "", "", "9", "", "", "12",
    "", "", "15", "", "", "18", "", "", "21", "", "", "24");

#Create a XYChart object of size 250 x 250 pixels
$c = new XYChart(250, 250);

#Set the plotarea at (30, 20) and of size 200 x 200 pixels
$c->setPlotArea(30, 20, 200, 200);

#Add a line chart layer using the given data
$c->addLineLayer($data);

#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

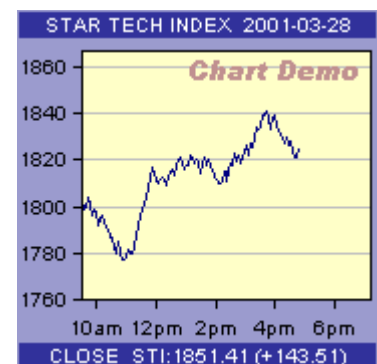
## Enhanced Line Chart

This example enhanced the previous example by adding titles, custom text, and by using different background colors for different parts of the chart.

- Create an XYChart object using the [XYChart](#) method and set its background color.
- Add title boxes on top and bottom using the [addTitle2](#) method
- Set the background color of the plot area using the [setBackground](#) method of the PlotArea object.
- Add custom text using the [addText](#) method.
- Control the fonts used on the axes using the [setLabelStyle](#) method of the BaseAxis object.

(The following program is available as “phpdemo/enhancedline.php”).

```
<?php
include("phpchartdir.php");
```



```

#
#   We use a random number generator to simulate a data collector
#   collecting data every 4 minutes from 9:28am onwards.
#
$noOfPoints = 148;
$data = array_pad(array(), $noOfPoints, 0);
$label = array_pad(array(), $noOfPoints, "");

#assume only the first 75% data points have value.
srand(1);
$data[0] = 1800;
for($i = 1; $i < $noOfPoints * 3 / 4; ++$i) {
    $data[$i] = $data[$i - 1] + rand() / getrandmax() * 10 - 5;
}
for($i = $noOfPoints * 3 / 4; $i < $noOfPoints; ++$i) {
    $data[$i] = NoValue;
}

#generate the x-axis labels at 10:00am, 12:00pm, 2:00pm, 4:00pm and 6:00pm
$startTime = 568;
$label[(10 * 60 - $startTime) / 4] = "10am";
$label[(12 * 60 - $startTime) / 4] = "12pm";
$label[(14 * 60 - $startTime) / 4] = "2pm";
$label[(16 * 60 - $startTime) / 4] = "4pm";
$label[(18 * 60 - $startTime) / 4] = "6pm";

#
#   Now we obtain the data into arrays, we can start to draw the chart
#   using ChartDirector
#

#Create a XYChart object of size 180 x 180 pixels with a blue background
#(0x9c9cce)
$c = new XYChart(180, 180, 0x9c9cce);

#Add titles to the top and bottom of the chart using 7.5pt Arial font. The
#text is white 0xffffffff on a deep blue 0x31319c background.
$c->addTitle2(Top, "STAR TECH INDEX 2001-03-28", "arial.ttf", 7.5,
    0xffffffff, 0x31319c);
$c->addTitle2(Bottom, "CLOSE STI:1851.41 (+143.51)", "arial.ttf", 7.5,
    0xffffffff, 0x31319c);

#Set the plotarea at (31, 21) and of size 145 x 124 pixels, with a pale
#yellow (0xfffffc8) background.
$c->setPlotArea(31, 21, 145, 124, 0xfffffc8);

#Add custom text at (176, 21) (top right corner of plotarea) using 11pt
#Times Bold Italic font/red (0xc09090) color
$textObj = $c->addText(176, 21, "Chart Demo", "timesbi.ttf", 11, 0xc09090);
$textObj->setAlignment(TopRight);

#Use 7.5 pts Arial as the y axis label font
$c->yAxis->setLabelStyle("", 7.5);

#Set the labels on the x axis
$c->xAxis->setLabels($label);

```

```
#Use 7.5 pts Arial as the x axis label font
$c->xAxis->setLabelStyle("", 7.5);

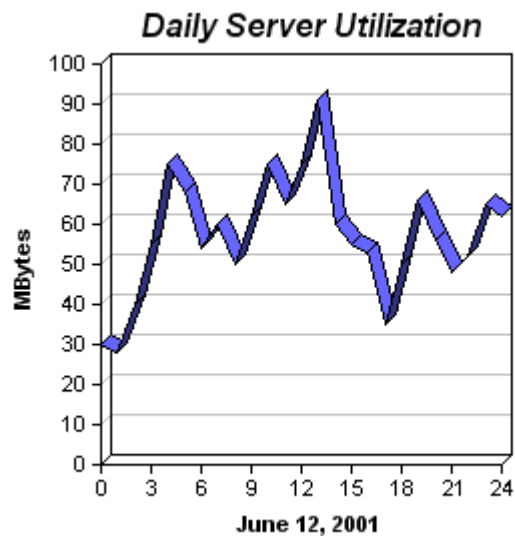
#Add a deep blue (0x000080) line layer to the chart
$c->addLineLayer($data, 0x80);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## 3D Line Chart

This example extends the previous simple line chart example by introducing the following features of the ChartDirector:

- Draw the line in 3D using the [set3D](#) method
- Add a title to the chart using the [addTitle](#) method
- Add a title to the x axis using the [setTitle](#) method of the x axis object
- Add a title to the y axis using the [setTitle](#) method of the y axis object



(The following program is available as “phpdemo/threedline.php”.)

```
<?php
include("phpchartdir.php");

#The data for the line chart
$data = array(30, 28, 40, 55, 75, 68, 54, 60, 50, 62, 75, 65, 75, 91, 60,
    55, 53, 35, 50, 66, 56, 48, 52, 65, 62);

#The labels for the line chart
$labels = array("0", "", "", "3", "", "", "6", "", "", "9", "", "", "12",
    "", "", "15", "", "", "18", "", "", "21", "", "", "24");

#Create a XYChart object of size 300 x 280 pixels
$c = new XYChart(300, 280);

#Set the plotarea at (45, 30) and of size 200 x 200 pixels
$c->setPlotArea(45, 30, 200, 200);

#Add a title to the chart using 12 pts Arial Bold Italic font
$c->addTitle("Daily Server Utilization", "arialbi.ttf", 12);
```

```
#Add a title to the y axis
$c->yAxis->setTitle("MBytes");

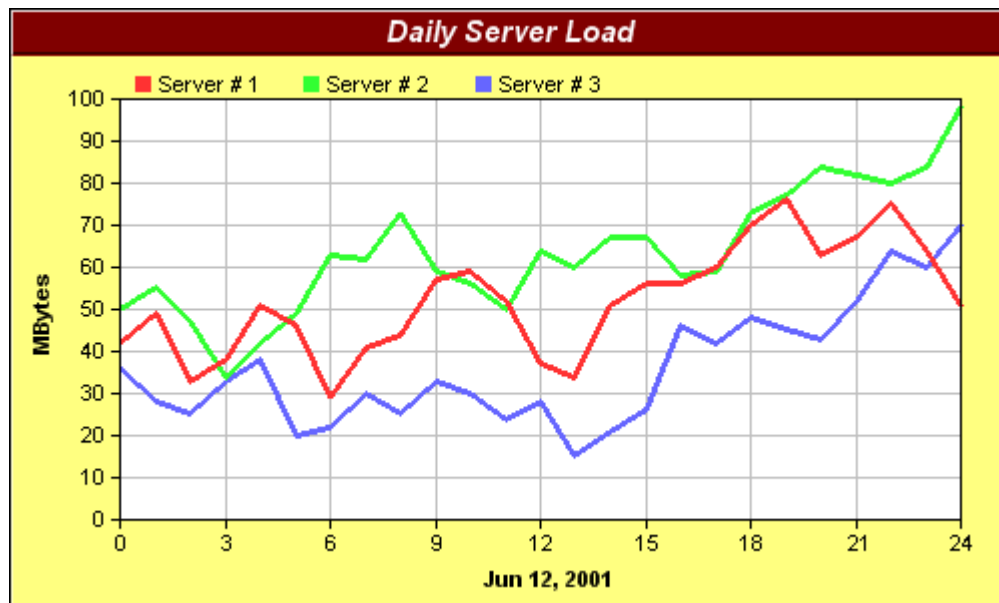
#Add a title to the x axis
$c->xAxis->setTitle("June 12, 2001");

#Add a blue (0x6666ff) 3D line chart layer using the give data
$lineLayerObj = $c->addLineLayer($data, 0x6666ff);
$lineLayerObj->set3D();

#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## Multi-Line Chart



This example demonstrates how you could plot multiple data sets on the same line chart layer. It also demonstrates how to set background colors, enable both vertical and horizontal grid lines, set the line width, and to position the legend box on the top of the chart.

- Set the background color and the 3D border for the chart by using the [setBackground](#) method
- Enable both vertical and horizontal grid lines by using the [setPlotArea](#) method
- Add a line layer using the [addLineLayer](#) method, then add multiple data sets to the line layer using the [addDataSet](#) method
- Set the line width using the [setLineWidth](#) method

- Add a legend to the top of the chart using the [addLegend](#) method

(The following program is available as “phpdemo/multiline.php”).

```
<?php
include("phpchartdir.php");

#The data for the line chart
$data0 = array(42, 49, 33, 38, 51, 46, 29, 41, 44, 57, 59, 52, 37, 34, 51,
    56, 56, 60, 70, 76, 63, 67, 75, 64, 51);
$data1 = array(50, 55, 47, 34, 42, 49, 63, 62, 73, 59, 56, 50, 64, 60, 67,
    67, 58, 59, 73, 77, 84, 82, 80, 84, 98);
$data2 = array(36, 28, 25, 33, 38, 20, 22, 30, 25, 33, 30, 24, 28, 15, 21,
    26, 46, 42, 48, 45, 43, 52, 64, 60, 70);

#The labels for the line chart
$labels = array("0", "", "", "3", "", "", "6", "", "", "9", "", "", "12",
    "", "", "15", "", "", "18", "", "", "21", "", "", "24");

#Create a XYChart object of size 500 x 300 pixels
$c = new XYChart(500, 300);

#Set background color to pale yellow 0xffff80, with a black edge and a 1
#pixel 3D border
$c->setBackground(0xffff80, 0x0, 1);

#Set the plotarea at (55, 45) and of size 420 x 210 pixels, with white
#background. Turn on both horizontal and vertical grid lines with light
#grey color (0xc0c0c0)
$c->setPlotArea(55, 45, 420, 210, 0xffffffff, -1, -1, 0xc0c0c0, -1);

#Add a legend box at (55, 25) (top of the chart) with horizontal layout.
#Use 8 pts Arial font. Set the background and border color to Transparent.
$legendObj = $c->addLegend(55, 25, false, "", 8);
$legendObj->setBackground(Transparent);

#Add a title box to the chart using 11 pts Arial Bold Italic font. The text
#is white (0xffffffff) on a dark red (0x800000) background, with a 1 pixel 3D
#border.
$titleObj = $c->addTitle("Daily Server Load", "arialbi.ttf", 11, 0xffffffff);
$titleObj->setBackground(0x800000, -1, 1);

#Add a title to the y axis
$c->yAxis->setTitle("MBytes");

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Add a title to the x axis
$c->xAxis->setTitle("Jun 12, 2001");

#Add a line layer to the chart
$layer = $c->addLineLayer();

#Set the default line width to 3 pixels
$layer->setLineWidth(3);
```

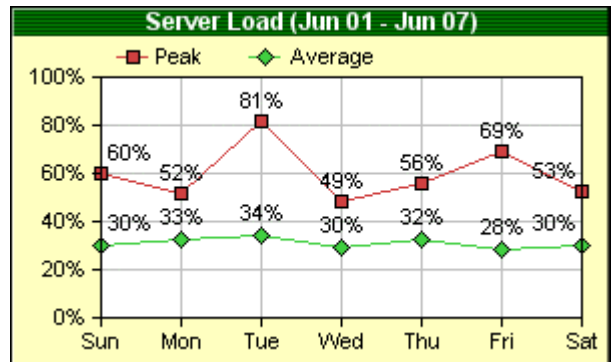
```
#Add the three data sets to the line layer
$layer->addDataSet($data0, -1, "Server # 1");
$layer->addDataSet($data1, -1, "Server # 2");
$layer->addDataSet($data2, -1, "Server # 3");

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## Symbol Line Chart

This example demonstrates plotting the data points on a line chart with “symbol” dots, with data labels above the data points. It also demonstrates how to control background colors and border effects in ChartDirector.

- Set the background color and add a 3D border to the chart using the [setBackground](#) method of the XYChart object.
- Set the background color of the title box using the [setBackground](#) method of the TextBox object. Create a pattern color using the [patternColor](#) method and use it as the background of the title box.
- Enable both vertical and horizontal grid lines using the [setPlotArea](#) method.
- Set “symbol” dots for the data points using the [setDataSymbol](#) method
- Add data labels on top of the data points using the [setDataLabelFormat](#)
- Use [setAutoScale](#) method of the YAxis object to reserve some margin at the top of the plot area. This ensures the data labels are always drawn within the plot area.



(The following program is available as “phpdemo/symbolline.php”).

```
<?php
include("phpchartdir.php");

#The data for the line chart
$data0 = array(60.2, 51.7, 81.3, 48.6, 56.2, 68.9, 52.8);
$data1 = array(30.0, 32.7, 33.9, 29.5, 32.2, 28.4, 29.8);
$labels = array("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat");

#Create a XYChart object of size 300 x 180 pixels
$c = new XYChart(300, 180);

#Set background color to pale yellow 0xffffc0, with a black edge and a 1
#pixel 3D border
```



```

$c->setBackground(0xffffc0, 0x0, 1);

#Set the plotarea at (45, 35) and of size 240 x 120 pixels, with white
#background. Turn on both horizontal and vertical grid lines with light
#grey color (0xc0c0c0)
$c->setPlotArea(45, 35, 240, 120, 0xffffffff, -1, -1, 0xc0c0c0, -1);

#Add a legend box at (45, 12) (top of the chart) using horizontal layout
#and 8 pts Arial font Set the background and border color to Transparent.
$legendObj = $c->addLegend(45, 12, false, "", 8);
$legendObj->setBackground(Transparent);

#Add a title to the chart using 9 pts Arial Bold/white font. Use a 1 x 2
#bitmap pattern as the background.
$titleObj = $c->addTitle("Server Load (Jun 01 - Jun 07)", "arialbd.ttf",
    9, 0xffffffff);
$titleObj->setBackground($c->patternColor(array(0x4000, 0x8000), 2));

#Set the y axis label format to nn%
$c->yAxis->setLabelFormat("{value}%");

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Add a line layer to the chart
$layer = $c->addLineLayer();

#Add the first line. Plot the points with a 7 pixel square symbol
$dataSetObj = $layer->addDataSet($data0, 0xcf4040, "Peak");
$dataSetObj->setDataSymbol(SquareSymbol, 7);

#Add the second line. Plot the points with a 9 pixel dismond symbol
$dataSetObj = $layer->addDataSet($data1, 0x40cf40, "Average");
$dataSetObj->setDataSymbol(DiamondSymbol, 9);

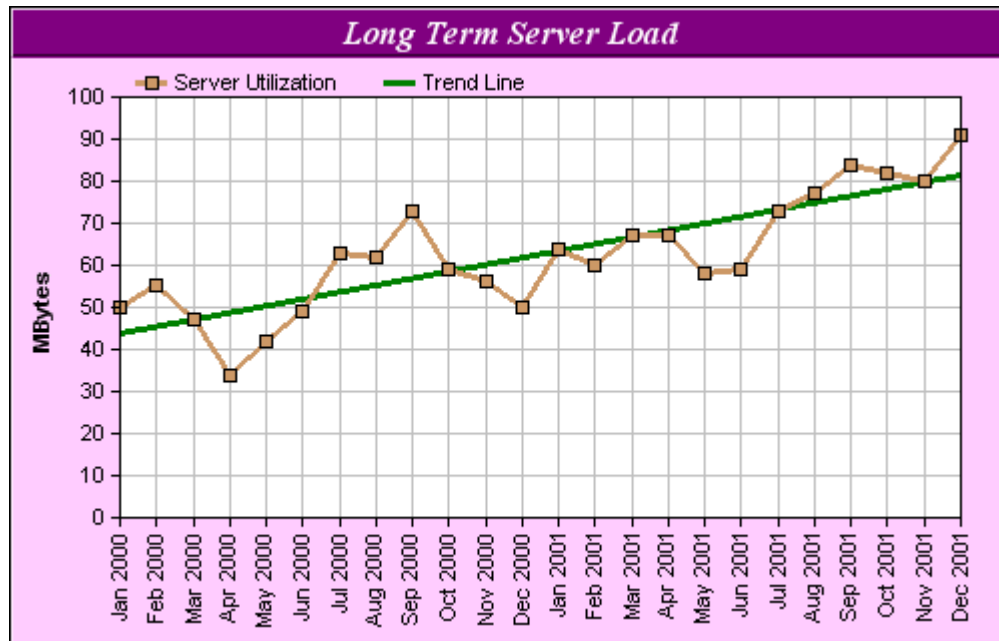
#Enable data label on the data points. Set the label format to nn%.
$layer->setDataLabelFormat("{value|0}%");

#Reserve 10% margin at the top of the plot area during auto-scaling to
#leave space for the data labels.
$c->yAxis->setAutoScale(0.1);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

# Trend Line Chart



A trend line is a straight line that fits a number of data points computed using linear regression (the least square method).

In ChartDirector, a trend line is drawn by adding a trend line layer using the [addTrendLayer](#) or [addTrendLayer2](#) method.

Typically, a trend line layer is used with other chart layers for the data points. For example, in the chart above, two layers are used. A line chart layer draws the line that joins the data points, while the trend line layer draws a straight line that best fit the data points.

Note that the x-axis labels on the above chart are rotated by 90 degrees to save space. This is achieved by using the [setFontAngle](#) method of the TextBox object that represents the x-axis label prototype.

(The following program is available as “phpdemo/trendline.php”).

```
<?php
include("phpchartdir.php");

#The data for the line chart
$data = array(50, 55, 47, 34, 42, 49, 63, 62, 73, 59, 56, 50, 64, 60, 67,
    67, 58, 59, 73, 77, 84, 82, 80, 91);

#The labels for the line chart
$labels = array("Jan 2000", "Feb 2000", "Mar 2000", "Apr 2000",
    "May 2000", "Jun 2000", "Jul 2000", "Aug 2000", "Sep 2000",
    "Oct 2000", "Nov 2000", "Dec 2000", "Jan 2001", "Feb 2001",
    "Mar 2001", "Apr 2001", "May 2001", "Jun 2001", "Jul 2001",
    "Aug 2001", "Sep 2001", "Oct 2001", "Nov 2001", "Dec 2001");
```

```

#Create a XYChart object of size 500 x 320 pixels
$c = new XYChart(500, 320);

#Set background color to pale purple 0xffccff, with a black edge and a 1
#pixel 3D border
$c->setBackground(0xffccff, 0x0, 1);

#Set the plotarea at (55, 45) and of size 420 x 210 pixels, with white
#background. Turn on both horizontal and vertical grid lines with light
#grey color (0xc0c0c0)

#Set the plotarea at (55, 45) and of size 420 x 210 pixels, with white
#background. Turn on both horizontal and vertical grid lines with light
#grey color (0xc0c0c0)
$c->setPlotArea(55, 45, 420, 210, 0xffffffff, -1, -1, 0xc0c0c0, -1);

#Add a legend box at (55, 25) (top of the chart) with horizontal layout.
#Use 8 pts Arial font. Set the background and border color to Transparent.
$legendObj = $c->addLegend(55, 25, false, "", 8);
$legendObj->setBackground(Transparent);

#Add a title box to the chart using 13 pts Times Bold Italic font. The text
#is white (0xffffffff) on a purple (0x800080) background, with a 1 pixel 3D
#border.
$titleObj = $c->addTitle("Long Term Server Load", "timesbi.ttf", 13,
    0xffffffff);
$titleObj->setBackground(0x800080, -1, 1);

#Add a title to the y axis
$c->yAxis->setTitle("MBytes");

#Set the labels on the x axis. Rotate the font by 90 degrees.
$labelsObj = $c->xAxis->setLabels($labels);
$labelsObj->setFontAngle(90);

#Add a line layer to the chart
$layer = $c->addLineLayer();

#Add the data to the line layer using light brown color (0xcc9966) with a 7
#pixel square symbol
$dataSetObj = $layer->addDataSet($data, 0xcc9966, "Server Utilization");
$dataSetObj->setDataSymbol(SquareSymbol, 7);

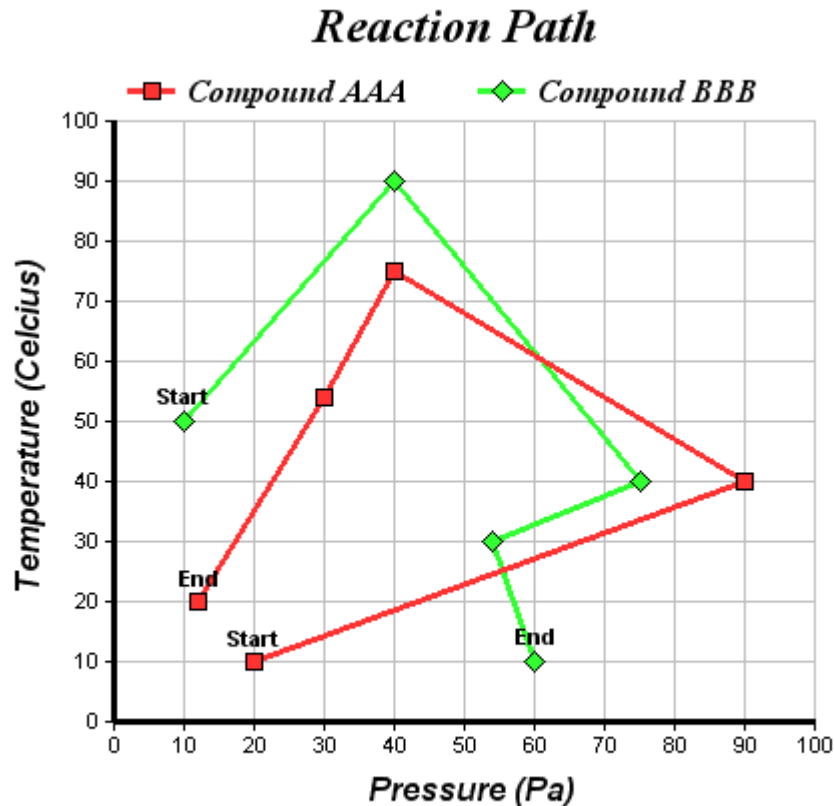
#Set the line width to 3 pixels
$layer->setLineWidth(3);

#Add a trend line layer using the same data with a dark green (0x008000)
#color. Set the line width to 3 pixels
$trendLayerObj = $c->addTrendLayer($data, 0x8000, "Trend Line");
$trendLayerObj->setLineWidth(3);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

# Arbitrary XY Line Chart



In previous examples, the data points are assumed to be equally spaced on the x-axis (e.g. one point per hour, or one point per day, etc). ChartDirector also supports irregular x values for the data points. In fact, the x values of the data points do not even need to be monotonically increasing.

This example illustrates the following ChartDirector features:

- Add a line chart layer using [addLineLayer](#) method, and set the x values of the data points using the [setXData](#) method
- Add custom labels to the selected points (the “Start” and “End” labels in the above chart) using the [addCustomDataLabel](#) method
- Set the x-axis scale using the [setLinearScale](#) method of the XAxis object
- Set the axis width using the [setWidth](#) method of the BaseAxis object

(The following program is available as “phpdemo/xyline.php”.)

```
<?php
include("phpchartdir.php");

#The (x, y) data for the first line
```

```

$dataX0 = array(20, 90, 40, 30, 12);
$dataY0 = array(10, 40, 75, 54, 20);

#The (x, y) data for the second line
$dataX1 = array(10, 40, 75, 54, 60);
$dataY1 = array(50, 90, 40, 30, 10);

#Create a XYChart object of size 450 x 450 pixels
$c = new XYChart(450, 450);

#Set the plotarea at (55, 65) and of size 350 x 300 pixels, with white
#background and a light grey border (0xc0c0c0). Turn on both horizontal and
#vertical grid lines with light grey color (0xc0c0c0)
$c->setPlotArea(55, 65, 350, 300, 0xffffffff, -1, 0xc0c0c0, 0xc0c0c0, -1);

#Add a legend box at (50, 30) (top of the chart) with horizontal layout.
#Use 12 pts Times Bold Italic font. Set the background and border color to
#Transparent.
$legendObj = $c->addLegend(50, 30, false, "timesbi.ttf", 12);
$legendObj->setBackground(Transparent);

#Add a title to the chart using 18 pts Times Bold Italic font
$c->addTitle("Reaction Path", "timesbi.ttf", 18);

#Add a title to the y axis using 12 pts Arial Bold Italic font
$c->yAxis->setTitle("Temperature (Celcius)", "arialbi.ttf", 12);

#Set the y axis line width to 3 pixels
$c->yAxis->setWidth(3);

#Reserve 10% margin at the top of the plot area during auto-scaling to
#leave space for the data labels.
$c->yAxis->setAutoScale(0.1);

#Add a title to the x axis using 12 pts Arial Bold Italic font
$c->xAxis->setTitle("Pressure (Pa)", "arialbi.ttf", 12);

#Set the x axis line width to 3 pixels
$c->xAxis->setWidth(3);

#Set the x axis scale from 0 - 100, with ticks every 10 units
$c->xAxis->setLinearScale(0, 100, 10);

#Add a red (0xff3333) line layer using dataX0 and dataY0
$layer1 = $c->addLineLayer($dataY0, 0xff3333, "Compound AAA");
$layer1->setXData($dataX0);

#Set the line width to 3 pixels
$layer1->setLineWidth(3);

#Use 9 pixel square symbols for the data points
$getDataSetObj = $layer1->getDataSet(0);
$getDataSetObj->setDataSymbol(SquareSymbol, 9);

#Add custom text labels to the first and last point on the scatter plot
#using Arial Bold font
$layer1->addCustomDataLabel(0, 0, "Start", "arialbd.ttf");

```

```

$layer1->addCustomDataLabel(0, 4, "End", "arialbd.ttf");

#Add a green (0x33ff33) line layer using dataX1 and dataY1
$layer2 = $c->addLineLayer($dataY1, 0x33ff33, "Compound BBB");
$layer2->setXData($dataX1);

#Set the line width to 3 pixels
$layer2->setLineWidth(3);

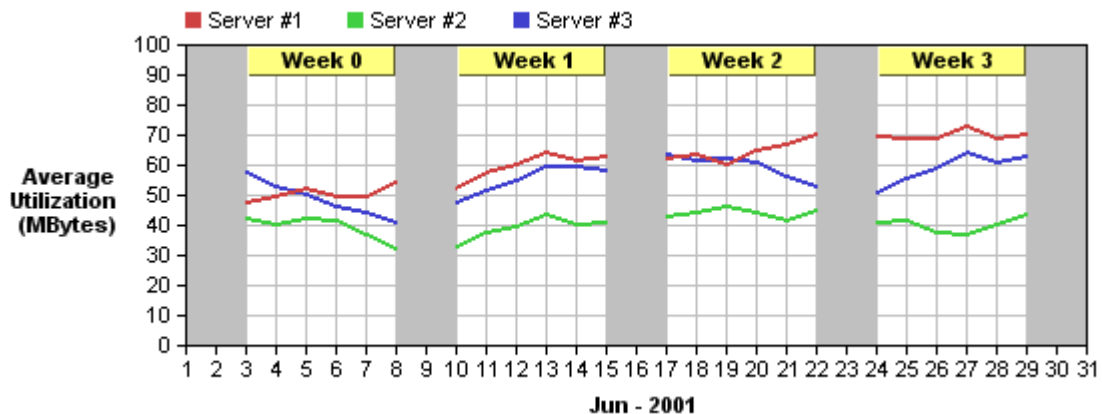
#Use 11 pixel diamond symbols for the data points
$getDataSetObj = $layer2->getDataSet(0);
$getDataSetObj->setDataSymbol(DiamondSymbol, 11);

#Add custom text labels to the first and last point on the scatter plot
#using Arial Bold font
$layer2->addCustomDataLabel(0, 0, "Start", "arialbd.ttf");
$layer2->addCustomDataLabel(0, 4, "End", "arialbd.ttf");

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## Discontinuous Lines



This example illustrates how to specify “missing” data points. In the above chart, note the lines are not continuous. This effect is achieved by setting some of the data points to the special constant [NoValue](#), which is used to denote that those data points are missing.

As an alternative to handling missing data points as discontinuous line, ChartDirector also supports directly joining the line through the missing points. The join line can be the same color and style as the line in the chart, or it can be of a different color and/or style (e.g. a dash line). All these are controlled using the [setGapColor](#) method of the LineLayer object.

In addition to illustrating “NoValue”, this example illustrates the following features of ChartDirector:

- Add vertical zones to the plot area using the [addZone](#) method of the XAxis object.

- Add custom text box to the plot area using the [addText](#) method.
- Locate the text box in the proper location using the [getXCoor](#) method of the Layer object.

(The following program is available as “phpdemo/xzone.php”).

```
<?php
include("phpchartdir.php");

#=====
#   For demo purpose, use random numbers as data for the chart
#=====

#Arrays to hold data for three lines and the labels
$data0 = array_pad(array(), 31, 0);
$data1 = array_pad(array(), 31, 0);
$data2 = array_pad(array(), 31, 0);
$labels = array_pad(array(), 31, "");

#Simulate the data using random numbers
srand(1);
$data0[0] = 40;
$data1[0] = 50;
$data2[0] = 60;
$labels[0] = "1";
for($i = 1; $i < 31; ++$i) {
    $data0[$i] = $data0[$i - 1] + rand() / getrandmax() * 10 - 5;
    $data1[$i] = $data1[$i - 1] + rand() / getrandmax() * 10 - 5;
    $data2[$i] = $data2[$i - 1] + rand() / getrandmax() * 10 - 5;
    $labels[$i] = $i + 1;
}

#Simulate some data points have no data value
for($i = 1; $i < 30; $i += 7) {
    $data0[$i] = NoValue;
    $data1[$i] = NoValue;
    $data2[$i] = NoValue;
}

#=====
#   Now we have the data ready. Actually drawing the chart.
#=====

#Create a XYChart object of size 600 x 220 pixels
$c = new XYChart(600, 220);

#Set the plot area at (100, 25) and of size 450 x 150 pixels. Enabled both
#vertical and horizontal grids by setting their colors to light grey
#(0xc0c0c0)
$plotAreaObj = $c->setPlotArea(100, 25, 450, 150);
$plotAreaObj->setGridColor(0xc0c0c0, 0xc0c0c0);

#Add a legend box (92, 0) (top of plot area) using horizontal layout. Use 8
#pts Arial font. Disable bounding box (set border to transparent).
$legendObj = $c->addLegend(92, 0, false, "", 8);
$legendObj->setBackground(Transparent);
```

```

#Add a title to the y axis. Draw the title upright (font angle = 0)
$titleObj = $c->yAxis->setTitle("Average\nUtilization\n(MBytes)");
$titleObj->setFontAngle(0);

#Use manually scaling of y axis from 0 to 100, with ticks every 10 units
$c->yAxis->setLinearScale(0, 100, 10);

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Set the title on the x axis
$c->xAxis->setTitle("Jun - 2001");

#Add x axis (vertical) zones to indicate Saturdays and Sundays
for($i = 0; $i < 29; $i += 7) {
    $c->xAxis->addZone($i, $i + 2, 0xc0c0c0);
}

#Add a line layer to the chart
$layer = $c->addLineLayer();

#Set the default line width to 2 pixels
$layer->setLineWidth(2);

#Add the three data sets to the line layer
$layer->addDataSet($data0, 0xcf4040, "Server #1");
$layer->addDataSet($data1, 0x40cf40, "Server #2");
$layer->addDataSet($data2, 0x4040cf, "Server #3");

#Layout the chart to fix the y axis scaling. We can then use getXCoor and
#getYCoor to determine the position of custom objects.
$c->layout();

#Add the "week n" custom text boxes at the top of the plot area.
for($i = 0; $i < 4; ++$i) {
    #Add the "week n" text box using 8 pt Arial font with top center
    #alignment.
    $textbox = $c->addText($layer->getXCoor($i * 7 + 2), 25, "Week ".$i,
        "arialbd.ttf", 8, 0x0, TopCenter);

    #Set the box width to cover five days
    $textbox->setSize($layer->getXCoor($i * 7 + 7) - $layer->getXCoor($i *
        7 + 2) + 1, 0);

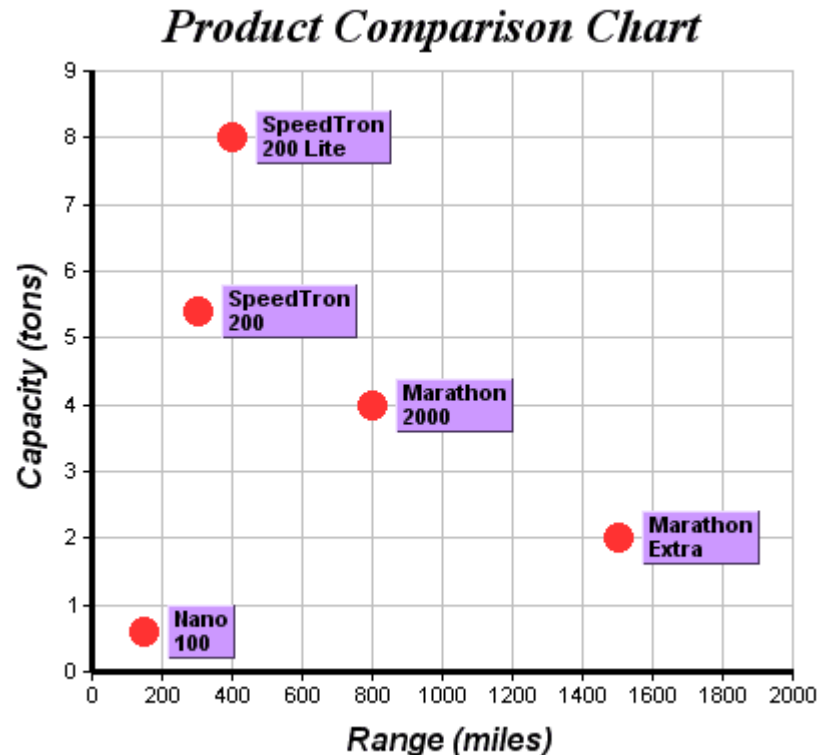
    #Set box background to pale yellow 0xffff80, with a 1 pixel 3D border
    $textbox->setBackground(0xffff80, Transparent, 1);
}

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```



# Scatter Chart



A scatter chart can be considered as a special kind of line chart, in which the data symbols are enabled and the line width is set to zero. Therefore only the data symbols are visible.

In ChartDirector, scatter chart is drawn by using the [addScatterLayer](#) method.

In the chart above, the scatter chart is drawn using 15-pixel circles to represents the data points. The labels on the left of the data points are created using the [addCustomDataLabel](#) method.

(The following program is available as “phpdemo/scatter.php”.)

```
<?php
include("phpchartdir.php");

#The XY points for the scatter chart
$dataX = array(150, 400, 300, 1500, 800);
$dataY = array(0.6, 8, 5.4, 2, 4);

#The labels for the points
$labels = array("Nano\n100", "SpeedTron\n200 Lite", "SpeedTron\n200",
    "Marathon\nExtra", "Marathon\n2000");

#Create a XYChart object of size 450 x 400 pixels
$c = new XYChart(450, 400);

#Set the plotarea at (55, 40) and of size 350 x 300 pixels, with white
#background and a light grey border (0xc0c0c0). Turn on both horizontal and
```

```

#vertical grid lines with light grey color (0xc0c0c0)
$c->setPlotArea(55, 40, 350, 300, 0xffffffff, -1, 0xc0c0c0, 0xc0c0c0, -1);

#Add a title to the chart using 18 pts Times Bold Italic font.
$c->addTitle("Product Comparison Chart", "timesbi.ttf", 18);

#Add a title to the y axis using 12 pts Arial Bold Italic font
$c->yAxis->setTitle("Capacity (tons)", "arialbi.ttf", 12);

#Set the y axis line width to 3 pixels
$c->yAxis->setWidth(3);

#Reserve 10% margin at the top of the plot area during auto-scaling to
#leave space for the data labels.
$c->yAxis->setAutoScale(0.1);

#Add a title to the y axis using 12 pts Arial Bold Italic font
$c->xAxis->setTitle("Range (miles)", "arialbi.ttf", 12);

#Set the x axis line width to 3 pixels
$c->xAxis->setWidth(3);

#Set the x axis scale from 0 - 2000, with ticks every 200 units
$c->xAxis->setLinearScale(0, 2000, 200);

#Add the data as a scatter chart layer, using a 15 pixel circle as the
#symbol
$layer = $c->addScatterLayer($dataX, $dataY, "Server BBB", CircleSymbol,
    15, 0xff3333, 0xff3333);

#Add the labels besides the data points using custom data labels
for($i = 0; $i < count($labels); ++$i) {
    #Add a custom data label to data point i using 8 ptrs Arial Bold font
    $textbox = $layer->addCustomDataLabel(0, $i, $labels[$i],
        "arialbd.ttf", 8);

    #Set the background of the custom label to purple (0xcc99ff) with a 1
    #pixel 3D border
    $textbox->setBackground(0xcc99ff, Transparent, 1);

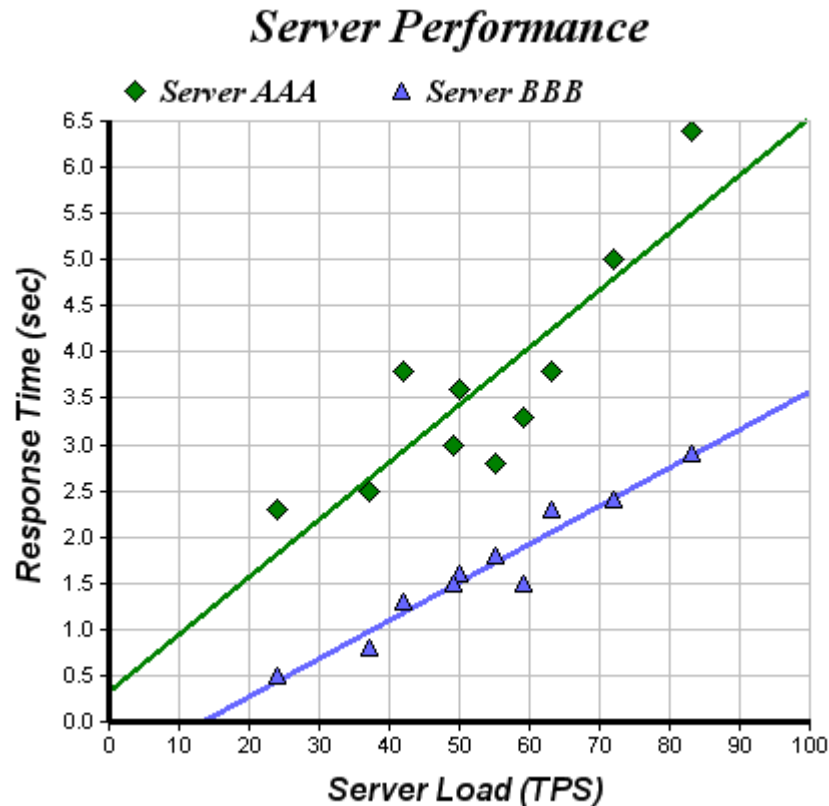
    #Align the text box so that the data point is on the left side
    $textbox->setAlignment(Left);

    #Add (4, 0) offset to the text box position (move right by 4 pixels)
    $textbox->setPos(4, 0);
}

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

# Scatter Trend Chart



This example illustrates a popular chart type, which is created by combining scatter charts with trend line charts. The chart above actually contains 4 layers – 2 scatter chart layers, and 2 trend line layers.

(The following program is available as “phpdemo/scattertrend.php”.)

```
<?php
include("phpchartdir.php");

#The XY data of the first data series
$dataX0 = array(50, 55, 37, 24, 42, 49, 63, 72, 83, 59);
$dataY0 = array(3.6, 2.8, 2.5, 2.3, 3.8, 3.0, 3.8, 5.0, 6.4, 3.3);

#The XY data of the second data series
$dataX1 = array(50, 55, 37, 24, 42, 49, 63, 72, 83, 59);
$dataY1 = array(1.6, 1.8, 0.8, 0.5, 1.3, 1.5, 2.3, 2.4, 2.9, 1.5);

#Create a XYChart object of size 450 x 420 pixels
$c = new XYChart(450, 420);

#Set the plotarea at (55, 65) and of size 350 x 300 pixels, with white
#background and a light grey border (0xc0c0c0). Turn on both horizontal and
#vertical grid lines with light grey color (0xc0c0c0)
$c->setPlotArea(55, 65, 350, 300, 0xffffffff, -1, 0xc0c0c0, 0xc0c0c0, -1);

#Add a legend box at (50, 30) (top of the chart) with horizontal layout.
```

```

#Use 12 pts Times Bold Italic font. Set the background and border color to
#Transparent.
$legendObj = $c->addLegend(50, 30, false, "timesbi.ttf", 12);
$legendObj->setBackground(Transparent);

#Add a title to the chart using 18 point Times Bold Italic font.
$c->addTitle("Server Performance", "timesbi.ttf", 18);

#Add a title to the y axis using 12 pts Arial Bold Italic font
$c->yAxis->setTitle("Response Time (sec)", "arialbi.ttf", 12);

#Set the y axis line width to 3 pixels
$c->yAxis->setWidth(3);

#Set the y axis label format to show 1 decimal point
$c->yAxis->setLabelFormat("{value|1}");

#Add a title to the x axis using 12 pts Arial Bold Italic font
$c->xAxis->setTitle("Server Load (TPS)", "arialbi.ttf", 12);

#Set the x axis line width to 3 pixels
$c->xAxis->setWidth(3);

#Set the x axis scale from 0 - 100, with ticks every 10 units
$c->xAxis->setLinearScale(0, 100, 10);

#Add a scatter layer using (dataX0, dataY0)
$c->addScatterLayer($dataX0, $dataY0, "Server AAA", DiamondSymbol, 11,
  0x8000);

#Add a trend line layer for (dataX0, dataY0)
$trendLayer2Obj = $c->addTrendLayer2($dataX0, $dataY0, 0x8000);
$trendLayer2Obj->setLineWidth(3);

#Add a scatter layer for (dataX1, dataY1)
$c->addScatterLayer($dataX1, $dataY1, "Server BBB", TriangleSymbol, 9,
  0x6666ff);

#Add a trend line layer for (dataX1, dataY1)
$trendLayer2Obj = $c->addTrendLayer2($dataX1, $dataY1, 0x6666ff);
$trendLayer2Obj->setLineWidth(3);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

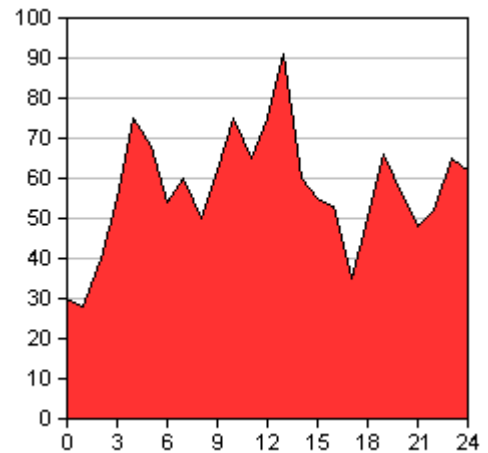
```

## Simple Area Chart

This example demonstrates the basic steps in creating an area chart. Note that the source code for this project is almost the same as that of the [Simple Line Chart](#) example. The difference is that instead of using the [addLineLayer](#) method to create a line chart layer, the [addAreaLayer](#) method is used to create an area chart layer.

The followings are the basic steps in creating an area chart:

- Create a XYChart object using the [XYChart](#) method.
- Specify the plot area of the chart using the [setPlotArea](#) method. The plotarea is the rectangle bounded by the x-axis and the y-axis. You should leave some margin on the sides for axis labels and titles, etc.
- Specify the label on the x-axis using the [setLabels](#) method of the x-axis object.
- Add an area chart layer and specify the data to draw the area using the [addAreaLayer](#) method.
- Generate the chart using the [makeChart2](#) method.



(The following program is available as “phpdemo/simplearea.php”).)

```
<?php
include("phpchartdir.php");

#The data for the area chart
$data = array(30, 28, 40, 55, 75, 68, 54, 60, 50, 62, 75, 65, 75, 91, 60,
    55, 53, 35, 50, 66, 56, 48, 52, 65, 62);

#The labels for the area chart
$labels = array("0", "", "", "3", "", "", "6", "", "", "9", "", "", "12",
    "", "", "15", "", "", "18", "", "", "21", "", "", "24");

#Create a XYChart object of size 250 x 250 pixels
$c = new XYChart(250, 250);

#Set the plotarea at (30, 20) and of size 200 x 200 pixels
$c->setPlotArea(30, 20, 200, 200);

#Add an area chart layer using the given data
$c->addAreaLayer($data);

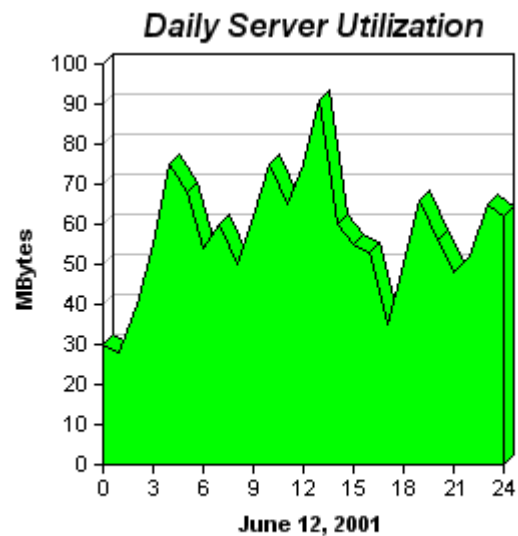
#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## 3D Area Chart

This example extends the previous simple area chart example by introducing the following features of the ChartDirector:

- Draw the area in 3D using the [set3D](#) method
- Add a title to the chart using the [addTitle](#) method
- Add a title to the x axis using the [setTitle](#) method of the x axis object
- Add a title to the y axis using the [setTitle](#) method of the y axis object



(The following program is available as “phpdemo/threedarea.php”.)

```
<?php
include("phpchartdir.php");

#The data for the area chart
$data = array(30, 28, 40, 55, 75, 68, 54, 60, 50, 62, 75, 65, 75, 91, 60,
    55, 53, 35, 50, 66, 56, 48, 52, 65, 62);

#The labels for the area chart
$labels = array("0", "", "", "3", "", "", "6", "", "", "9", "", "", "12",
    "", "", "15", "", "", "18", "", "", "21", "", "", "24");

#Create a XYChart object of size 300 x 300 pixels
$c = new XYChart(300, 300);

#Set the plotarea at (45, 30) and of size 200 x 200 pixels
$c->setPlotArea(45, 30, 200, 200);

#Add a title to the chart using 12 pts Arial Bold Italic font
$c->addTitle("Daily Server Utilization", "arialbi.ttf", 12);

#Add a title to the y axis
$c->yAxis->setTitle("MBytes");

#Add a title to the x axis
$c->xAxis->setTitle("June 12, 2001");

#Add a green (0x00ff00) 3D area chart layer using the give data
$areaLayerObj = $c->addAreaLayer($data, 0xff00);
```

```

$areaLayerObj->set3D();

#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

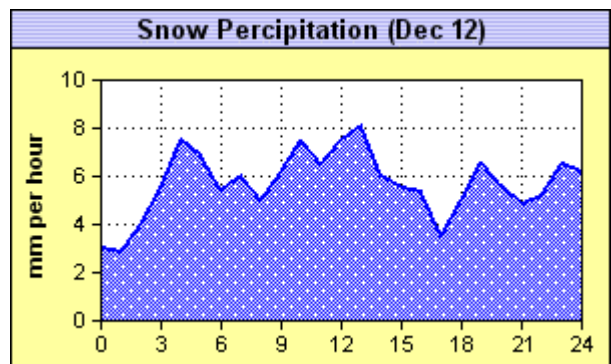
#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## Line Area Chart

This example illustrates how to create a nice looking area chart in which the boundary line of the area chart is highlighted. It also demonstrates filling the areas in an area chart using a pattern.

- Create a chart with a background color and a border using the [XYChart](#) method.
- Set the background color of the title box using the [setBackground](#) method of the [TextBox](#) object. Create a pattern color using the [patternColor](#) method and use it as the background of the title box.
- Enable both vertical and horizontal grid lines using the [setPlotArea](#) method. Use the [dashLineColor](#) method to create dash line colors for the grid lines.
- Load a “snow pattern” from an image file using the [patternColor2](#) method.
- Use the [setDataColor](#) method of the [DataSet](#) object to specify the area and line colors. Use the “snow pattern” as the area color.
- Use the [setLineWidth](#) method of the layer object to increase the line width to highlight it



(The following program is available as “phpdemo/linearea.php”).

```

<?php
include("phpchartdir.php");

#The data for the area chart
$data = array(3.0, 2.8, 4.0, 5.5, 7.5, 6.8, 5.4, 6.0, 5.0, 6.2, 7.5, 6.5,
    7.5, 8.1, 6.0, 5.5, 5.3, 3.5, 5.0, 6.6, 5.6, 4.8, 5.2, 6.5, 6.2);

#The labels for the area chart
$labels = array("0", "", "", "3", "", "", "6", "", "", "9", "", "", "12",
    "", "", "15", "", "", "18", "", "", "21", "", "", "24");

#Create a XYChart object of size 300 x 180 pixels. Set the background to
#pale yellow (0xffffa0) with a black border (0x0)

```

```

$c = new XYChart(300, 180, 0xffffa0, 0x0);

#Set the plotarea at (45, 35) and of size 240 x 120 pixels. Set the
#background to white (0xffffffff). Set both horizontal and vertical grid
#lines to black (&H0&) dotted lines (pattern code 0x0103)
$c->setPlotArea(45, 35, 240, 120, 0xffffffff, -1, -1, $c->dashLineColor(0x0,
    0x103), $c->dashLineColor(0x0, 0x103));

#Add a title to the chart using 10 pts Arial Bold font. Use a 1 x 2 bitmap
#pattern as the background. Set the border to black (0x0).
$titleObj = $c->addTitle("Snow Percipitation (Dec 12)", "arialbd.ttf", 10);
$titleObj->setBackground($c->patternColor(array(0xb0b0f0, 0xe0e0ff), 2),
    0x0);

#Add a title to the y axis
$c->yAxis->setTitle("mm per hour");

#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

#Add an area layer to the chart
$layer = $c->addAreaLayer();

#Load a snow pattern from an external file "snow.png".
$snowPattern = $c->patternColor(dirname($PATH_TRANSLATED)."/snow.png");

#Add a data set to the area layer using the snow pattern as the fill color.
#Use deep blue (0x0000ff) as the area border line color (&H0000ff&)
$dataSetObj = $layer->addDataSet($data);
$dataSetObj->setDataColor($snowPattern, 0xff);

#Set the line width to 2 pixels to highlight the line
$layer->setLineWidth(2);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

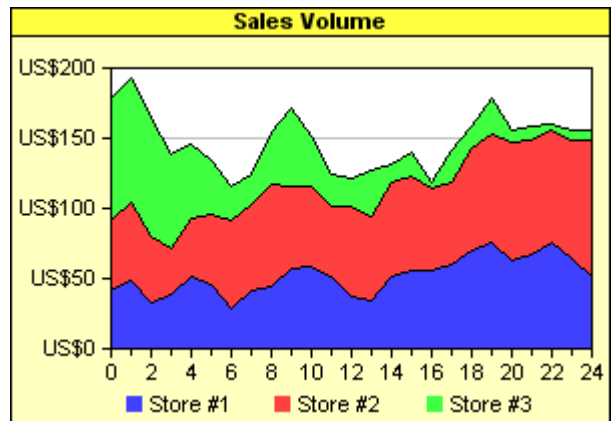


# Stacked Area Chart

This example illustrates how you could include multiple data sets in the same area chart. In this case, the areas will simply stack on top of one another.

This example demonstrate the following features of the ChartDirector:

- Add an area layer using the [addAreaLayer](#) method, and the add multiple data sets to the area layer using the [addDataSet](#) method
- Add a legend to the bottom of the chart using the [addLegend](#) method
- Set the y-axis label format to “US\$nn” using the [setLabelFormat](#) method of the YAxis object.



(The following program is available as “phpdemo/stackedarea.php”).

```
<?php
include("phpchartdir.php");

#The data for the area chart
$data0 = array(42, 49, 33, 38, 51, 46, 29, 41, 44, 57, 59, 52, 37, 34, 51,
    56, 56, 60, 70, 76, 63, 67, 75, 64, 51);
$data1 = array(50, 55, 47, 34, 42, 49, 63, 62, 73, 59, 56, 50, 64, 60, 67,
    67, 58, 59, 73, 77, 84, 82, 80, 84, 98);
$data2 = array(87, 89, 85, 66, 53, 39, 24, 21, 37, 56, 37, 22, 21, 33, 13,
    17, 4, 23, 16, 25, 9, 10, 5, 7, 6);
$labels = array("0", "-", "2", "-", "4", "-", "6", "-", "8", "-", "10",
    "-", "12", "-", "14", "-", "16", "-", "18", "-", "20", "-", "22", "-",
    "24");

#Create a XYChart object of size 300 x 210 pixels. Set the background to
#pale yellow (0xffffc0) with a black border (0x0)
$c = new XYChart(300, 210, 0xffffc0, 0x0);

#Set the plotarea at (50, 30) and of size 240 x 140 pixels. Use white
#(0xffffffff) background.
$plotAreaObj = $c->setPlotArea(50, 30, 240, 140);
$plotAreaObj->setBackground(0xffffffff);

#Add a legend box at (50, 185) (below of plot area) using horizontal
#layout. Use 8 pts Arial font with Transparent background.
$legendObj = $c->addLegend(50, 185, false, "", 8);
$legendObj->setBackground(Transparent);

#Add a title box to the chart using 8 pts Arial Bold font, with yellow
#(0xffff40) background and a black border (0x0)
$titleObj = $c->addTitle("Sales Volume", "arialbd.ttf", 8);
$titleObj->setBackground(0xffff40, 0);
```

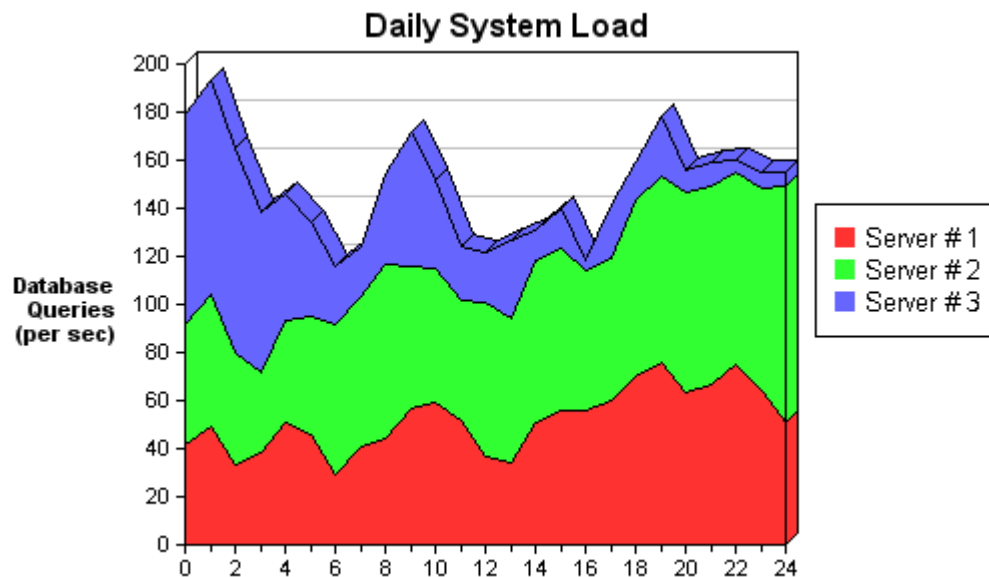
```
#Set the y axis label format to US$nnnn
$c->yAxis->setLabelFormat("US\${value}");

#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

#Add an stack area layer with three data sets
$layer = $c->addAreaLayer2(Stack);
$layer->addDataSet($data0, 0x4040ff, "Store #1");
$layer->addDataSet($data1, 0xff4040, "Store #2");
$layer->addDataSet($data2, 0x40ff40, "Store #3");

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## 3D Stacked Area Chart



This example illustrates a stacked area chart in 3D. It demonstrates the following features of the ChartDirector:

- Add an area layer using the [addAreaLayer](#) method, and the add multiple data sets to the area layer using the [addDataSet](#) method
- Add a legend to the chart using the [addLegend](#) method
- Add a title to the y-axis using the [addTitle](#) method, and draw the title upright using the [setFontAngle](#) method. Note that the y-axis title can contain multiple lines. This is by including the line break character in the title.

(The following program is available as “phpdemo/ 3dstackarea.php”.)

```

<?php
include("phpchartdir.php");

#The data for the area chart
$data0 = array(42, 49, 33, 38, 51, 46, 29, 41, 44, 57, 59, 52, 37, 34, 51,
    56, 56, 60, 70, 76, 63, 67, 75, 64, 51);
$data1 = array(50, 55, 47, 34, 42, 49, 63, 62, 73, 59, 56, 50, 64, 60, 67,
    67, 58, 59, 73, 77, 84, 82, 80, 84, 98);
$data2 = array(87, 89, 85, 66, 53, 39, 24, 21, 37, 56, 37, 22, 21, 33, 13,
    17, 4, 23, 16, 25, 9, 10, 5, 7, 6);
$labels = array("0", "-", "2", "-", "4", "-", "6", "-", "8", "-", "10",
    "-", "12", "-", "14", "-", "16", "-", "18", "-", "20", "-", "22", "-",
    "24");

#Create a XYChart object of size 500 x 300 pixels
$c = new XYChart(500, 300);

#Set the plotarea at (90, 30) and of size 300 x 240 pixels.
$c->setPlotArea(90, 30, 300, 240);

#Add a legend box at (405, 100)
$c->addLegend(405, 100);

#Add a title to the chart
$c->addTitle("Daily System Load");

#Add a title to the y axis. Draw the title upright (font angle = 0)
$titleObj = $c->yAxis->setTitle("Database\nQueries\n(per sec)");
$titleObj->setFontAngle(0);

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Add an area layer
$layer = $c->addAreaLayer();

#Draw the area layer in 3D
$layer->set3D();

#Add the three data sets to the area layer
$layer->addDataSet($data0, -1, "Server # 1");
$layer->addDataSet($data1, -1, "Server # 2");
$layer->addDataSet($data2, -1, "Server # 3");

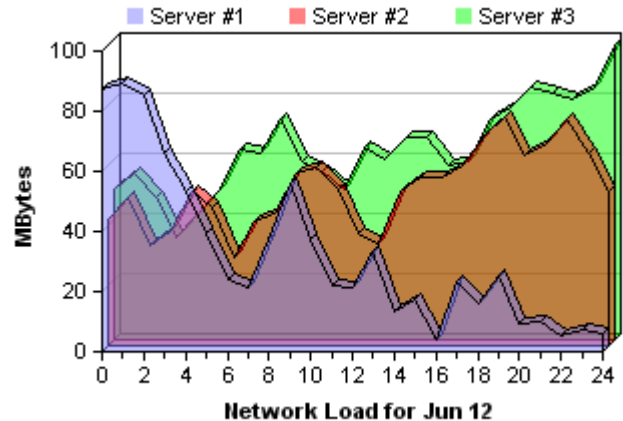
#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## Depth Area Chart

This example illustrates how you could display multiple data sets by using layers. ChartDirector allows any arbitrary XY chart types for each layer. In this particular example, all layers are of area chart type.

Note also the transparency feature of ChartDirector. The areas are drawn in semi-transparent colors so that you can see through the areas.



(The following program is available as “phpdemo/deptharea.php”).

```
<?php
include("phpchartdir.php");

#The data for the area chart
$data0 = array(42, 49, 33, 38, 51, 46, 29, 41, 44, 57, 59, 52, 37, 34, 51,
    56, 56, 60, 70, 76, 63, 67, 75, 64, 51);
$data1 = array(50, 55, 47, 34, 42, 49, 63, 62, 73, 59, 56, 50, 64, 60, 67,
    67, 58, 59, 73, 77, 84, 82, 80, 84, 98);
$data2 = array(87, 89, 85, 66, 53, 39, 24, 21, 37, 56, 37, 22, 21, 33, 13,
    17, 4, 23, 16, 25, 9, 10, 5, 7, 6);
$labels = array("0", "-", "2", "-", "4", "-", "6", "-", "8", "-", "10",
    "-", "12", "-", "14", "-", "16", "-", "18", "-", "20", "-", "22", "-",
    "24");

#Create a XYChart object of size 350 x 230 pixels
$c = new XYChart(350, 230);

#Set the plotarea at (50, 30) and of size 250 x 150 pixels.
$c->setPlotArea(50, 30, 250, 150);

#Add a legend box at (55, 0) (top of the chart) using 8 pts Arial Font. Set
#background and border to Transparent.
$legendObj = $c->addLegend(55, 0, false, "", 8);
$legendObj->setBackground(Transparent);

#Add a title to the x axis
$c->xAxis->setTitle("Network Load for Jun 12");

#Add a title to the y axis
$c->yAxis->setTitle("MBytes");

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Add three area layers, each representing one data set. The areas are drawn
#in semi-transparent colors.
```

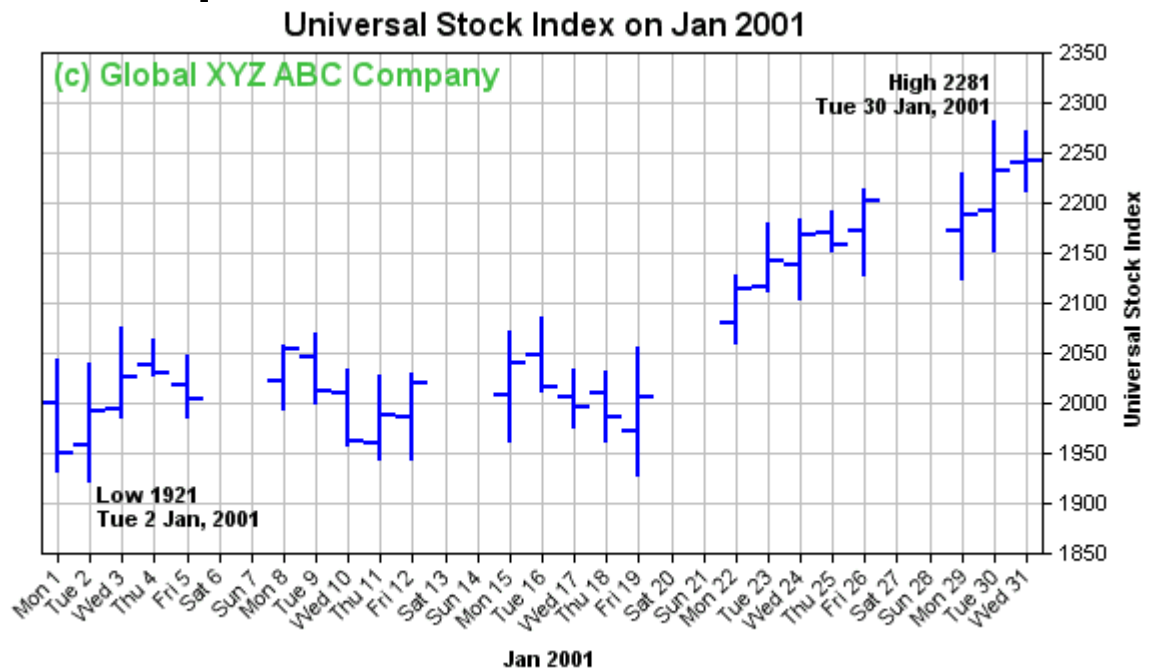
```

$c->addAreaLayer($data2, 0x808080ff, "Server #1", 3);
$c->addAreaLayer($data0, 0x80ff0000, "Server #2", 3);
$c->addAreaLayer($data1, 0x8000ff00, "Server #3", 3);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## High-Low-Open-Close Chart



This example illustrates the high-low-open-close chart, as well as a number of features of ChartDirector. These include:

- Add a HLOC layer to the chart using the [addHLOCLayer](#) method.
- Add custom text to the chart using the [addText](#) method. Note the green “(c) Global XYZ ABC Company” text inside the plot area.
- Add custom data labels using the [addCustomDataLabel](#) method. Note the “High 2281” and “Low 1921” texts in this chart.
- Reserve spaces on the top and bottom of the chart for putting custom texts using the [setAutoScale](#) method
- Use the special constant [NoValue](#) to denote that some points have no value. Note that in the above chart, Saturday and Sunday data points have no value.

- Draw the y-axis on the right using the [setYAxisOnRight](#) method.
- Draw the x-axis labels at 45 degrees using the [setFontAngle](#) method.

(The following program is available as “phpdemo/hloc.php”.)

```
<?php
include("phpchartdir.php");

#
#Sample data for the HLOC chart. Represents the high, low, open and close
#values for 31 days
#
$nil = NoValue;

$highData = array(2043, 2039, 2076, 2064, 2048, $nil, $nil, 2058, 2070,
    2033, 2027, 2029, $nil, $nil, 2071, 2085, 2034, 2031, 2056, $nil,
    $nil, 2128, 2180, 2183, 2192, 2213, $nil, $nil, 2230, 2281, 2272);

$lowData = array(1931, 1921, 1985, 2028, 1986, $nil, $nil, 1994, 1999,
    1958, 1943, 1944, $nil, $nil, 1962, 2011, 1975, 1962, 1928, $nil,
    $nil, 2059, 2112, 2103, 2151, 2127, $nil, $nil, 2123, 2152, 2212);

$openData = array(2000, 1957, 1993, 2037, 2018, $nil, $nil, 2021, 2045,
    2009, 1959, 1985, $nil, $nil, 2008, 2048, 2006, 2010, 1971, $nil,
    $nil, 2080, 2116, 2137, 2170, 2172, $nil, $nil, 2171, 2191, 2240);

$closeData = array(1950, 1991, 2026, 2029, 2004, $nil, $nil, 2053, 2011,
    1962, 1987, 2019, $nil, $nil, 2040, 2016, 1996, 1985, 2006, $nil,
    $nil, 2113, 2142, 2167, 2158, 2201, $nil, $nil, 2188, 2231, 2242);

#The labels for the HLOC chart
$labels = array("Mon 1", "Tue 2", "Wed 3", "Thu 4", "Fri 5", "Sat 6",
    "Sun 7", "Mon 8", "Tue 9", "Wed 10", "Thu 11", "Fri 12", "Sat 13",
    "Sun 14", "Mon 15", "Tue 16", "Wed 17", "Thu 18", "Fri 19", "Sat 20",
    "Sun 21", "Mon 22", "Tue 23", "Wed 24", "Thu 25", "Fri 26", "Sat 27",
    "Sun 28", "Mon 29", "Tue 30", "Wed 31");

#Create a XYChart object of size 600 x 350 pixels
$c = new XYChart(600, 350);

#Set the plotarea at (50, 25) and of size 500 x 250 pixels. Enable both the
#horizontal and vertical grids by setting their colors to grey (0xc0c0c0)
$plotAreaObj = $c->setPlotArea(50, 25, 500, 250);
$plotAreaObj->setGridColor(0xc0c0c0, 0xc0c0c0);

#Add a title to the chart
$c->addTitle("Universal Stock Index on Jan 2001");

#Add a custom text at (50, 25) (the upper left corner of the plotarea). Use
#12 pts Arial Bold/pale green (0x40c040) as the font.
$c->addText(50, 25, "(c) Global XYZ ABC Company", "arialbd.ttf", 12,
    0x40c040);

#Add a title to the x axis
$c->xAxis->setTitle("Jan 2001");
```

```

#Set the labels on the x axis. Rotate the labels by 45 degrees.
$labelsObj = $c->xAxis->setLabels($labels);
$labelsObj->setFontAngle(45);

#Add a title to the y axis
$c->yAxis->setTitle("Universal Stock Index");

#Draw the y axis on the right hand side of the plot area
$c->setYAxisOnRight(true);

#Reserve 10% margin at the top and bottom of the plot area during
#auto-scaling. This is to leave space for the high and low data labels.
$c->yAxis->setAutoScale(0.1, 0.1);

#Add an HLOC layer using blue 0x0000ff color
$layer = $c->addHLOCLayer($highData, $lowData, $openData, $closeData, 0xff)
;

#Set the line width to 2 pixels
$layer->setLineWidth(2);

#
#Now we add the "High" and "Low" text labels. We first find out which are
#the highest and lowest positions.
#

$highPos = 0;
$lowPos = 0;
for($i = 0; $i < count($highData); ++$i) {
    if ($highData[$i] != NoValue & $highData[$i] > $highData[$highPos]) {
        $highPos = $i;
    }
    if ($lowData[$i] != NoValue & $lowData[$i] < $lowData[$lowPos]) {
        $lowPos = $i;
    }
}

#By default, we put text at the center position. If the data point is too
#close to the right or left border of the plot area, we align the text to
#the right and left to avoid the text overflows outside the plot area
$align = BottomCenter;
if ($highPos > 25) {
    $align = BottomRight;
} else if ($highPos < 5) {
    $align = BottomLeft;
}

#Add the custom high label at the high position
$customDataLabelObj = $layer->addCustomDataLabel(0, $highPos,
    "High {high}\n{xLabel} Jan, 2001", "arialbd.ttf");
$customDataLabelObj->setAlignment($align);

#Similarly, we compute the alignment for the low label based on its x
#position.
$align = TopCenter;
if ($lowPos > 25) {

```

```

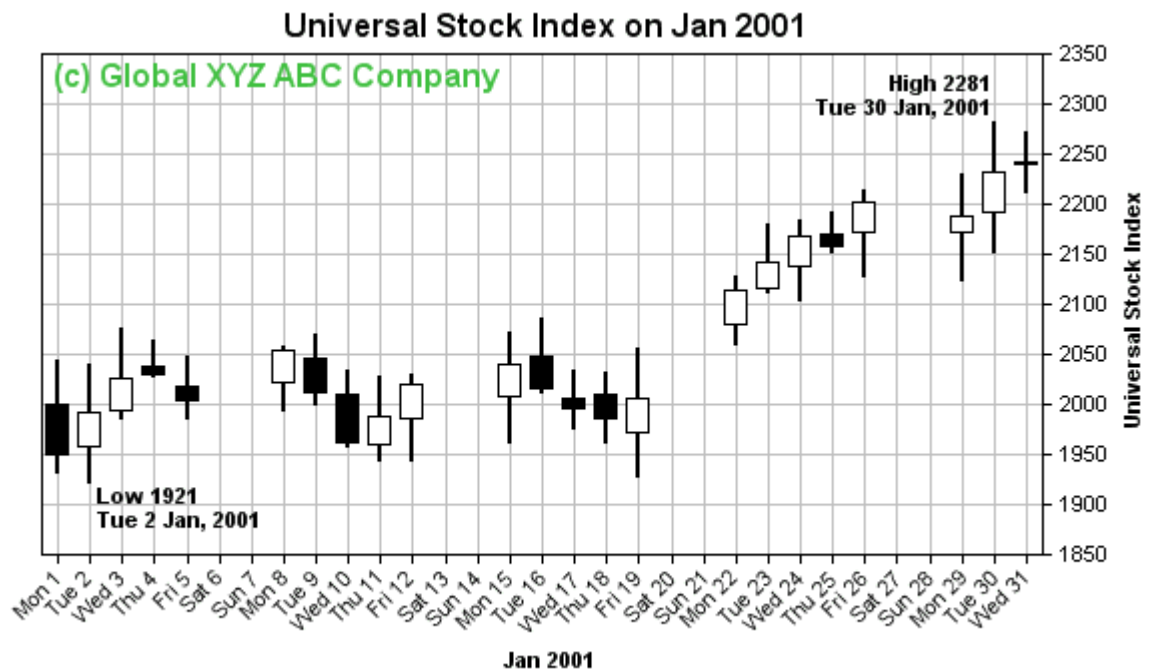
    $align = TopRight;
} else if ($lowPos < 5) {
    $align = TopLeft;
}

#Add the custom low label at the low position
$customDataLabelObj = $layer->addCustomDataLabel(0, $lowPos,
    "Low {low}\n{xLabel} Jan, 2001", "arialbd.ttf");
$customDataLabelObj->setAlignment($align);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## Candle Stick Chart



This example is similar to the previous High-Low-Open-Close chart example, except that candle sticks are used to show the high, low, open and close values.

- Add a CandleStick layer to the chart using the [addCandleStickLayer](#) method.
- Add custom text to the chart using the [addText](#) method. Note the green “(c) Global XYZ ABC Company” text inside the plot area.
- Add custom data labels using the [addCustomDataLabel](#) method. Note the “High 2281” and “Low 1921” texts in this chart.



- Reserve spaces on the top and bottom of the chart for putting custom texts using the [setAutoScale](#) method
- Use the special constant [NoValue](#) to denote that some points have no value. Note that in the above chart, Saturday and Sunday data points have no value.
- Draw the y-axis on the right using the [setYAxisOnRight](#) method.

Draw the x-axis labels at 45 degrees using the [setFontAngle](#) method.

(The following program is available as “phpdemo/candlestick.php”).

```
<?php
include("phpchartdir.php");

#
#Sample data for the HLOC chart. Represents the high, low, open and close
#values for 31 days
#
$nil = NoValue;

$highData = array(2043, 2039, 2076, 2064, 2048, $nil, $nil, 2058, 2070,
    2033, 2027, 2029, $nil, $nil, 2071, 2085, 2034, 2031, 2056, $nil,
    $nil, 2128, 2180, 2183, 2192, 2213, $nil, $nil, 2230, 2281, 2272);

$lowData = array(1931, 1921, 1985, 2028, 1986, $nil, $nil, 1994, 1999,
    1958, 1943, 1944, $nil, $nil, 1962, 2011, 1975, 1962, 1928, $nil,
    $nil, 2059, 2112, 2103, 2151, 2127, $nil, $nil, 2123, 2152, 2212);

$openData = array(2000, 1957, 1993, 2037, 2018, $nil, $nil, 2021, 2045,
    2009, 1959, 1985, $nil, $nil, 2008, 2048, 2006, 2010, 1971, $nil,
    $nil, 2080, 2116, 2137, 2170, 2172, $nil, $nil, 2171, 2191, 2240);

$closeData = array(1950, 1991, 2026, 2029, 2004, $nil, $nil, 2053, 2011,
    1962, 1987, 2019, $nil, $nil, 2040, 2016, 1996, 1985, 2006, $nil,
    $nil, 2113, 2142, 2167, 2158, 2201, $nil, $nil, 2188, 2231, 2242);

#The labels for the HLOC chart
$labels = array("Mon 1", "Tue 2", "Wed 3", "Thu 4", "Fri 5", "Sat 6",
    "Sun 7", "Mon 8", "Tue 9", "Wed 10", "Thu 11", "Fri 12", "Sat 13",
    "Sun 14", "Mon 15", "Tue 16", "Wed 17", "Thu 18", "Fri 19", "Sat 20",
    "Sun 21", "Mon 22", "Tue 23", "Wed 24", "Thu 25", "Fri 26", "Sat 27",
    "Sun 28", "Mon 29", "Tue 30", "Wed 31");

#Create a XYChart object of size 600 x 350 pixels
$c = new XYChart(600, 350);

#Set the plotarea at (50, 25) and of size 500 x 250 pixels. Enable both the
#horizontal and vertical grids by setting their colors to grey (0xc0c0c0)
$plotAreaObj = $c->setPlotArea(50, 25, 500, 250);
$plotAreaObj->setGridColor(0xc0c0c0, 0xc0c0c0);

#Add a title to the chart
$c->addTitle("Universal Stock Index on Jan 2001");
```

```

#Add a custom text at (50, 25) (the upper left corner of the plotarea). Use
#12 pts Arial Bold/pale green (0x40c040) as the font.
$c->addText(50, 25, "(c) Global XYZ ABC Company", "arialbd.ttf", 12,
    0x40c040);

#Add a title to the x axis
$c->xAxis->setTitle("Jan 2001");

#Set the labels on the x axis. Rotate the labels by 45 degrees.
$labelsObj = $c->xAxis->setLabels($labels);
$labelsObj->setFontAngle(45);

#Add a title to the y axis
$c->yAxis->setTitle("Universal Stock Index");

#Draw the y axis on the right hand side of the plot area
$c->setYAxisOnRight(true);

#Reserve 10% margin at the top and bottom of the plot area during
#auto-scaling. This is to leave space for the high and low data labels.
$c->yAxis->setAutoScale(0.1, 0.1);

#Add a CandleStick layer to the chart
$layer = $c->addCandleStickLayer($highData, $lowData, $openData, $closeData
    );

#Set the line width to 2 pixels
$layer->setLineWidth(2);

#
#Now we add the "High" and "Low" text labels. We first find out which are
#the highest and lowest positions.
#
$highPos = 0;
$lowPos = 0;
for($i = 0; $i < count($highData); ++$i) {
    if ($highData[$i] != NoValue & $highData[$i] > $highData[$highPos]) {
        $highPos = $i;
    }
    if ($lowData[$i] != NoValue & $lowData[$i] < $lowData[$lowPos]) {
        $lowPos = $i;
    }
}

#By default, we put text at the center position. If the data point is too
#close to the right or left border of the plot area, we align the text to
#the right and left to avoid the text overflows outside the plot area
$align = BottomCenter;
if ($highPos > 25) {
    $align = BottomRight;
} else if ($highPos < 5) {
    $align = BottomLeft;
}

#Add the custom high label at the high position
$customDataLabelObj = $layer->addCustomDataLabel(0, $highPos,

```

```

    "High {high}\n{xLabel} Jan, 2001", "arialbd.ttf");
$customDataLabelObj->setAlignment($align);

#Similarly, we compute the alignment for the low label based on its x
#position.
$align = TopCenter;
if ($lowPos > 25) {
    $align = TopRight;
} else if ($lowPos < 5) {
    $align = TopLeft;
}

#Add the custom low label at the low position
$customDataLabelObj = $layer->addCustomDataLabel(0, $lowPos,
    "Low {low}\n{xLabel} Jan, 2001", "arialbd.ttf");
$customDataLabelObj->setAlignment($align);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

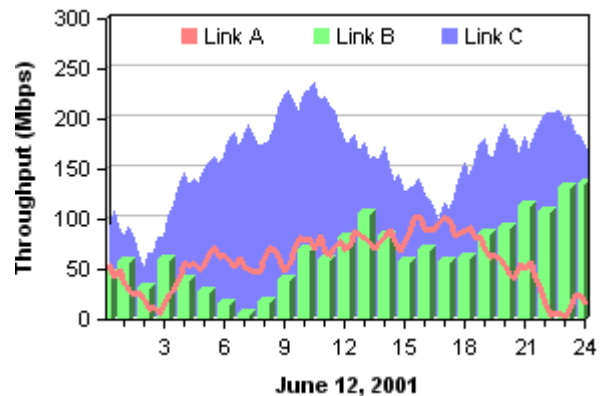
```

## Combination Chart

This example illustrates that you can combine various XYChart layers together to create combination charts of your choice.

This example employs a line layer at the front, a bar layer at the middle and an area layer at the back. Each layer can be either flat or 3D and can have different depths.

Note that in this chart, the resolutions of the data points of the chart layers are different. The chart shows data in a 24-hour period. The line and area layers have data points every 15 minutes, while the bar layer has data points every hour.



(The following program is available as “phpdemo/combo.php”.)

```

<?php
include("phpchartdir.php");

#
# We use a random number generator to simulate collecting data for 24
# hours. The first two data sets have data points every 15 minutes. The
# third data set has data points very one hour.
#
$noOfPoints = 97;
$data0 = array_pad(array(), $noOfPoints, 0);
$data1 = array_pad(array(), $noOfPoints, 0);

```

```

$data2 = array_pad(array(), $noOfPoints, 0);
$labels = array_pad(array(), $noOfPoints, "");

srand(1);
$data0[0] = 50;
$data1[0] = rand() / getrandmax() * 20 + 40;
$data2[0] = rand() / getrandmax() * 20 + 40;
$labels[0] = "0";
for($i = 1; $i < $noOfPoints; ++$i) {
    $data0[$i] = abs($data0[$i - 1] + rand() / getrandmax() * 30 - 15);
    $data1[$i] = abs($data1[$i - 1] + rand() / getrandmax() * 40 - 20);

    #data2 only has data once per hour (once every 4 points)
    if ($i % 4 == 0) {
        $data2[$i] = abs($data2[$i - 4] + rand() / getrandmax() * 60 - 30);
    } else {
        $data2[$i] = NoValue;
    }

    if ($i % 12 == 0) {
        $labels[$i] = $i / 4;
    } else if ($i % 4 == 0) {
        $labels[$i] = "-";
    }
}

#
#Now we obtain the data into arrays, we can start to draw the chart using
#ChartDirector
#

#Create a XYChart object of size 300 x 210 pixels
$c = new XYChart(300, 210);

#Set the plot area at (45, 10) and of size 240 x 150 pixels
$c->setPlotArea(45, 10, 240, 150);

#Add a legend box at (75, 5) (top of plot area) using 8 pts Arial font. Set
#the background and border to Transparent.
$legendObj = $c->addLegend(75, 5, false, "", 8);
$legendObj->setBackground(Transparent);

#Reserve 10% margin at the top of the plot area during auto-scaling to
#leave space for the legend box
$c->yAxis->setAutoScale(0.1);

#Add a title to the y axis
$c->yAxis->setTitle("Throughput (Mbps)");

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Add a title to the x-axis
$c->xAxis->setTitle("June 12, 2001");

#Use non-indent mode for the axis. In this mode, the first and last bar in
#a bar chart is only drawn in half

```

```

$c->xAxis->setIndent(false);

#Add a line chart layer using red (0xff8080) with a line width of 3 pixels
$lineLayerObj = $c->addLineLayer($data0, 0xff8080, "Link A");
$lineLayerObj->setLineWidth(3);

#Add a 3D bar chart layer with depth of 2 pixels using green (0x80ff80) as
#both the fill color and line color
$barLayer = $c->addBarLayer2(Side, 2);
$dataSetObj = $barLayer->addDataSet($data2, -1, "Link B");
$dataSetObj->setDataColor(0x80ff80, 0x80ff80);

#Because each data point is 15 minutes, but the bar layer has only 1 point
#per hour, so for every 4 data points, only 1 point that has data. We will
#increase the bar width so that it will cover the neighbouring points that
#have no data. This is done by decreasing the bar gap to a negative value.
$barLayer->setBarGap(-2);

#Add an area chart layer using blue (0x8080ff) for area and line colors
$areaLayerObj = $c->addAreaLayer();
$dataSetObj = $areaLayerObj->addDataSet($data1, -1, "Link C");
$dataSetObj->setDataColor(0x8080ff, 0x8080ff);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

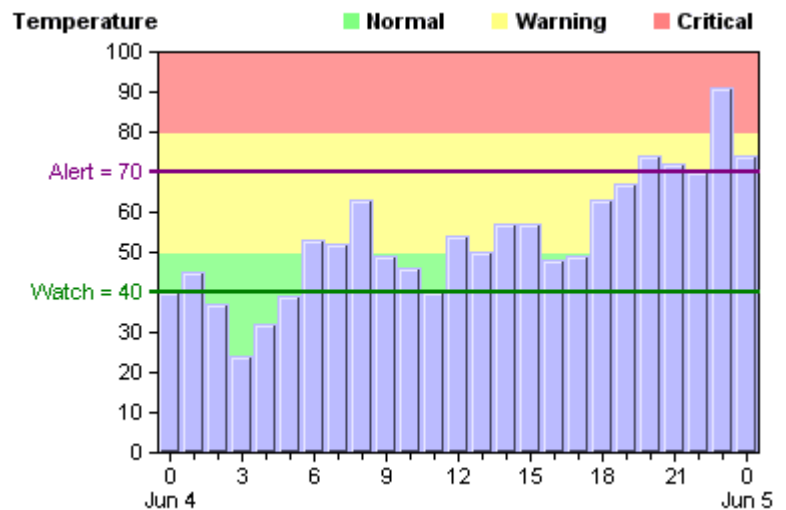
```

## Marks and Zones

This example illustrates the marks and zones features of ChartDirector.

A “mark” is a line drawn in the plot area. The purple “Alert” line and the green “Watch” line on the chart are mark lines. It is added to the chart using the [addMark](#) method.

Mark lines can be horizontal or vertical. They can be drawn on the front of the plot area, or at the back (that is, like grid lines).



A “zone” is a color band on the back of the plot area. In this example, the red, yellow and green areas are zones. They are added to the chart using the [addZone](#) method. Like mark lines, zones can be horizontal or vertical.

This example also illustrates how you could add custom legends using the [addKey](#) method of the `LegendBox` object. In this example, the `addKey` method is used to add the names of the zones to the legend box (instead of using the names of the data set).

(The following program is available as “[phpdemo/markzone.php](#)”.)

```
<?php
include("phpchartdir.php");

#The data for the chart
$data = array(40, 45, 37, 24, 32, 39, 53, 52, 63, 49, 46, 40, 54, 50, 57,
    57, 48, 49, 63, 67, 74, 72, 70, 91, 74);
$labels = array("0\nJun 4", "-", "-", "3", "-", "-", "6", "-", "-", "9",
    "-", "-", "12", "-", "-", "15", "-", "-", "18", "-", "-", "21", "-",
    "-", "0\nJun 5");

#Create a XYChart object of size 400 x 270 pixels
$c = new XYChart(400, 270);

#Set the plotarea at (80, 60) and of size 300 x 200 pixels. Turn off the
#grid lines by setting their colors to Transparent.
$plotAreaObj = $c->setPlotArea(80, 28, 300, 200);
$plotAreaObj->setGridColor(Transparent);

#Add a title to the y axis
$textbox = $c->yAxis->setTitle("Temperature");

#Set the y axis title upright (font angle = 0)
$textbox->setFontAngle(0);

#Put the y axis title on top of the axis
$textbox->setAlignment(Top);

#Move the y axis title by (-36, 8) (36 pixels left, 8 pixels up)
$textbox->setPos(-36, 8);

#Add green (0x99ff99), yellow (0xffff99) and red (0xff9999) zones to the y
#axis to represent the ranges 0 - 50, 50 - 80, and 80 - 100.
$c->yAxis->addZone(0, 50, 0x99ff99);
$c->yAxis->addZone(50, 80, 0xffff99);
$c->yAxis->addZone(80, 100, 0xff9999);

#Add a purple (0x800080) mark at y = 70 using a line width of 2.
$markObj = $c->yAxis->addMark(70, 0x800080, "Alert = 70");
$markObj->setLineWidth(2);

#Add a green (0x008000) mark at y = 40 using a line width of 2.
$markObj = $c->yAxis->addMark(40, 0x008000, "Watch = 40");
$markObj->setLineWidth(2);

#Add a legend box at (165, 0) (top right of the chart) using 8 pts Arial
#font. and horizontal layout.
$legend = $c->addLegend(165, 0, false, "arialbd.ttf", 8);

#Disable the legend box boundary by setting the colors to Transparent
$legend->setBackground(Transparent, Transparent);
```

```

#Add 3 custom entries to the legend box to represent the 3 zones
$legend->addKey("Normal", 0x80ff80);
$legend->addKey("Warning", 0xffff80);
$legend->addKey("Critical", 0xff8080);

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Add a 3D bar layer with the given data
$layer = $c->addBarLayer($data, 0xbbbbff);

#Set the bar gap to 0 so that the bars are packed tightly
$layer->setBarGap(0);

#Set the border color of the bars same as the fill color. Use a 1 pixel 3D
#border effect.
$layer->setBorderColor(SameAsMainColor, 1);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

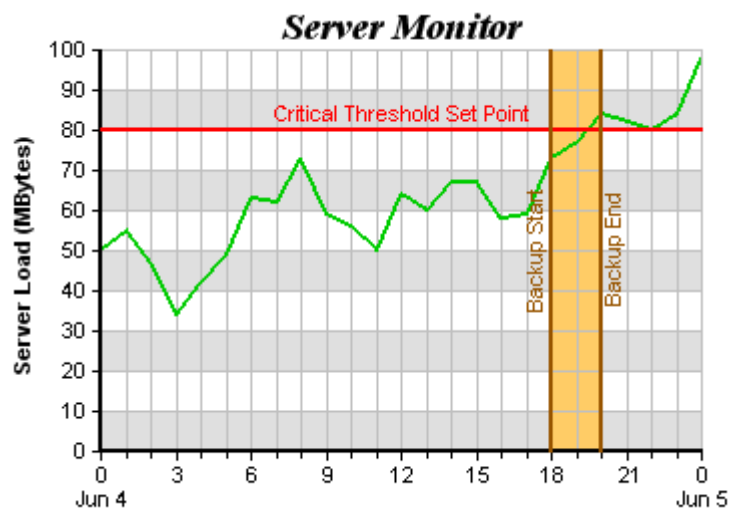
```

## Marks and Zones (2)

Like the last example, this example illustrates the marks and zones features of ChartDirector.

The chart in the example consists of three mark lines (one horizontal and two vertical) and one vertical zone.

This example demonstrates how to position the mark labels. Instead of positioning them as axis labels, in this example the mark labels are drawn along the mark line.



(The following program is available as “phpdemo/marks.php”.)

```

<?php
include("phpchartdir.php");

#The data for the chart
$data = array(50, 55, 47, 34, 42, 49, 63, 62, 73, 59, 56, 50, 64, 60, 67,
    67, 58, 59, 73, 77, 84, 82, 80, 84, 98);

#The labels for the chart. Note the "-" means a minor tick.
$labels = array("0\nJun 4", "-", "-", "3", "-", "-", "6", "-", "-", "9",

```

```

    "-", "-", "12", "-", "-", "15", "-", "-", "18", "-", "-", "21", "-",
    "-", "0\nJun 5");

#Create a XYChart object of size 400 x 270 pixels
$c = new XYChart(400, 270);

#Set the plotarea at (80, 25) and of size 300 x 200 pixels. Use alternate
#color background (0xe0e0e0) and (0xffffffff). Set border and grid colors to
#grey (0xc0c0c0).
$c->setPlotArea(50, 25, 300, 200, 0xe0e0e0, 0xffffffff, 0xc0c0c0, 0xc0c0c0,
    0xc0c0c0);

#Add a title to the chart using 14 pts Times Bold Italic font
$c->addTitle("Server Monitor", "timesbi.ttf", 14);

#Add a title to the y axis
$c->yAxis->setTitle("Server Load (MBytes)");

#Set the y axis width to 2 pixels
$c->yAxis->setWidth(2);

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Set the x axis width to 2 pixels
$c->xAxis->setWidth(2);

#Add a horizontal red (0x800080) mark line at y = 80
$yMark = $c->yAxis->addMark(80, 0xff0000, "Critical Threshold Set Point");

#Set the mark line width to 2 pixels
$yMark->setLineWidth(2);

#Put the mark label at the top center of the mark line
$yMark->setAlignment(TopCenter);

#Add an orange (0xffcc66) zone from x = 18 to x = 20
$c->xAxis->addZone(18, 20, 0xffcc66);

#Add a vertical brown (0x995500) mark line at x = 18
$xMark1 = $c->xAxis->addMark(18, 0x995500, "Backup Start");

#Set the mark line width to 2 pixels
$xMark1->setLineWidth(2);

#Put the mark label at the left of the mark line
$xMark1->setAlignment(Left);

#Rotate the mark label by 90 degrees so it draws vertically
$xMark1->setFontAngle(90);

#Add a vertical brown (0x995500) mark line at x = 20
$xMark2 = $c->xAxis->addMark(20, 0x995500, "Backup End");

#Set the mark line width to 2 pixels
$xMark2->setLineWidth(2);

```



```
#Put the mark label at the right of the mark line
$xMark2->setAlignment(Right);

#Rotate the mark label by 90 degrees so it draws vertically
$xMark2->setFontAngle(90);

#Add a green (0x00cc00) line layer with line width of 2 pixels
$lineLayerObj = $c->addLineLayer($data, 0xcc00);
$lineLayerObj->setLineWidth(2);

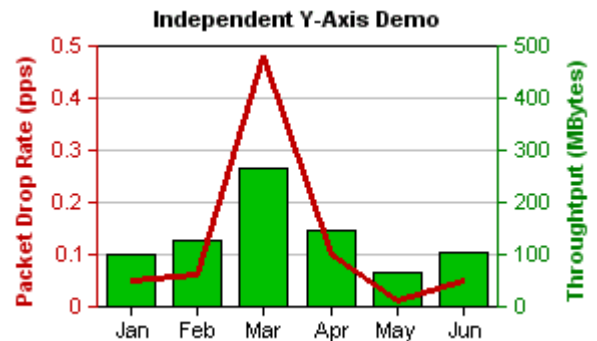
#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## Dual Y-Axis

This example show demonstrates the dual y-axis feature of the ChartDirector.

By default, all data sets used the primary (left) y-axis and the secondary (right) y-axis is unused. The [setUserYAxis2](#) can be used to bind a data set to the secondary (right) y-axis.

Note that the y-axes in this example are of different colors. This is achieved by using the [setColors](#) method of the YAxis object to control the colors of the axis itself, the ticks, the labels and the axis title.



If you want to show only one y-axis on the right (that is, no left axis), there are two methods:

- You can bind all data sets to the secondary y-axis
- You can use default binding (that is, all data sets bind to the primary y-axis), and then use the [setYAxisOnRight](#) to draw the primary y-axis on the right side (and therefore the secondary-axis, if used, will be drawn on the left side).

(The following program is available as “phpdemo/dualyaxis.php”).

```
<?php
include("phpchartdir.php");

#The data for the chart
$data0 = array(0.05, 0.06, 0.48, 0.1, 0.01, 0.05);
$data1 = array(100, 125, 265, 147, 67, 105);
$labels = array("Jan", "Feb", "Mar", "Apr", "May", "Jun");

#Create a XYChart object of size 300 x 180 pixels
$c = new XYChart(300, 180);
```

```

#Set the plot area at (50, 20) and of size 200 x 130 pixels
$c->setPlotArea(50, 20, 200, 130);

#Add a title to the chart using 8 pts Arial Bold font
$c->addTitle("Independent Y-Axis Demo", "arialbd.ttf", 8);

#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

#Add a title to the primary (left) y axis
$c->yAxis->setTitle("Packet Drop Rate (pps)");

#Set the axis, label and title colors for the primary y axis to red
#(0xc00000) to match the first data set
$c->yAxis->setColors(0xc00000, 0xc00000, 0xc00000);

#Add a title to the secondary (right) y axis
$yAxis2Obj = $c->yAxis2();
$yAxis2Obj->setTitle("Throughput (MBytes)");

#set the axis, label and title colors for the primary y axis to green
#(0x008000) to match the second data set
$yAxis2Obj = $c->yAxis2();
$yAxis2Obj->setColors(0x8000, 0x8000, 0x8000);

#Add a line layer to for the first data set using red (0xc00000) color with
#a line width to 3 pixels
$lineLayerObj = $c->addLineLayer($data0, 0xc00000);
$lineLayerObj->setLineWidth(3);

#Add a bar layer to for the second data set using green (0x00C000) color.
#Bind the second data set to the secondary (right) y axis
$barLayerObj = $c->addBarLayer();
$dataSetObj = $barLayerObj->addDataSet($data1, 0xc000);
$dataSetObj->setUseYAxis2();

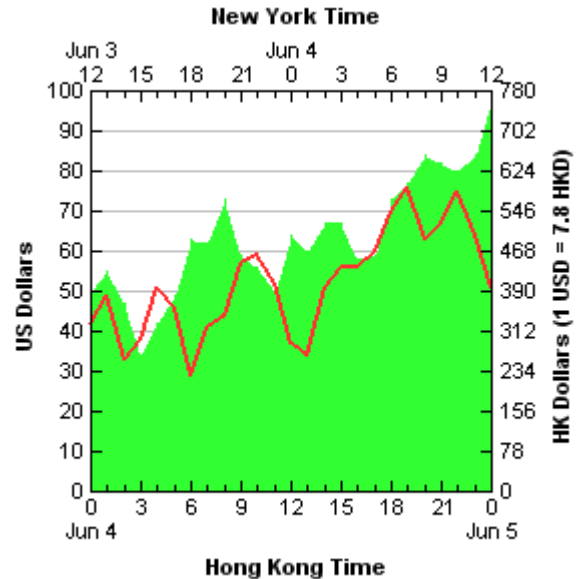
#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

# Dual X-Axis

This example illustrates the following features of the ChartDirector:

- Accessing both the top and bottom x-axes by using the [xAxis](#) method and the [xAxis2](#) method.
- Accessing both the primary and secondary y-axes by using the [yAxis](#) method and [yAxis2](#) method.
- Synchronizing the primary and secondary y-axes by using the [syncYAxis](#) method. In this particular example, the two y-axes scale has a fixed ratio of 1:7.8.
- Use the “-” for an x-axis label to represent a minor tick.
- Use the [setTickLength2](#) method to control the major and minor tick length, and their direction (internal to the chart, or external to the chart).



(The following program is available as “phpdemo/dualxaxis.php”).

```
<?php
include("phpchartdir.php");

#The data for the chart
$data0 = array(42, 49, 33, 38, 51, 46, 29, 41, 44, 57, 59, 52, 37, 34, 51,
    56, 56, 60, 70, 76, 63, 67, 75, 64, 51);
$data1 = array(50, 55, 47, 34, 42, 49, 63, 62, 73, 59, 56, 50, 64, 60, 67,
    67, 58, 59, 73, 77, 84, 82, 80, 84, 98);

#The labels for the bottom x axis. Note the "-" means a minor tick.
$label0 = array("0\nJun 4", "-", "-", "3", "-", "-", "6", "-", "-", "9",
    "-", "-", "12", "-", "-", "15", "-", "-", "18", "-", "-", "21", "-",
    "-", "0\nJun 5");

#The labels for the top x axis. Note that "-" means a minor tick.
$label1 = array("Jun 3\n12", "-", "-", "15", "-", "-", "18", "-", "-",
    "21", "-", "-", "Jun 4\n0", "-", "-", "3", "-", "-", "6", "-", "-",
    "9", "-", "-", "12");

#Create a XYChart object of size 310 x 310 pixels
$c = new XYChart(310, 310);

#Set the plotarea at (50, 50) and of size 200 x 200 pixels
$c->setPlotArea(50, 50, 200, 200);

#Add a title to the primary (left) y axis
$c->yAxis->setTitle("US Dollars");
```

```

#Set the tick length to -4 pixels (-ve means ticks inside the plot area)
$c->yAxis->setTickLength(-4);

#Add a title to the secondary (right) y axis
$yAxis2Obj = $c->yAxis2();
$yAxis2Obj->setTitle("HK Dollars (1 USD = 7.8 HKD)");

#Set the tick length to -4 pixels (-ve means ticks inside the plot area)
$yAxis2Obj = $c->yAxis2();
$yAxis2Obj->setTickLength(-4);

#Synchronize the y-axis such that y2 = 7.8 x y1
$c->syncYAxis(7.8);

#Add a title to the bottom x axis
$c->xAxis->setTitle("Hong Kong Time");

#Set the x axis labels using the given labels
$c->xAxis->setLabels($label0);

#Set the major tick length to -4 pixels and minor tick length to -2 pixels
#(-ve means ticks inside the plot area)
$c->xAxis->setTickLength(-4, -2);

#Add a title to the top x-axis
$xAxis2Obj = $c->xAxis2();
$xAxis2Obj->setTitle("New York Time");

#Set the x-axis labels using the given labels
$xAxis2Obj = $c->xAxis2();
$xAxis2Obj->setLabels($label1);

#Set the major tick length to -4 pixels and minor tick length to -2 pixels
#(-ve means ticks inside the plot area)
$xAxis2Obj = $c->xAxis2();
$xAxis2Obj->setTickLength(-4, -2);

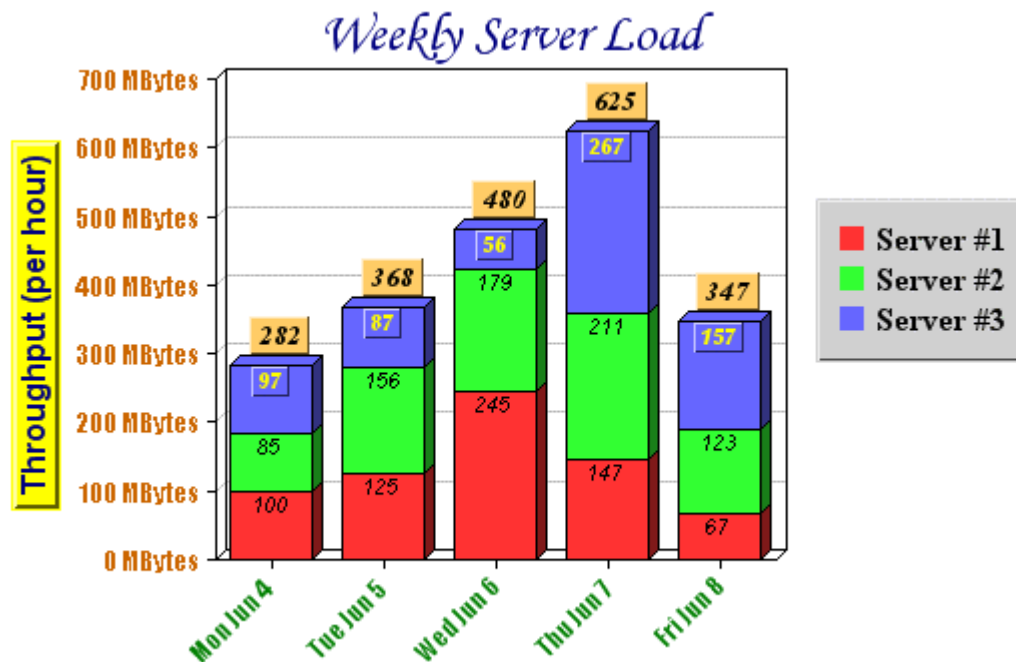
#Add a line layer to the chart with a line width of 2 pixels
$lineLayerObj = $c->addLineLayer($data0, -1, "Server Load");
$lineLayerObj->setLineWidth(2);

#Add an area layer to the chart with no area boundary line
$areaLayerObj = $c->addAreaLayer($data1, -1, "Transaction");
$areaLayerObj->setLineWidth(0);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## Text Style and Colors



This example illustrates you could control the fonts, colors, rotation angles, background box, etc., for most of the text objects in ChartDirector. Note the chart title, y-axis title, y-axis labels, x-axis labels, legend keys, and the data labels on the bars. They all use different fonts and colors, and have different background and 3D border effects. The y-axis labels are appended with “Mbytes” as the unit. The x-axis labels are rotated 45 degrees.

(The following program is available as “phpdemo/fontxy.php”).

```
<?php
include("phpchartdir.php");

#The data for the chart
$data0 = array(100, 125, 245, 147, 67);
$data1 = array(85, 156, 179, 211, 123);
$data2 = array(97, 87, 56, 267, 157);
$labels = array("Mon Jun 4", "Tue Jun 5", "Wed Jun 6", "Thu Jun 7",
    "Fri Jun 8");

#Create a XYChart object of size 540 x 350 pixels
$c = new XYChart(540, 350);

#Set the plot area to start at (120, 40) and of size 280 x 240 pixels
$c->setPlotArea(120, 40, 280, 240);

#Add a title to the chart using 20 pts Monotype Corsiva (mtcorsva.ttf) font
#and using a deep blue color (0x000080)
$c->addTitle("Weekly Server Load", "mtcorsva.ttf", 20, 0x80);

#Add a legend box at (420, 100) (right of plot area) using 12 pts Times
#Bold font. Sets the background of the legend box to light grey 0xd0d0d0
```

```

#with a 1 pixel 3D border.
$legendObj = $c->addLegend(420, 100, true, "timesbd.ttf", 12);
$legendObj->setBackground(0xd0d0d0, 0xd0d0d0, 1);

#Add a title to the y-axis using 12 pts Arial Bold/deep blue (0x000080)
#font. Set the background to yellow (0xffff00) with a 2 pixel 3D border.
$titleObj = $c->yAxis->setTitle("Throughput (per hour)", "arialbd.ttf",
    12, 0x80);
$titleObj->setBackground(0xffff00, 0xffff00, 2);

#Use 10 pts Impact/orange (0xcc6600) font for the y axis labels
$c->yAxis->setLabelStyle("impact.ttf", 10, 0xcc6600);

#Set the axis label format to "nnn MBytes"
$c->yAxis->setLabelFormat("{value} MBytes");

#Use 10 pts Impact/green (0x008000) font for the x axis labels. Set the
#label angle to 45 degrees.
$labelStyleObj = $c->xAxis->setLabelStyle("impact.ttf", 10, 0x8000);
$labelStyleObj->setFontAngle(45);

#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

#Add a 3D stack bar layer with a 3D depth of 5 pixels
$layer = $c->addBarLayer2(Stack, 5);

#Use Arial Italic as the default data label font in the bars
$layer->setDataLabelStyle("ariali.ttf");

#Use 10 pts Times Bold Italic (timesbi.ttf) as the aggregate label font.
#Set the background to flesh (0xffcc66) color with a 1 pixel 3D border.
$aggregateLabelStyleObj = $layer->setAggregateLabelStyle("timesbi.ttf", 10)
;
$aggregateLabelStyleObj->setBackground(0xffcc66, Transparent, 1);

#Add the first data set to the stacked bar layer
$layer->addDataSet($data0, -1, "Server #1");

#Add the second data set to the stacked bar layer
$layer->addDataSet($data1, -1, "Server #2");

#Add the third data set to the stacked bar layer, and set its data label
#font to Arial Bold Italic.
$dataSetObj = $layer->addDataSet($data2, -1, "Server #3");
$textbox = $dataSetObj->setDataLabelStyle("arialbi.ttf");

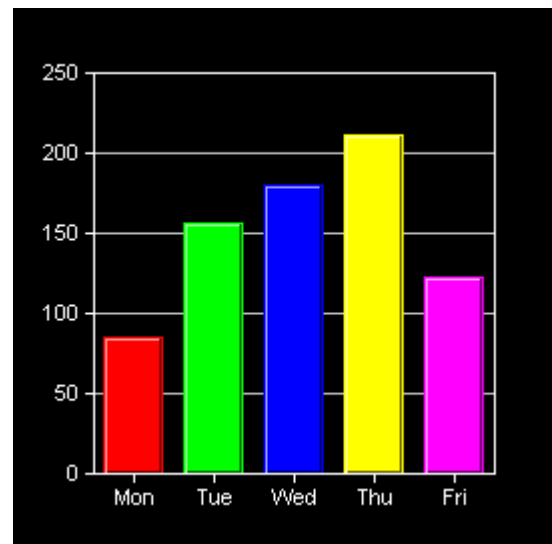
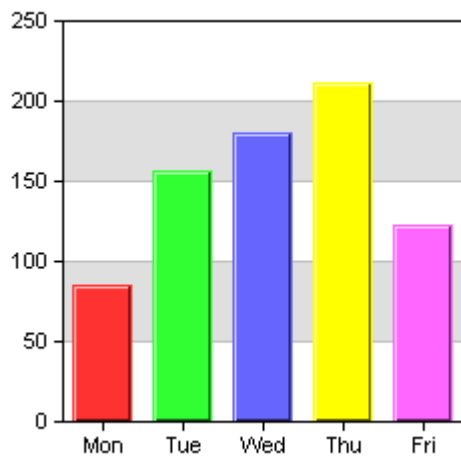
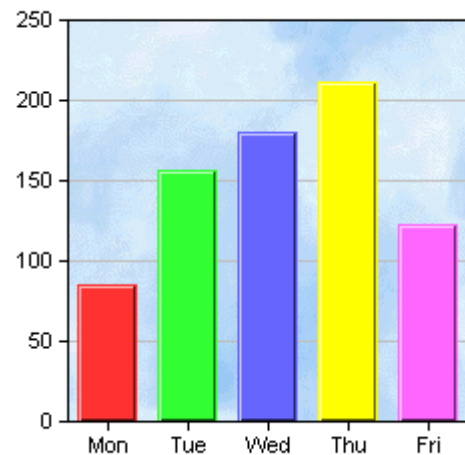
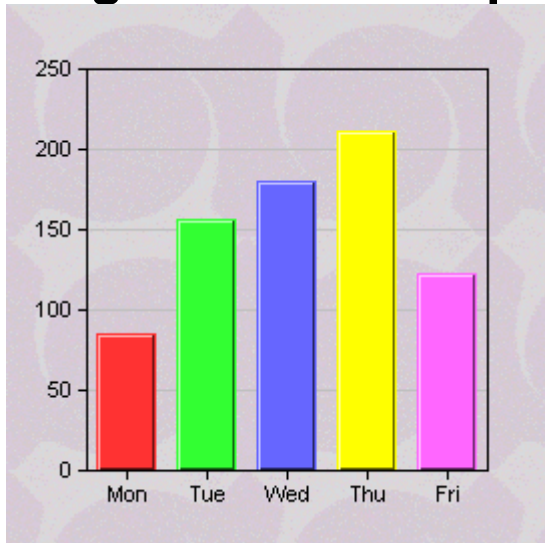
#Set the data label font color for the third data set to yellow (0xffff00)
$textbox->setFontColor(0xffff00);

#Set the data label background color to the same color as the bar segment,
#with a 1 pixel 3D border.
$textbox->setBackground(SameAsMainColor, Transparent, 1);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));

```

## Background and Wallpaper



This example illustrates some of the background effects supported by ChartDirector.

ChartDirector supports the use of an image file as the wallpaper as the entire chart background and/or as the background of the plot area only. The former is by using the [setWallpaper](#) method of the BaseChart object, while the latter is by using the [setBackground2](#) method of the PlotArea object.

In addition to using wallpapers, ChartDirector also supports different kinds of background color. The plot area background can be alternately colored by using the [setBackground](#) method of the PlotArea object. You can also switch to a dark background chart with white lines in one step by changing the color palette to a “whiteOnBlackPalette”. This is by using the [setColors](#) method.

(The following program is available as “phpdemo/background.php”).

```
<?php
include("phpchartdir.php");

#The data for the chart
$data = array(85, 156, 179.5, 211, 123);
$labels = array("Mon", "Tue", "Wed", "Thu", "Fri");

#Create a XYChart object of size 270 x 270 pixels
$c = new XYChart(270, 270);

#Set the plot area at (40, 32) and of size 200 x 200 pixels
$plotarea = $c->setPlotArea(40, 32, 200, 200);

#Set the background style based on the input parameter
if ($HTTP_GET_VARS["img"] == "0") {
    #Has wallpaper image
    $c->setWallpaper(dirname($PATH_TRANSLATED)."/tile.gif");
} else if ($HTTP_GET_VARS["img"] == "1") {
    #Use a background image as the plot area background
    $plotarea->setBackground2(dirname($PATH_TRANSLATED)."/bg.png");
} else if ($HTTP_GET_VARS["img"] == "2") {
    #Use white (0xffffffff) and grey (0xe0e0e0) as two alternate plotarea
    #background colors
    $plotarea->setBackground(0xffffffff, 0xe0e0e0);
} else {
    #Use a dark background palette
    $c->setColors($whiteOnBlackPalette);
}

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

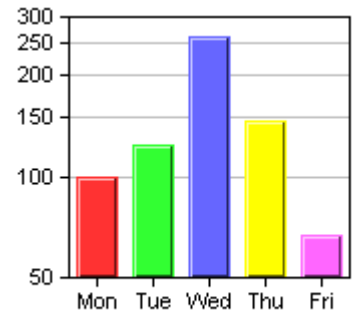
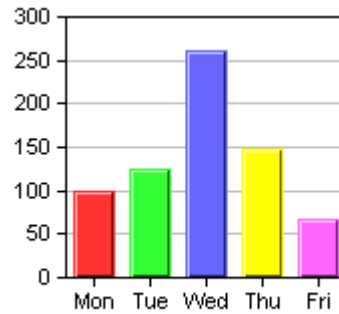
#Add a color bar layer using the given data. Use a 1 pixel 3D border for
#the bars.
$barLayer3Obj = $c->addBarLayer3($data);
$barLayer3Obj->setBorderColor(-1, 1);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```



## Log Scale Axis

This example illustrates the log scale axis feature of the ChartDirector. On the right are two charts. One of them is drawn using the default y-axis, which is linearly scaled. The other is drawn using a log scale y-axis by calling the [setLogScale](#) method of the YAxis object.



(The following program is available as “phpdemo/logaxis.php”).

```
<?php
include("phpchartdir.php");

#The data for the chart
$data = array(100, 125, 260, 147, 67);
$labels = array("Mon", "Tue", "Wed", "Thu", "Fri");

#Create a XYChart object of size 200 x 180 pixels
$c = new XYChart(200, 180);

#Set the plot area at (30, 10) and of size 140 x 130 pixels
$c->setPlotArea(30, 10, 140, 130);

#Use log scale axis if required
if ($HTTP_GET_VARS["img"] == "1") {
    $c->yAxis->setLogScale();
}

#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Add a color bar layer using the given data. Use a 1 pixel 3D border for
#the bars.
$barLayer3Obj = $c->addBarLayer3($data);
$barLayer3Obj->setBorderColor(-1, 1);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

## Y-Axis Scaling

This example illustrates how you could control the scaling of the y-axis.

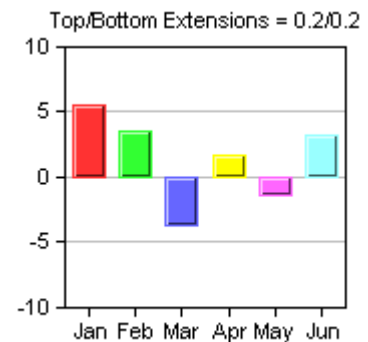
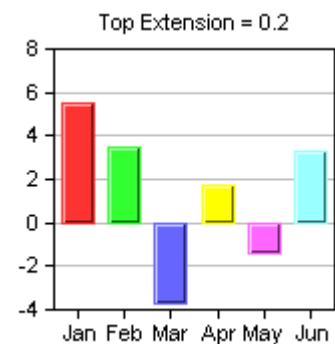
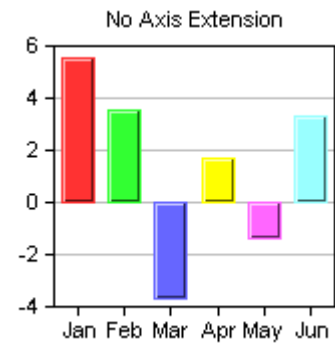
By default, the y-axis is auto-scaled. The `ChartDirector` will automatically determine the most suitable scaling for the y-axis by taking into consideration the maximum and minimum values of the data. It will attempt to ensure all the axis ticks are whole numbers, and to include the zero point if the scale looks reasonable.

The first chart in this example employs standard auto-scaling.

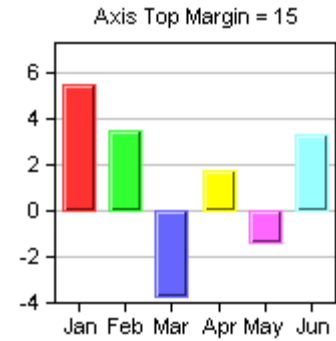
In many cases, it is desirable to leave some spaces at the top and/or bottom of the plot area. For example, you may want to add some custom text or legend box at those locations, or you may simply think it looks better this way. In these cases, the [`setAutoScale`](#) method can be used to inform the auto-scaling algorithm that some spaces should reserved on the top and/or bottom. The auto-scaling algorithm will then guarantee it reserve at least the required spaces. Note that it may reserve more space than specified in order to make the scaling looks nice (e.g. it may do so to ensure all the axis ticks are whole numbers).

The second chart illustrates what the chart looks like when 20% of the spaces on the top are reserved by using `setAutoScale(0.2)`.

The third chart illustrates what the chart looks like when 20% of the spaces on the top and 20% of the space on the bottom are reserved by using [`setAutoScale\(0.2, 0.2\)`](#).

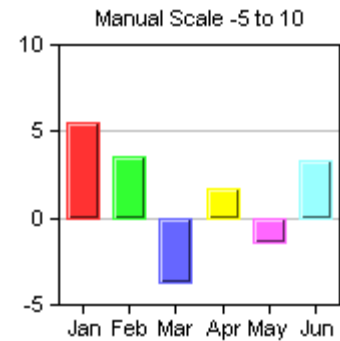


An alternative way to reserve space on the top of the plot area is to use the [setTopMargin](#) method. With this method, a segment on the top of the y-axis is excluded from scaling. There are no ticks and no scaling there. The chart on the right illustrates this feature. Note that the top of the y-axis has no scaling there.



Although auto-scaling is convenient, in many cases manual scaling may be more preferable. For example, in many percentage charts, you may want the percentage scale to be from 0 – 100 irrespective of the actual data range.

The [setLinearScale](#) and [setLogScale](#) method in the ChartDirector API can be used to specify manual scaling. The chart on the right illustrates an axis manually scaled to range from –5 to 10, with a tick every 5 units.



(The following program is available as “[phpdemo/axisscale.php](#)”.)

```
<?php
include("phpchartdir.php");

#The data for the chart
$data = array(5.5, 3.5, -3.7, 1.7, -1.4, 3.3);
$labels = array("Jan", "Feb", "Mar", "Apr", "May", "Jun");

#Create a XYChart object of size 200 x 180 pixels
$c = new XYChart(200, 180);

#Set the plot area at (30, 20) and of size 140 x 130 pixels
$c->setPlotArea(30, 20, 140, 130);

#Configure the axis as according to the input parameter
if ($HTTP_GET_VARS["img"] == "0") {
    $c->addTitle("No Axis Extension", "arial.ttf", 8);
} else if ($HTTP_GET_VARS["img"] == "1") {
    $c->addTitle("Top Extension = 0.2", "arial.ttf", 8);
    #Reserve 20% margin at top of plot area when auto-scaling
    $c->yAxis->setAutoScale(0.2);
} else if ($HTTP_GET_VARS["img"] == "2") {
    $c->addTitle("Top/Bottom Extensions = 0.2/0.2", "arial.ttf", 8);
    #Reserve 20% margin at top and bottom of plot area when auto-scaling
    $c->yAxis->setAutoScale(0.2, 0.2);
} else if ($HTTP_GET_VARS["img"] == "3") {
    $c->addTitle("Axis Top Margin = 15", "arial.ttf", 8);
    #Reserve 15 pixels at top of plot area
    $c->yAxis->setTopMargin(15);
```

```

} else {
  $c->addTitle("Manual Scale -5 to 10", "arial.ttf", 8);
  #Set the y axis to scale from -5 to 10, with ticks every 5 units
  $c->yAxis->setLinearScale(-5, 10, 5);
}

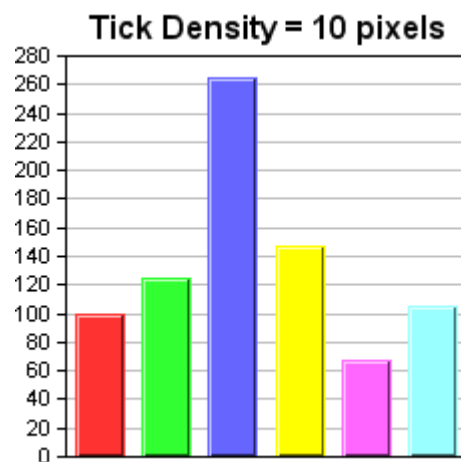
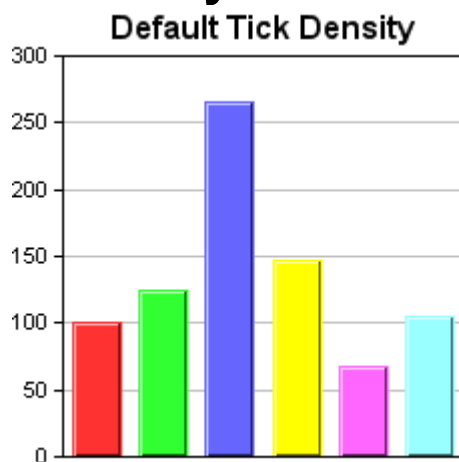
#Set the labels on the x axis
$c->xAxis->setLabels($labels);

#Add a color bar layer using the given data. Use a 1 pixel 3D border for
#the bars.
$barLayer3Obj = $c->addBarLayer3($data);
$barLayer3Obj->setBorderColor(-1, 1);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## Tick Density



This example illustrates how to control the axis tick density during auto-scaling.

In manual scaling, you can directly control the axis tick density using the [setLinearScale](#) or [setLogScale2](#) method.

In auto-scaling, you can indirectly control the tick density by using the [setTickDensity](#) method to specify a preferred tick spacing.

The auto-scaling algorithm will attempt to use a tick spacing that matches the requested tick spacing. The resulting tick spacing may be larger than requested because the auto-scaling algorithm also needs to ensure that the ticks are whole numbers, and the axis contains an integral number of ticks, etc.

In this example, one of the charts is drawn using the default tick density. The other chart is drawn using a requested tick spacing of 10 pixels. Note that the actual tick spacing is slightly larger than 10 pixels.

Also note that the scaling of the axis has changed because of the tick density. In the first chart the scale is from 0 – 300, while in the second chart the scale is from 0 – 280. The ChartDirector may need to change the scale to meet the tick spacing requirements and other constraints that it may have.

(The following program is available as “phpdemo/ticks.php”.)

```
<?php
include("phpchartdir.php");

#The data for the chart
$data = array(100, 125, 265, 147, 67, 105);

#Create a XYChart object of size 250 x 250 pixels
$c = new XYChart(250, 250);

#Set the plot area at (25, 25) and of size 200 x 200 pixels
$c->setPlotArea(25, 25, 200, 200);

if ($HTTP_GET_VARS["img"] == "1") {
    #High tick density, uses 10 pixels as tick spacing
    $c->addTitle("Tick Density = 10 pixels");
    $c->yAxis->setTickDensity(10);
} else {
    #Normal tick density, just use the default setting
    $c->addTitle("Default Tick Density");
}

#Add a color bar layer using the given data. Use a 1 pixel 3D border for
#the bars.
$barLayer3Obj = $c->addBarLayer3($data);
$barLayer3Obj->setBorderColor(-1, 1);

#output the chart in PNG format
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>
```

---

# Using Data Sources with ChartDirector

The ChartDirector API is designed to accept data values as arrays, which is the most flexible way of accepting data. This flexibility permits ChartDirector to accept data from any kind data source. For example, ChartDirector can accept data as HTTP GET or POST parameters, from a text file, or from a database. All is needed is to read the data values into arrays and pass them to ChartDirector.

This chapter will provide a number of examples on various methods of passing data to ChartDirector.

## Using ChartDirector in Web Pages

As mentioned previous in [Chapter 3 - Getting Started](#), in real application, it is likely the charts will be embedded in a HTML web page. This can be done by using <IMG> tags in the HTML web page to load the chart image. For example:

```
<HTML>
<BODY>

<h1>Hello World!</h1>
<p>Hi, this is my first web page with ChartDirector charts.</p>

<IMG SRC=" http://aaa.bbb.ccc.ddd/phpdemo/simplebar.php">

More HTML elements .....

</BODY>
</HTML>
```

You can include multiple charts in the same page by using multiple <IMG> tags in the HTML page. The HTML page itself can also be a PHP page.

## Strategies in Passing Data to ChartDirector

In the sample codes in previous chapters, all data are hard coded in the sample codes for ease of explanation. In this case, the containing HTML page just needs a simple <IMG> tag to load the chart.

In real life, most probably the data are dynamic. They may come from a database, a text file or as HTTP GET or POST parameters.

Suppose the data come from a database through a database query. There are several options where the query is performed and how the data is passed to ChartDirector.

**1) Perform the database query directly in the ChartDirector PHP page.**

This is the simplest option. Instead of using hard coded data, simply read the data from a database query.

**2) Perform the database query in the containing HTML page (which is also a PHP page), and pass the data to the ChartDirector PHP page as HTTP GET query parameters.**

Although being more complex, this method has the advantage that you can use the results of the database query in the containing HTML page as well. For example, in addition to displaying the data as charts, you can display the data in the containing HTML page as tables. Also, since the database query is done in the containing HTML page, you can display multiple ChartDirector charts with just one database query. (This may be important if your database query is resource intensive.)

**3) Instead of using HTTP GET query parameters to pass the data to the ChartDirector PHP page as “session variables”.**

One limitation of passing data using HTTP GET query parameters is that you cannot pass too much data. Most browsers and web servers cannot support URL of longer than 2048 characters. If there is too much data to pass as HTTP GET query parameters, the URL may become too long.

This limitation could be overcome by using “session variables” for passing parameters. Many web server platforms support “session variables” as a way of maintaining “state” information across multiple HTTP queries for the same user.

You could save the data in a session variable in the containing HTML document, and retrieve the variable in the ChartDirector PHP page. In effect, the data is passed from the containing HTML document to the ChartDirector PHP page.

**4) Perform the database query and create the chart image in the containing HTML page (which is also an PHP page). Save the chart image to a “session variable”. Use an <IMG> tag to load the chart back from the “session variable”.**

In some applications, the chart must be created in the containing HTML page (e.g. when creating clickable charts). However, it is impossible to directly display a chart in HTML. The <IMG> tag must be used. So we need to save the chart to “some place”, and then load the chart back using the <IMG> tag.

One popular method is to save the chart image to a “session variable”. The <IMG> tag invokes a PHP script that simply retrieves the image from the session variable and returns it to the browser.

## Database Sample Code Overview

In the followings, we will present ChartDirector sample code with database access using methods (1), (2) and (4) as described in the last section.

The sample codes in this chapter will be based on a MySQL database. To actually run the sample codes, you need to:

- Verify that your PHP has MySQL extensions installed
- Set up a MySQL server to run on the same machine as the web server
- Create a database called "sample" in your MySQL server
- Create an account with username "test" and password "test" that can be used to access "sample".
- Import the table "sample.sql" (located in the "phpdemo" subdirectory) into the "sample" database

The “sample” database contains only one table called “revenue”. The table stores the monthly revenue data for three business units - Hardware, Software and Services. It contains four columns as follows:

Column Name	Type
TimeStamp	Date/Time
Hardware	Number
Software	Number
Services	Number

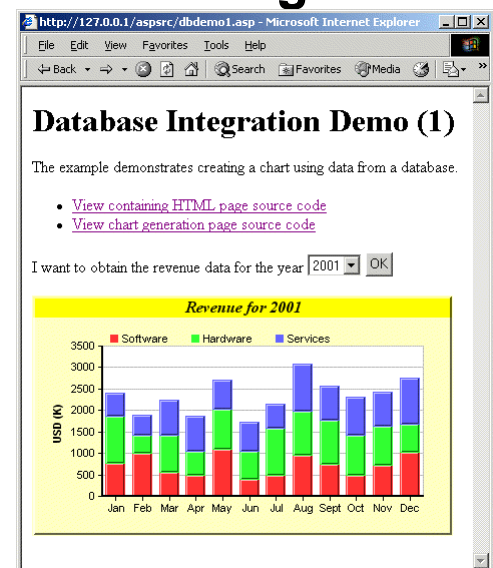
Each record in the database contains the monthly revenue for the business units, where the month is stored in the “TimeStamp” column. The sample database contains records from Jan 1990 to Dec 2001.

## Perform Query in the ChartDirector PHP Page

In this example, we will demonstrate how to create a web page to show the monthly revenue for a given year as shown below.

The user will select a year from a HTML form and then press OK. The web server will then return a web page containing the bar chart for the selected year.

The code for producing the HTML form is listed below. It outputs a drop down select list to allow the user to select a year. Based on the selected year, it uses an <IMG> tag with a ChartDirector PHP page as the image URL to produce the chart for the selected year.



(The following program is available as “phpdemo\dbdemo1.php”).)

```
<html>
```



```

<body>
<h1>Database Integration Demo (1)</h1>
<p>The example demonstrates creating a chart using data from a database.</p>

<ul>
  <li><a href="viewsource.php?file=<?=$HTTP_SERVER_VARS["SCRIPT_NAME"]?>">
    View containing HTML page source code
  </a></li>
  <li><a href="viewsource.php?file=dbdemola.php">
    View chart generation page source code
  </a></li>
</ul>

<form action="<?=$HTTP_SERVER_VARS["SCRIPT_NAME"]?>">
  I want to obtain the revenue data for the year
  <select name="year">
    <option value="1990">1990
    <option value="1991">1991
    <option value="1992">1992
    <option value="1993">1993
    <option value="1994">1994
    <option value="1995">1995
    <option value="1996">1996
    <option value="1997">1997
    <option value="1998">1998
    <option value="1999">1999
    <option value="2000">2000
    <option value="2001">2001
  </select>
  <input type="submit" value="OK">
</form>

<?
$SelectedYear = $HTTP_GET_VARS["year"];
if (!$SelectedYear) $SelectedYear = 2001;
?>

<SCRIPT>
  //make sure the select box displays the current selected year.
  document.forms[0].year.selectedIndex = <?=$SelectedYear - 1990?>;
</SCRIPT>



</body>
</html>

```

As seen from the code above, the actual chart is created by the URL in the <IMG> tag, which is “dbdemola.php”. The “dbdemola.php” is as follows.

(The following program is available as “phpdemo\dbdemola.php”).

```

<?
include("phpchartdir.php");

#

```

```

#Displays the monthly revenue for the selected year. The selected year
#should be passed in as a query parameter called "year"
#
$SelectedYear = $HTTP_GET_VARS["year"];
if (!$SelectedYear) $SelectedYear = 2001;

#
#Create an SQL statement to get the revenues of each month for the
#selected year.
#
$SQLstatement = "Select Month(TimeStamp) - 1, " .
    "Software, Hardware, Services " .
    "From revenue Where Year(TimeStamp)=" . $SelectedYear;

#
#Read in the revenue data into arrays
#
mysql_connect("localhost", "test", "test");
$result = mysql_db_query("sample", $SQLstatement);
$software = array_pad(array(), 12, 0);
$hardware = array_pad(array(), 12, 0);
$services = array_pad(array(), 12, 0);
while ($row = mysql_fetch_row($result)) {
    $software[$row[0]] = $row[1];
    $hardware[$row[0]] = $row[2];
    $services[$row[0]] = $row[3];
}

#
#Now we obtain the data into arrays, we can start to draw the chart
#using ChartDirector
#

#Create a XYChart of size 420 pixels x 240 pixels
$c = new XYChart(420, 240);

#Set the chart background to pale yellow (0xffffc0) with a 2 pixel 3D border
$c->setBackground(0xffffc0, 0xffffc0, 2);

#Set the plotarea at (70, 50) and of size 320 x 150 pixels. Set background
#color to white (0xffffffff). Enable both horizontal and vertical grids by
#setting their colors to light grey (0xc0c0c0)
$c->setPlotArea(70, 50, 320, 150, 0xffffffff, 0xffffffff, 0xc0c0c0, 0xc0c0c0);

#Add a title to the chart
$title = $c->addTitle("Revenue for " . $SelectedYear, "timesbi.ttf");
$title->setBackground(0xffff00);

#Add a legend box at the top of the plotarea
$legend = $c->addLegend(70, 30, 0, "", 8);
$legend->setBackground(Transparent);

#Add a stacked bar chart layer using the supplied data
$layer = $c->addBarLayer2(Stack);
$layer->addDataSet($software, -1, "Software");
$layer->addDataSet($hardware, -1, "Hardware");
$layer->addDataSet($services, -1, "Services");

```

```

$layer->setBorderColor(Transparent, 1);

#Set the x axis labels. In this example, the labels must be Jan - Dec.
$labels = array("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
    "Sept", "Oct", "Nov", "Dec");
$c->xAxis->setLabels($labels);

#Set the x-axis width to 2 pixels
$c->xAxis->setWidth(2);

#Set the y axis title
$c->yAxis->setTitle("USD (K)");

#Set the y-axis width to 2 pixels
$c->yAxis->setWidth(2);

#Output the chart
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

The first part of the above code is a database query using MySQL. The SQL statement is:

```

$SQLstatement = "Select Month(TimeStamp) - 1, " .
    "Software, Hardware, Services " .
    "From revenue Where Year(TimeStamp)=" . $SelectedYear;

```

Note that the first column “Month(TimeStamp)-1” will be a number from 0 – 11, representing Jan – Dec. We can use it as index to the data arrays.

The code then executes the SQL query and read the data into an array using a loop. In the loop, the data are read using:

```

mysql_connect("localhost", "test", "test");
$result = mysql_db_query("sample", $SQLstatement);
$software = array_pad(array(), 12, 0);
$hardware = array_pad(array(), 12, 0);
$services = array_pad(array(), 12, 0);
while ($row = mysql_fetch_row($result)) {
    $software[$row[0]] = $row[1];
    $hardware[$row[0]] = $row[2];
    $services[$row[0]] = $row[3];
}

```

The advantages of using the first column as an “array index” to read the data into arrays are that it does not require the records to be sorted. It also works well in case there are some “missing records” in the database.

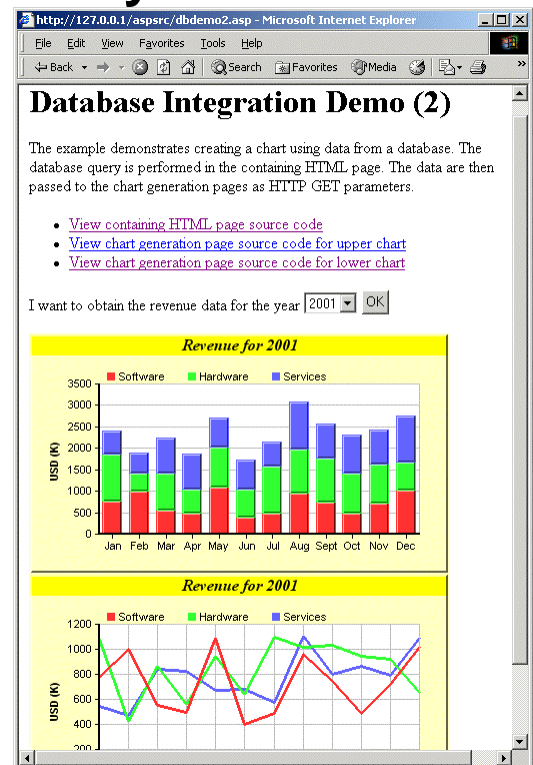
After we read the data into arrays, the second part of the code is to create a stacked bar chart using the given data. This is very similar to the examples in the previous chapters, so it will not explain further here.

# Pass Data to ChartDirector as Query Parameters

In this example, we will introduce a technique where the database query is performed in the containing HTML page, and the data is passed to the ChartDirector PHP page as HTTP GET query parameters. Using this method, it is possible to perform one database query and feed the data to multiple charts.

As in the last example, we allow the user to select the year using a simple HTML form. The web server will then return a web page containing the bar chart and the line chart for the selected year.

The code for producing the HTML form is listed below. It outputs a drop down select list to allow the user to select a year. Based on the selected year, it queries the database and read the data into arrays. The data are serialized into comma delimited strings using the “join” function. Two <IMG> tags are used to invoke two PHP pages for charts generation, where the data are passed in as HTTP GET query parameters using the comma delimited strings.



(The following is available as “phpdemo\dbdemo2.php”).

```
<html>
<body>
<h1>Database Integration Demo (2)</h1>
<p style="width:500px;">This example demonstrates creating a chart using
data from a database. The database query is performed in the containing
HTML page. The data are then passed to the chart generation pages as
HTTP GET parameters.</p>

<ul>
  <li><a href="viewsource.php?file=dbdemo2.php">
    View containing HTML page source code
  </a></li>
  <li><a href="viewsource.php?file=dbdemo2a.php">
    View chart generation page source code for upper chart
  </a></li>
  <li><a href="viewsource.php?file=dbdemo2b.php">
    View chart generation page source code for lower chart
  </a></li>
</ul>

<form action="<?=$HTTP_SERVER_VARS["SCRIPT_NAME"]?>">
  I want to obtain the revenue data for the year
  <select name="year">
    <option value="1990">1990
    <option value="1991">1991
    <option value="1992">1992
```

```

        <option value="1993">1993
        <option value="1994">1994
        <option value="1995">1995
        <option value="1996">1996
        <option value="1997">1997
        <option value="1998">1998
        <option value="1999">1999
        <option value="2000">2000
        <option value="2001">2001
    </select>
    <input type="submit" value="OK">
</form>

<?
#
#Perform the database query to get the required data. The selected year
#should be passed in as a query parameter called "year"
#
$SelectedYear = $_HTTP_GET_VARS["year"];
if (!$SelectedYear) $SelectedYear = 2001;

#
#Create an SQL statement to get the revenues of each month for the
#selected year.
#
$SQLstatement = "Select Month(TimeStamp) - 1, " .
    "Software, Hardware, Services " .
    "From revenue Where Year(TimeStamp)=" . $SelectedYear;

#
#Read in the revenue data into arrays
#
mysql_connect("localhost", "test", "test");
$result = mysql_db_query("sample", $SQLstatement);
$software = array_pad(array(), 12, 0);
$hardware = array_pad(array(), 12, 0);
$services = array_pad(array(), 12, 0);
while ($row = mysql_fetch_row($result)) {
    $software[$row[0]] = $row[1];
    $hardware[$row[0]] = $row[2];
    $services[$row[0]] = $row[3];
}

#Serialize the data into a string to be used as HTTP query parameters
$httpParam = "year=" . $SelectedYear . "&software=".join(",", $software) .
    "&hardware=".join(",", $hardware) . "&services=".join(",", $services);
?>

<SCRIPT>
    //make sure the select box displays the current selected year.
    document.forms[0].year.selectedIndex = <?=$SelectedYear - 1990?>;
</SCRIPT>

<br>


</body>

```

```
</html>
```

The <IMG> tags in the above code invoke “dbdemo2a.php” and “dbdemo2b.php” for charts generation. In “dbdemo2a.php” and “dbdemo2b.php”, the data is retrieved from the HTTP GET query parameters and deserialized back into arrays by using the “split” function. The data are then used to generate the charts.

(The following program is available as “phpdemo\dbdemo2a.php”.)

```
<?
include("phpchartdir.php");

#
#Retrieve the data from the query parameters
#
$SelectedYear = $HTTP_GET_VARS["year"];
if (!$SelectedYear) $SelectedYear = 2001;

$software = split(",", $HTTP_GET_VARS["software"]);
$hardware = split(",", $HTTP_GET_VARS["hardware"]);
$services = split(",", $HTTP_GET_VARS["services"]);

#
#Now we obtain the data into arrays, we can start to draw the chart
#using ChartDirector
#

#Create a XYChart of size 420 pixels x 240 pixels
$c = new XYChart(420, 240);

#Set the chart background to pale yellow (0xffffc0) with a 2 pixel 3D border
$c->setBackground(0xffffc0, 0xffffc0, 2);

#Set the plotarea at (70, 50) and of size 320 x 150 pixels. Set background
#color to white (0xffffffff). Enable both horizontal and vertical grids by
#setting their colors to light grey (0xc0c0c0)
$c->setPlotArea(70, 50, 320, 150, 0xffffffff, 0xffffffff, 0xc0c0c0, 0xc0c0c0);

#Add a title to the chart
$title = $c->addTitle("Revenue for " . $SelectedYear, "timesbi.ttf");
$title->setBackground(0xffff00);

#Add a legend box at the top of the plotarea
$legend = $c->addLegend(70, 30, 0, "", 8);
$legend->setBackground(Transparent);

#Add a stacked bar chart layer using the supplied data
$layer = $c->addBarLayer2(Stack);
$layer->addDataSet($software, -1, "Software");
$layer->addDataSet($hardware, -1, "Hardware");
$layer->addDataSet($services, -1, "Services");
$layer->setBorderColor(Transparent, 1);

#Set the x axis labels. In this example, the labels must be Jan - Dec.
```

```

$labels = array("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
    "Sept", "Oct", "Nov", "Dec");
$c->xAxis->setLabels($labels);

#Set the x-axis width to 2 pixels
$c->xAxis->setWidth(2);

#Set the y axis title
$c->yAxis->setTitle("USD (K)");

#Set the y-axis width to 2 pixels
$c->yAxis->setWidth(2);

#Output the chart
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

(The following program is available as “phpdemo\dbdemo2b.php”.)

```

<?
include("phpchartdir.php");

#
#Retrieve the data from the query parameters
#
$SelectedYear = $_HTTP_GET_VARS["year"];
if (!$SelectedYear) $SelectedYear = 2001;

$software = split(",", $_HTTP_GET_VARS["software"]);
$hardware = split(",", $_HTTP_GET_VARS["hardware"]);
$services = split(",", $_HTTP_GET_VARS["services"]);

#
#Now we obtain the data into arrays, we can start to draw the chart
#using ChartDirector
#

#Create a XYChart of size 420 pixels x 240 pixels
$c = new XYChart(420, 240);

#Set the chart background to pale yellow (0xffffc0) with a 2 pixel 3D border
$c->setBackground(0xffffc0, 0xffffc0, 2);

#Set the plotarea at (70, 50) and of size 320 x 150 pixels. Set background
#color to white (0xffffffff). Enable both horizontal and vertical grids by
#setting their colors to light grey (0xc0c0c0)
$c->setPlotArea(70, 50, 320, 150, 0xffffffff, 0xffffffff, 0xc0c0c0, 0xc0c0c0);

#Add a title to the chart
$title = $c->addTitle("Revenue for " . $SelectedYear, "timesbi.ttf");
$title->setBackground(0xffff00);

#Add a legend box at the top of the plotarea
$legend = $c->addLegend(70, 30, 0, "", 8);
$legend->setBackground(Transparent);

```

```

#Add a line chart layer using the supplied data
$layer = $c->addLineLayer2();
$dataset = $layer->addDataSet($software, -1, "Software");
$dataset->setLineWidth(3);
$dataset = $layer->addDataSet($hardware, -1, "Hardware");
$dataset->setLineWidth(3);
$dataset = $layer->addDataSet($services, -1, "Services");
$dataset->setLineWidth(3);
$layer->setBorderColor(Transparent, 1);

#Set the x axis labels. In this example, the labels must be Jan - Dec.
$labels = array("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
    "Sept", "Oct", "Nov", "Dec");
$c->xAxis->setLabels($labels);

#Set the x-axis width to 2 pixels
$c->xAxis->setWidth(2);

#Set the y axis title
$c->yAxis->setTitle("USD (K)");

#Set the y-axis width to 2 pixels
$c->yAxis->setWidth(2);

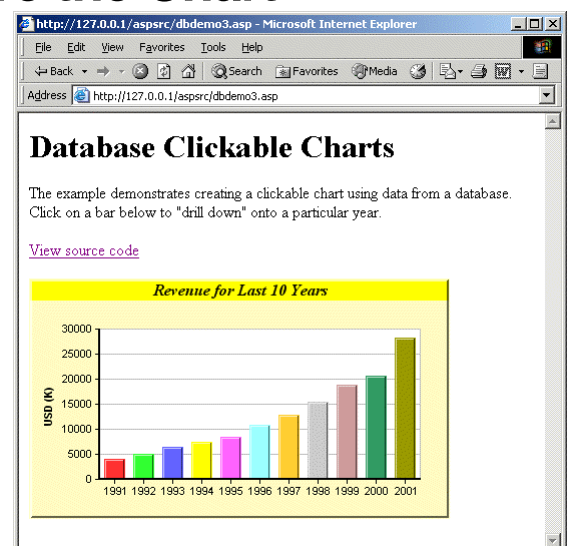
#Output the chart
header("Content-type: image/png");
print($c->makeChart2(PNG));
?>

```

## Use Session Variables to Store the Chart

In this example, we will create a chart that is “clickable”. We will display a bar chart that shows the revenue for the last 10 years. When the user clicks on a bar, it will “drill down” to another bar chart showing the monthly revenue for the year represented by the bar clicked.

To discuss this example, some understanding on how ChartDirector generates clickable charts is needed. Clickable Charts are discussed in the next chapter. So this example will be presented in the next chapter under the section [Database Clickable Charts](#).





---

# Clickable Charts

ChartDirector creates charts as images in PNG, GIF, JPEG or WBMP format. These images are usually incorporated into HTML web pages by using the `<IMG>` tags. In HTML, an image can be made “clickable” by using an “image map” (sometimes also called an “area map”) to define the “hot spots” on the image. An “image map” is defined using the `<MAP>` and `<AREA>` tags.

ChartDirector can automatically generate image maps for the charts it produces, thereby making the charts clickable. ChartDirector can generate image maps for the data contents of the chart (that is, bars for bar charts, sectors for pie charts, areas for area charts, etc.), for the legend keys, title boxes, or custom text boxes.

By clickable, it means the chart can response to mouse events just like an ordinary HTML link or button. This includes mouse clicking, and also other mouse events such as “onMouseOver” or “onMouseOut”.

Clickable charts are most often used to produce charts that have “drill down” or “zoom in” capabilities. For example, a pie chart could be created such that when a user clicks on the sectors, the browser will load another web page containing a more detail chart regarding that sector. Similarly, a day-of-week bar chart could be created such that when a user clicks on a bar, the browser will load an hour-of-day chart for the day that the user has clicked.

Other applications of image maps include providing “tool tips” or “bubble help” for the objects in the chart, or to activate Javascripts when the user clicks on the objects.

## Operation Overview

For a browser to display an image with an image map, the following sequence of events must have occurred:

1. The browser sends a HTTP request to the server.
2. The server returns a HTML web page that contains an `<IMG>` tag for the image, and the `<MAP>` and `<AREA>` tags for the image map.
3. The browser receives the HTML web page. It then sends another HTTP request to the server using the URL in the `<IMG>` tag for the actual image..
4. The server returns the actual image to the browser.

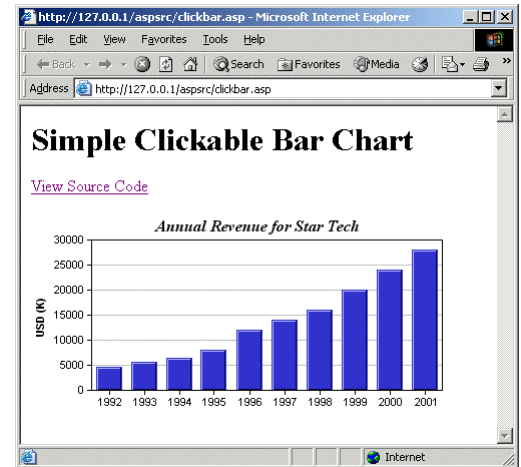
Note that a ChartDirector chart image needs to be generated in step (2), so that we can produce the image map. However, the image itself can only be delivered to the browser in step (4), which is another HTTP connection. So after the chart image is produced in step (2), it must be saved in “some place”. When step (4) occurs, the server can send the image to the browser.

The “some place” can be a “session variable” or a temporary file. The examples shown in this chapter will be based on the “session variable” approach.

## Simple Clickable Charts

In this example, we will create a bar chart that displays annual revenue for a company for the last 10 years. When a bar is clicked, the browser will load a line chart showing the monthly revenue for the selected year. When the line chart is clicked, the browser will load a pie chart showing the breakdown of the revenue for the selected month. When the pie chart is clicked, it will show the data in detail in an HTML page.

The capability is often called “drill-down”, because the user can “zoom-in” to get more details by clicking on the chart.



(The following program is available as “phpdemo/clickbar.php”).

```
<?php
include("phpchartdir.php");

#
#For demo purpose, we use hard coded data. In real life, the following data
#could come from a database.
#
$revenue = array(4500, 5600, 6300, 8000, 12000, 14000, 16000, 20000,
    24000, 28000);
$labels = array("1992", "1993", "1994", "1995", "1996", "1997", "1998",
    "1999", "2000", "2001");

#Create a XYChart object of size 450 x 200 pixels
$c = new XYChart(450, 200);

#Add a title to the chart using Times Bold Italic font
$c->addTitle("Annual Revenue for Star Tech", "timesbi.ttf");

#Set the plotarea at (60, 25) and of size 350 x 150 pixels
$c->setPlotArea(60, 25, 350, 150);

#Add a blue (0x3333cc) bar chart layer using the given data. Set the bar
#border to 1 pixel 3D style.
$barLayerObj = $c->addBarLayer($revenue, 0x3333cc, "Revenue");
$barLayerObj->setBorderColor(-1, 1);

#Set x axis labels using the given labels
```

```

$c->xAxis->setLabels($labels);

#Add a title to the y axis
$c->yAxis->setTitle("USD (K)");

#Create the image and save it in a session variable
session_register("chart");
$HTTP_SESSION_VARS["chart"] = $chart = $c->makeChart2(PNG);
$chartURL = "myimage.php?img=chart&id=".uniqid(session_id())."&".SID;

#Create an image map for the chart
$imageMap = $c->getHTMLImageMap("clickline.php", "",
    "alt='{xLabel}: USD {value|0}K'");
?>
<html>
<body>
<h1>Simple Clickable Bar Chart</h1>
<p><a href="viewsource.php?file=<?=$HTTP_SERVER_VARS["SCRIPT_NAME"]?>">
View Source Code
</a></p>


<map name="map1">
<?=$imageMap?>
</map>

</body>
</html>

```

In the above code, a ChartDirector chart is generated and saved in a session variable called “chart”. A URL is then created to allow the chart to be retrieved using the <IMG> tag. The code that performs these functions is as follows:

```

session_register("chart");
$HTTP_SESSION_VARS["chart"] = $chart = $c->makeChart2(PNG);
$chartURL = "myimage.php?img=chart&id=".uniqid(session_id())."&".SID;

```

In the above code, the “myimage.php” (explained below) is a simple PHP script that retrieves the image from the session variable and return the image to the browser. The “uniqid(session\_id())” parameter does not have any function, other than making the URL “unique”. We need to make the URL unique to prevent the browser from loading the image from cache. The “SID” in the URL is required by PHP to support session variables for browsers that disable cookies.

### A Note on Browser Cache

Most browsers will cache images (at least in memory, if not in a temporary file), even if you told it not to by using various HTTP headers.

The browser may or may not use the cached image depending on how the HTML page is loaded. For example, if the page is loaded by using the “Refresh” button, the browser may not use the cached image. However, if the page is loaded by the “Back” button, the

browser may use the cached image even if the HTTP headers instruct the browser not to cache the contents. The exact behaviour depends on the version and brand of the browsers.

To solve the caching problem, one method is to make sure different images have different URLs. The technique is used in the ChartDirector sample code by creating unique URLs for each image.

The image map for the chart is created using the following code:

```
$imageMap = $c->getHTMLImageMap("clickline.php", "",  
    "alt='{xLabel}: USD {value|0}K'");
```

As seen above, all you need is just one line of code. The [getHTMLImageMap](#) method will generate the image map for the charts. The image map will use “clickline.php” as the handler when the user clicks on the bars.

After producing the chart image and the image map, the rest of the code is just standard HTML tags to produce the web page.

If you right click on the browser and choose “View Source” to look at the HTML of the web page as received by the browser, you can see the image map generated by the above line of code will be something like the followings:

```
<area shape="rect" coords="65,152,90,175" alt='1992: USD 4500K'  
    href="clickline.php?x=0&xLabel=1992&dataSet=0&dataSetName=Revenue&value=4500">  
  
<area shape="rect" coords="100,147,125,175" alt='1993: USD 5600K'  
    href="clickline.php?x=1&xLabel=1993&dataSet=0&dataSetName=Revenue&value=5600">  
  
<area shape="rect" coords="135,143,160,175" alt='1994: USD 6300K'  
    href="clickline.php?x=2&xLabel=1994&dataSet=0&dataSetName=Revenue&value=6300">  
  
<area shape="rect" coords="169,135,194,175" alt='1995: USD 8000K'  
    href="clickline.php?x=3&xLabel=1995&dataSet=0&dataSetName=Revenue&value=8000">  
  
..... (one <area> tag for each bar) .....
```

The image map generated by ChartDirector will contain one area tag for each of the bars. The “href” attribute of the <area> tag will invoke “clickline.php” with a some query parameters to describe the bar. In this way, the “clickline.php” will know which bar the user has clicked.

The <area> tag also includes the “alt” attribute, which will become the “tool tip” text when the mouse moves over the bar. In this particular example, the “tool tip” text is in the format:

```
"alt='{xLabel}: USD {value|0}K'"
```

which is specified as the third argument when calling the [getHTMLImageMap](#) method.

For more details on how [getHTMLImageMap](#) generates image maps, please refer to the [ChartDirector API Reference](#).

As mentioned above, the “myimage.php” is used to retrieve the image from the session variable and returns it to the browser. The “myimage.php” is a simple “utility” which will be repeatedly used in the examples in this chapter. The code for “myimage.php” is as follows:

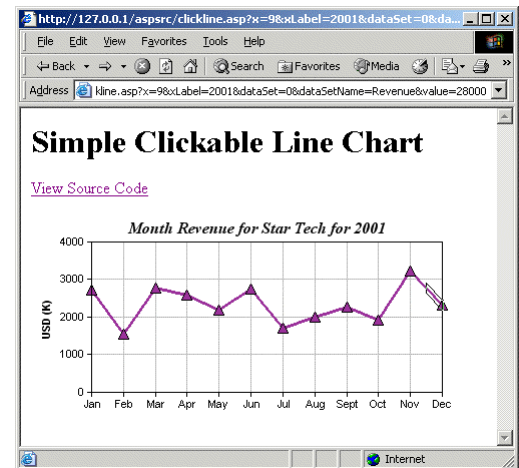
(The following program is available as “phpdemo/myimage.php”).

```
<?php
session_start();
print $HTTP_SESSION_VARS[$HTTP_GET_VARS["img"]];
?>
```

The code for the image map handler “clickline.php” is listed below.

The “clickline.php” determines which year the user has clicked from the query parameters. It then draws a line chart based on selected year. To produce a clickable line chart, it saves the chart in a session variable and creates a unique URL based on “myimage.php”. It generates the image map using the [getHTMLImageMap](#) method, with “clickpie.php” as the handler.

As “clickline.php” is very similar to “clickbar.php”, it will not be described further here.



(The following program is available as “phpdemo/clickline.php”).

```
<?php
include("phpchartdir.php");

#Get the selected year.
$selectedYear = $HTTP_GET_VARS["xLabel"];

#Get the total revenue
$totalRevenue = $HTTP_GET_VARS["value"];

#
# In this demo, we just split the total revenue into 12 months using
# random numbers. In real life, the data can come from a database.
#
srand($selectedYear);
$data = array_pad(array(), 12, 0);
for($i = 0; $i < 11; ++$i) {
    $data[$i] = $totalRevenue * (rand() / getrandmax() * 0.6 + 0.6) / (12
        - $i);
    $totalRevenue = $totalRevenue - $data[$i];
}
$data[11] = $totalRevenue;
```

```

#
# Now we obtain the data into arrays, we can start to draw the chart using
# ChartDirector
#

#Create a XYChart object of size 450 x 200 pixels
$c = new XYChart(450, 200);

#Add a title to the chart
$c->addTitle("Month Revenue for Star Tech for ".$selectedYear,
    "timesbi.ttf");

#Set the plotarea at (60, 5) and of size 350 x 150 pixels. Enable both
#horizontal and vertical grids by setting their colors to grey (0xc0c0c0)
$plotAreaObj = $c->setPlotArea(60, 25, 350, 150);
$plotAreaObj->setGridColor(0xc0c0c0, 0xc0c0c0);

#Add a line chart layer using the data
$lineLayerObj = $c->addLineLayer();
$dataSet = $lineLayerObj->addDataSet($data, 0x993399);

#Set the line width to 3 pixels
$dataSet->setLineWidth(3);

#Use a 11 point triangle symbol to plot the data points
$dataSet->setDataSymbol(TriangleSymbol, 11);

#Set the x axis labels. In this example, the labels must be Jan - Dec.
$labels = array("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
    "Sept", "Oct", "Nov", "Dec");
$c->xAxis->setLabels($labels);

#Add a title to the x axis to reflect the selected year
$c->xAxis->setTitle("Year ".$selectedYear);

#Add a title to the y axis
$c->yAxis->setTitle("USD (K)");

#Reserve 10% margin at the top of the plot area just to make sure the line
#does not go too near the top of the plot area
$c->yAxis->setAutoScale(0.1);

#Create the image and save it in a session variable
session_register("chart");
$HTTP_SESSION_VARS["chart"] = $chart = $c->makeChart2(PNG);
$chartURL = "myimage.php?img=chart&id=".uniqid(session_id())."&".SID;

#Create an image map for the chart
$imageMap = $c->getHTMLImageMap("clickpie.php?year=".$selectedYear, "",
    "alt='{xLabel}: USD {value|0}K'");
?>
<html>
<body>
<h1>Simple Clickable Line Chart</h1>
<p><a href="viewsource.php?file=<?=$HTTP_SERVER_VARS["SCRIPT_NAME"]?>">
View Source Code
</a></p>

```

```

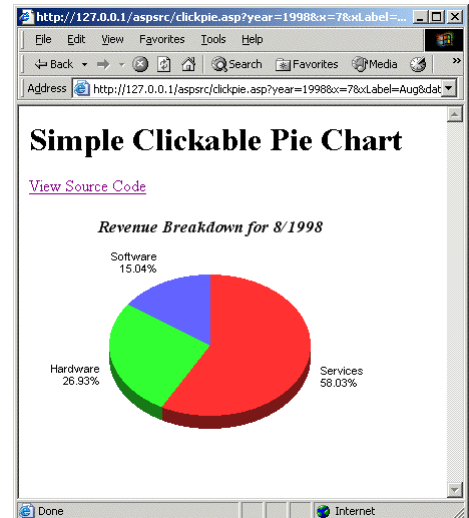

<map name="map1">
<?=$imageMap?>
</map>
</body>
</html>

```

When the line chart is clicked, the handler “clickpie.php” will be invoked. Its code is listed below.

The “clickpie.php” determines which year and month the user has clicked from the query parameters. It then draws a pie chart based on those parameters. As similar to “clickbar.php” and “clickline.php”, it saves the chart in a session variable, creates a unique URL based on “myimage.php”, and generates the image map using the getHTMLImageMap method, with “piestub.php” as the handler.

As “clickpie.php” is very similar to “clickbar.php” and “clickline.php”, it will not be described further here.



(The following program is available as “phpdemo/clickpie.php”).

```

<?php
include("phpchartdir.php");

#Get the selected year and month
$selectedYear = $HTTP_GET_VARS["year"];
$selectedMonth = $HTTP_GET_VARS["x"] + 1;

#Get the monthly revenue
$monthlyRevenue = $HTTP_GET_VARS["value"];

#
# In this demo, we just split the total revenue into 3 parts using random
# numbers. In real life, the data probably can come from a database.
#
srand($selectedMonth * 2000 + $selectedYear);
$data = array_pad(array(), 3, 0);
$data[0] = (rand() / getrandmax() * 0.1 + 0.5) * $monthlyRevenue;
$data[1] = (rand() / getrandmax() * 0.1 + 0.2) * $monthlyRevenue;
$data[2] = $monthlyRevenue - $data[0] - $data[1];

#The labels for the pie chart
$labels = array("Services", "Hardware", "Software");

#Create a PieChart object of size 360 x 260 pixels
$c = new PieChart(360, 260);

#Set the center of the pie at (180, 140) and the radius to 100 pixels
$c->setPieSize(180, 130, 100);

```

```

#Add a title to the pie chart using 13 pts Times Bold Italic font
$c->addTitle("Revenue Breakdown for ".$selectedMonth."/ ".$selectedYear,
    "timesbi.ttf", 13);

#Draw the pie in 3D
$c->set3D();

#Set the pie data and the pie labels
$c->setData($data, $labels);

#Create the image and save it in a session variable
session_register("chart");
$HTTP_SESSION_VARS["chart"] = $chart = $c->makeChart2(PNG);
$chartURL = "myimage.php?img=chart&id=".uniqid(session_id())."&".SID;

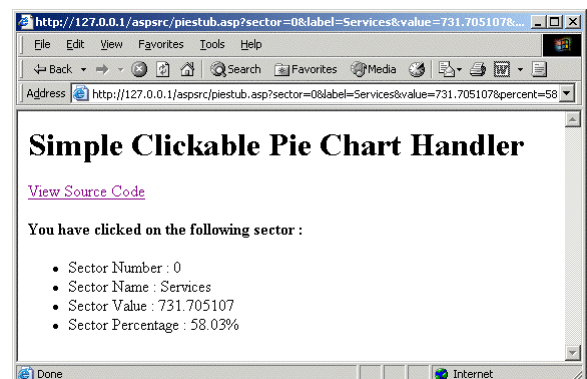
#Create an image map for the chart
$imageMap = $c->getHTMLImageMap("piestub.php", "",
    "alt='{label}:USD {value|0}K'");
?>
<html>
<body>
<h1>Simple Clickable Pie Chart</h1>
<p><a href="viewsource.php?file=<?=$HTTP_SERVER_VARS["SCRIPT_NAME"]?>">
View Source Code
</a></p>


<map name="map1">
<?=$imageMap?>
</map>
</body>
</html>

```

Finally, when the pie is clicked, the handler “piestub.php” will be invoked. Its code is listed below.

In this example, for demo purposes the “piestub.php” simply displays the data about the pie clicked in text format.



(The following program is available as “phpdemo/piestub.php”).

```

<html>
<body>
<h1>Simple Clickable Pie Chart Handler</h1>
<p><a href="viewsource.php?file=<?=$HTTP_SERVER_VARS["SCRIPT_NAME"]?>">
View Source Code
</a></p>

```



```

<p><b>You have clicked on the following sector :</b></p>
<ul>
  <li>Sector Number : <?=$HTTP_GET_VARS["sector"]?></li>
  <li>Sector Name : <?=$HTTP_GET_VARS["label"]?></li>
  <li>Sector Value : <?=$HTTP_GET_VARS["value"]?></li>
  <li>Sector Percentage : <?=$HTTP_GET_VARS["percent"]?>%</li>
</ul>
</body>
</html>

```

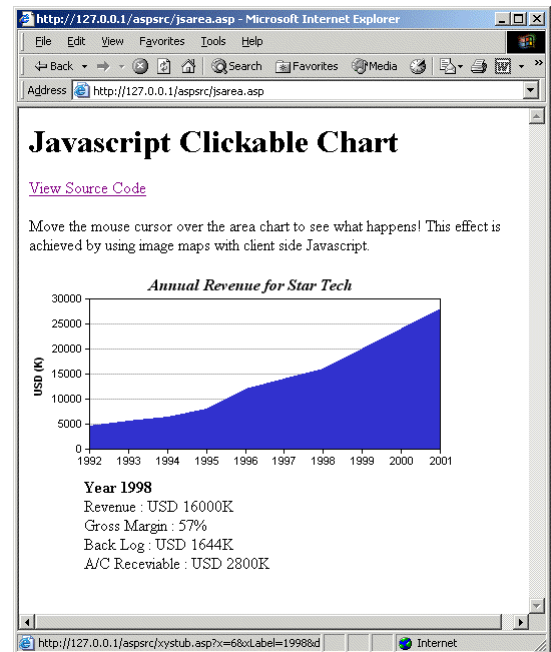
## Javascript Clickable Charts

This example demonstrates how you could use client side Javascript with ChartDirector's image map features.

In this example, a “clickable” area chart will be produced. Apart from responding to mouse clicks, the area chart will display in real time detail information about the data points when the mouse moves over the area in the chart. It is accomplishing by using the “onMouseOver” and “onMouseOut” event with client side Javascript and dynamic HTML.

(This example assumes the reader has basic understanding of client side Javascript and dynamic HTML.)

The code for this example is listed below.



(The following program is available as “phpdemo/jsarea.php”).

```

<?php
include("phpchartdir.php");

#
#For demo purpose, we use hard coded data. In real life, the following data
#could come from a database.
#
$revenue = array(4500, 5600, 6300, 8000, 12000, 14000, 16000, 20000,
  24000, 28000);
$grossMargin = array(62, 65, 63, 60, 55, 56, 57, 53, 52, 50);
$backLog = array(563, 683, 788, 941, 1334, 1522, 1644, 1905, 2222, 2544);
$receivable = array(750, 840, 860, 1200, 2200, 2700, 2800, 3900, 4900, 6000
);
$labels = array("1992", "1993", "1994", "1995", "1996", "1997", "1998",
  "1999", "2000", "2001");

#Create a XYChart object of size 440 x 200 pixels
$c = new XYChart(440, 200);

```

```

#Add a title to the chart using Times Bold Italic font
$c->addTitle("Annual Revenue for Star Tech", "timesbi.ttf");

#Set the plotarea at (60, 5) and of size 350 x 150 pixels
$c->setPlotArea(60, 25, 350, 150);

#Add an area chart layer for the revenue data
$areaLayerObj = $c->addAreaLayer($revenue, 0x3333cc, "Revenue");
$areaLayerObj->setBorderColor(SameAsMainColor);

#Set the x axis labels using the given labels
$c->xAxis->setLabels($labels);

#Add a title to the y axis
$c->yAxis->setTitle("USD (K)");

#Create the image and save it in a session variable
session_register("chart");
$HTTP_SESSION_VARS["chart"] = $chart = $c->makeChart2(PNG);
$chartURL = "myimage.php?img=chart&id=".uniqid(session_id())."&".SID;

#Client side Javascript to show detail information "onmouseover"
$showText = "onmouseover='setDIV(\"info{x}\", \"visible\");' ";

#Client side Javascript to hide detail information "onmouseout"
$hideText = "onmouseout='setDIV(\"info{x}\", \"hidden\");' ";

#"alt" attribute to show tool tip
$toolTip = "alt='{xLabel}: USD {value|0}K'";

#Create an image map for the chart
$imageMap = $c->getHTMLImageMap("xystub.php", "", $showText.$hideText.
    $toolTip);
?>
<html>
<body>
<h1>Javascript Clickable Chart</h1>
<p><a href="viewsource.php?file=<?=$HTTP_SERVER_VARS["SCRIPT_NAME"]?>">
View Source Code
</a></p>

<p style="width:500px">
Move the mouse cursor over the area chart to see what happens!
This effect is achieved by using image maps with client side Javascript.
</p>


<map name="map1">
<?=$imageMap?>
</map>

<br>

<!-------
      Create the DIV layers to show detail information
----->

```

```

<?for($i = 0; $i < count($revenue); ++$i) {?>

    <div id="info<?=$i?>"
        style="visibility:hidden;position:absolute;left:65px;">
        <b>Year <?=$labels[$i]?></b><br>
        Revenue : USD <?=$revenue[$i]?>K<br>
        Gross Margin : <?=$grossMargin[$i]?>%<br>
        Back Log : USD <?=$backLog[$i]?>K<br>
        A/C Receivable : USD <?=$receivable[$i]?>K<br>
    </div>

<?}?>

<!-------
    Client side utility function to show and hide
    a layer. Works in both IE and Netscape browsers.
----->
<SCRIPT>
function setDIV(id, cmd) {
    if (document.all)
        //IE Browser
        document.all[id].style.visibility = cmd;
    else
        //Netscape Browser
        document[id].visibility = cmd;
}
</SCRIPT>

</body>
</html>

```

Similar to the previous examples, the above code follows the general structure of producing a clickable chart. It creates the chart and saves it to a session variable. It then creates a unique URL to retrieve the chart using “myimage.php”. The image map is produced using [getHTMLImageMap](#).

The key difference between this example and the previous example is what is included in the image map. In this example, the image map is created using the following code:

```

#Client side Javascript to show detail information "onmouseover"
$showText = "onmouseover='setDIV(\"info{x}\", \"visible\");' ";

#Client side Javascript to hide detail information "onmouseout"
$hideText = "onmouseout='setDIV(\"info{x}\", \"hidden\");' ";

#"alt" attribute to show tool tip
$toolTip = "alt='{xLabel}: USD {value|0}K'";

#Create an image map for the chart
$imageMap = $c->getHTMLImageMap("xystub.php", "", $showText.$hideText.
    $toolTip);

```

Note that in addition to the “alt” attribute for the “tool tip”, the image map also includes “onmouseover” and “onmouseout” attributes. These two attributes will instruct the browser to execute

the corresponding code when the mouse moves over and moves out of the areas defined in the <area> tags.

The code in the “onmouseover” and “onmouseout” attribute calls a client side Javascript function “setDIV”. This function is simple a utility to show and hide a <DIV> block in dynamic HTML. The definition of “setDIV” is as follows. It is included as part of the web page to be sent to the browser.

```
<!-------
      Client side utility function to show and hide
      a layer. Works in both IE and Netscape browsers.
----->
<SCRIPT>
function setDIV(id, cmd) {
    if (document.all)
        //IE Browser
        document.all[id].style.visibility = cmd;
    else
        //Netscape Browser
        document[id].visibility = cmd;
}
</SCRIPT>
```

The <DIV> blocks that show the detail information are generated using the following code.

```
<!-------
      Create the DIV layers to show detail information
----->
<?for($i = 0; $i < count($revenue); ++$i) {?>

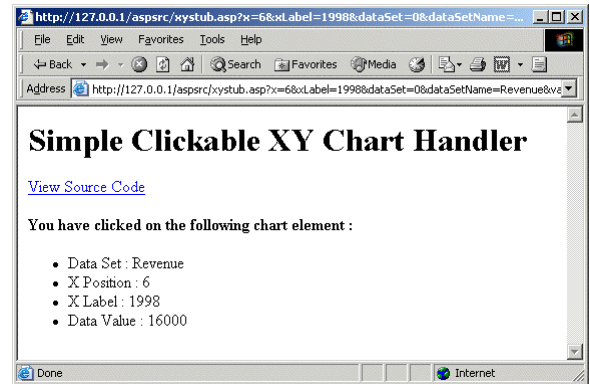
    <div id="info<?=$i?>"
        style="visibility:hidden;position:absolute;left:65px;">
        <b>Year <?=$labels[$i]?></b><br>
        Revenue : USD <?=$revenue[$i]?>K<br>
        Gross Margin : <?=$grossMargin[$i]?>%<br>
        Back Log : USD <?=$backLog[$i]?>K<br>
        A/C Receivable : USD <?=$receivable[$i]?>K<br>
    </div>

<?}?>
```

Each data point is associated with a DIV block that shows its detail information. As seen from the “style” attribute in the above code, initially the DIV blocks are hidden. When the mouse moves over the area defined in an <area> tag, the “onmouseover” event handler will call the “setDIV” function to show the corresponding DIV block. When the mouse moves out of that area, the “onmouseout” event handler will call the “setDIV” function to hide the DIV block.

In addition to responding to mouse over and mouse out events, the area chart is also clickable using “xystub.php” as the handler. In this example, the “xystub.php” simply displays what is clicked.

The code for the “xystub.php” is listed below.



(The following program is available as “phpdemo/xysub.php”).

```
<html>
<body>
<h1>Simple Clickable XY Chart Handler</h1>
<p><a href="viewsource.php?file=<?=$HTTP_SERVER_VARS["SCRIPT_NAME"]?>">
View Source Code
</a></p>

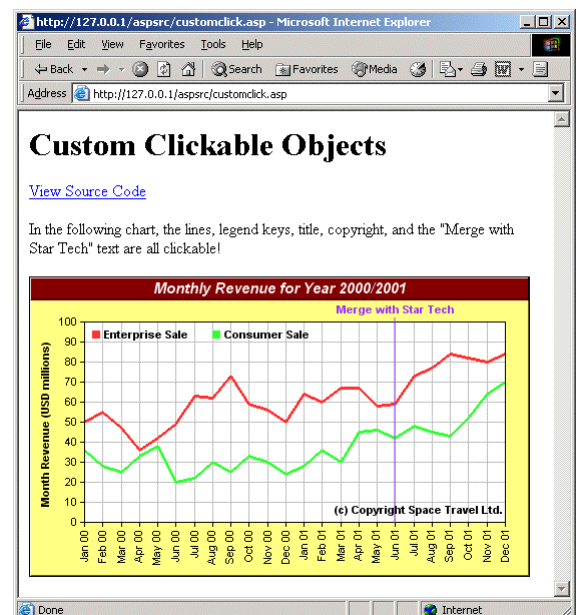
<p><b>You have clicked on the following chart element :</b></p>
<ul>
<li>Data Set : <?=$HTTP_GET_VARS["dataSetName"]?></li>
<li>X Position : <?=$HTTP_GET_VARS["x"]?></li>
<li>X Label : <?=$HTTP_GET_VARS["xLabel"]?></li>
<li>Data Value : <?=$HTTP_GET_VARS["value"]?></li>
</ul>
</body>
</html>
```

## Custom Clickable Objects

In this example, a clickable line chart will be created with legend keys, a title box, a vertical mark with label, and a custom text box. All these objects will be clickable in addition to the chart data contents (that is, the lines in the line chart).

This example also demonstrate how to use client side Javascript as the click handler.

The code for this example is listed below.



(The following program is available as “phpdemo/customclick.php”).

```
<?php
include("phpchartdir.php");

#The data for the line chart
$data0 = array(50, 55, 47, 36, 42, 49, 63, 62, 73, 59, 56, 50, 64, 60, 67,
    67, 58, 59, 73, 77, 84, 82, 80, 84);
$data1 = array(36, 28, 25, 33, 38, 20, 22, 30, 25, 33, 30, 24, 28, 36, 30,
    45, 46, 42, 48, 45, 43, 52, 64, 70);

#The labels for the line chart
$labels = array("Jan-00", "Feb-00", "Mar-00", "Apr-00", "May-00",
    "Jun-00", "Jul-00", "Aug-00", "Sep-00", "Oct-00", "Nov-00", "Dec-00",
    "Jan-01", "Feb-01", "Mar-01", "Apr-01", "May-01", "Jun-01", "Jul-01",
    "Aug-01", "Sep-01", "Oct-01", "Nov-01", "Dec-01");

#Create a XYChart object of size 500 x 300 pixels
$c = new XYChart(500, 300);

#Use a pale yellow background (0xffff80) with a black (0x0) edge and a 1
#pixel 3D border
$c->setBackground(0xffff80, 0x0, 1);

#Set plotarea at (55, 45) with size of 420 x 200 pixels. Use white
#(0xffffffff) background. Enable both horizontal and vertical grid by setting
#their colors to light grey (0xc0c0c0)
$plotAreaObj = $c->setPlotArea(55, 45, 420, 200, 0xffffffff);
$plotAreaObj->setGridColor(0xc0c0c0, 0xc0c0c0);

#Add a legend box at (55, 45) (top of plot area) using 8 pts Arial Bold
#font with horizontal layout Set border and background colors of the legend
#box to Transparent
$legendBox = $c->addLegend(55, 45, false, "arialbd.ttf", 8);
$legendBox->setBackground(Transparent);

#Reserve 10% margin at the top of the plot area during auto-scaling to
#leave space for the legends.
$c->yAxis->setAutoScale(0.1);

#Add a title to the chart using 11 pts Arial Bold Italic font. The text is
#white 0xffffffff on a dark red 0x800000 background.
$title = $c->addTitle("Monthly Revenue for Year 2000/2001", "arialbi.ttf",
    11, 0xffffffff);
$title->setBackground(0x800000, -1, 1);

#Add a title to the y axis
$c->yAxis->setTitle("Month Revenue (USD millions)");

#Set the labels on the x axis. Draw the labels vertical (angle = 90)
$labelsObj = $c->xAxis->setLabels($labels);
$labelsObj->setFontAngle(90);

#Add a vertical mark at x = 17 using a semi-transparent purple (0x809933ff)
#color and Arial Bold font. Attached the mark (and therefore its label) to
#the top x axis.
```

```

$xAxis2Obj = $c->xAxis2();
$mark = $xAxis2Obj->addMark(17, 0x809933ff, "Merge with Star Tech",
    "arialbd.ttf");

#Set the mark line width to 2 pixels
$mark->setLineWidth(2);

#Set the mark label font color to purple (0x9933ff)
$mark->setFontColor(0x9933ff);

#Add a copyright message at (475, 240) (bottom right of plot area) using
#Arial Bold font
$copyRight = $c->addText(475, 240, "(c) Copyright Space Travel Ltd.",
    "arialbd.ttf");
$copyRight->setAlignment(BottomRight);

#Add a line layer to the chart
$layer = $c->addLineLayer();

#Set the default line width to 3 pixels
$layer->setLineWidth(3);

#Add the data sets to the line layer
$layer->addDataSet($data0, -1, "Enterprise");
$layer->addDataSet($data1, -1, "Consumer");

#Create the image and save it in a session variable
session_register("chart");
$HTTP_SESSION_VARS["chart"] = $chart = $c->makeChart2(PNG);
$chartURL = "myimage.php?img=chart&id=".uniqid(session_id())."&".SID;

#Create an image map for the chart
$chartImageMap = $c->getHTMLImageMap("xystub.php", "",
    "alt='{dataSetName} @ {xLabel} = USD {value|0} millions'");

#Create an image map for the legend box
$legendImageMap = $legendBox->getHTMLImageMap("javascript:doSomething();",
    " ", "alt='This legend key is clickable!'");

#Obtain the image map coordinates for the title, mark, and copyright
#message. These will be used to define the image map inline. (See HTML code
#below.)
$titleCoor = $title->getImageCoor();
$markCoor = $mark->getImageCoor();
$copyRightCoor = $copyRight->getImageCoor();
?>
<html>
<body>
<h1>Custom Clickable Objects</h1>
<p><a href="viewsource.php?file=<?=$HTTP_SERVER_VARS["SCRIPT_NAME"]?>">
View Source Code
</a></p>

<p style="width:500px">
In the following chart, the lines, legend keys, title, copyright,
and the "Merge with Star Tech" text are all clickable!
</p>

```

```


<map name="map1">
<?=$chartImageMap?>
<?=$legendImageMap?>
<area <?=$titleCoor?> href='javascript:doSomething();'
    alt='The title is clickable! '>
<area <?=$markCoor?> href='javascript:doSomething();'
    alt='The "Merge with Star Tech" text is clickable! '>
<area <?=$copyRightCoor?> href='javascript:doSomething();'
    alt='The copyright text is clickable! '>
</map>

<SCRIPT>
function doSomething() {
    alert("This is suppose to do something meaningful, but for demo " +
        "purposes, we just pop up this message box to prove that " +
        "the object can response to mouse clicks.");
}
</SCRIPT>
</body>
</html>

```

Similar to the previous examples, the above code follows the general structure of producing a clickable chart. It creates the chart and saves it to a session variable. It then creates a unique URL to retrieve the chart using “myimage.php”.

The image map in this example, however, consists of multiple statements.

The image map for the line chart contents are produced using the [getHTMLImageMap](#) method of the BaseChart object, with “xystub.php” as the handler, and with the tool tip text is set to “'{dataSetName} @ {xLabel} = USD {value|0} millions'”.

```

$chartImageMap = $c->getHTMLImageMap("xystub.php", "",
    "alt='{dataSetName} @ {xLabel} = USD {value|0} millions'");

```

The “xystub.php” in this example is the same as the one used in the last example, so it will not be explained further.

The image map for the legend keys are produced using the [getHTMLImageMap](#) method of the LegendBox object. Instead of using a URL as the handler, in this example, a client side Javascript will be executed when the user clicks on the legend key.

```

$legendImageMap = $legendBox->getHTMLImageMap("javascript:doSomething();",
    " ", "alt='This legend key is clickable!'");

```

As shown above, the “javascript:doSomething();” is used as the handler. ChartDirector will put the handler in the “href” attribute of the <area> tag. In HTML, if the “href” tag starts with “javascript:”, the browser will interpret it as a client side Javascript statement to be executed when the <area> is clicked.



Also note that the second argument to the `getHTMLImageMap` method is set to a space " ". The second argument controls what HTTP query parameters get passed to the handler. In previous examples, this argument is always an empty string "", which means the default set of query parameters will be used. For this example, since the handler is a client side Javascript, no HTTP query parameter is necessary. So a space character is used instead.

The client side Javascript that is executed in this example is a function called `doSomething()`. In this example, this just pops up a message.

```
<SCRIPT>
function doSomething() {
    alert("This is suppose to do something meaningful, but for demo " +
        "purposes, we just pop up this message box to prove that " +
        "the object can response to mouse clicks.");
}
</SCRIPT>
```

Most text messages in ChartDirector are represented as `TextBox` objects. You may obtain the image map coordinates of the `TextBox` using the `getImageCoor` method. With the image map coordinates, the `<area>` tag for the `TextBox` object can be created easily.

(For `TextBox`, there is no `getHTMLImageMap` method. It is because each `TextBox` object only needs one `<area>` tag. Thus it is more convenient to enter the handler and tool tip text directly into the `<area>` tag, rather than generating them using `getHTMLImageMap`.)

In this example, the image map coordinates of the title `TextBox`, vertical mark `TextBox`, and the copyright message (custom text) `TextBox` are obtained as follows.

```
$titleCoor = $title->getImageCoor();
$markCoor = $mark->getImageCoor();
$copyrightCoor = $copyright->getImageCoor();
```

The image map coordinates are then used to make the `<area>` tags as follows:

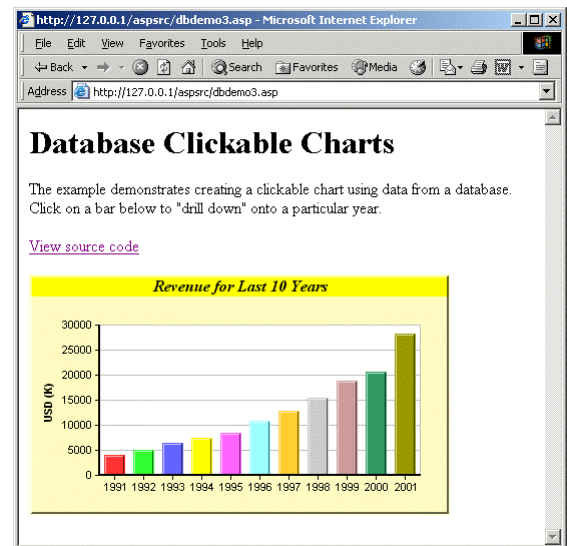
```
<area <?=$titleCoor?> href='javascript:doSomething();'
    alt='The title is clickable! '>
<area <?=$markCoor?> href='javascript:doSomething();'
    alt='The "Merge with Star Tech" text is clickable! '>
<area <?=$copyrightCoor?> href='javascript:doSomething();'
    alt='The copyright text is clickable! '>
```

# Database Clickable Charts

In this section, we will present an example that employs both database and image maps.

We will display a bar chart that shows the revenue for the last 10 years. When the user clicks on a bar, it will “drill down” to another bar chart showing the monthly revenue for the year represented by the bar clicked. All data comes from a database.

The code that creates the clickable bar chart for the last 10 years are as follows.



(The following program is available as “phpdemo\dbdemo3.php”).

```
<?
include("phpchartdir.php");

#
#Get the revenue for the last 10 years
#
$SQLstatement =
    "Select Sum(Software + Hardware + Services), Year(TimeStamp) " .
    "From revenue Where Year(TimeStamp) >= 1991 " .
    "And Year(TimeStamp) <= 2001 " .
    "Group By Year(TimeStamp) Order By Year(TimeStamp)";

#
#Read in the revenue data into arrays
#
mysql_connect("localhost", "test", "test");
$result = mysql_db_query("sample", $SQLstatement);
while ($row = mysql_fetch_row($result)) {
    $revenue[] = $row[0];
    $timestamp[] = $row[1];
}

#
#Now we obtain the data into arrays, we can start to draw the chart
#using ChartDirector
#

#Create a XYChart of size 420 pixels x 240 pixels
$c = new XYChart(420, 240);

#Set the chart background to pale yellow (0xffffc0) with a 2 pixel 3D border
$c->setBackground(0xffffc0, 0xffffc0, 2);
```

```

#Set the plotarea at (70, 50) and of size 320 x 150 pixels. Set background
#color to white (0xffffffff). Enable both horizontal and vertical grids by
#setting their colors to light grey (0xc0c0c0)
$c->setPlotArea(70, 50, 320, 150, 0xffffffff, 0xffffffff, 0xc0c0c0, 0xc0c0c0);

#Add a title to the chart
$title = $c->addTitle("Revenue for Last 10 Years", "timesbi.ttf");
$title->setBackground(0xffff00);

#Add a legend box at the top of the plotarea
$legend = $c->addLegend(70, 30, 0, "", 8);
$legend->setBackground(Transparent);

#Add a multi-color bar chart layer using the supplied data
$layer = $c->addBarLayer3($revenue);
$layer->setBorderColor(Transparent, 1);

#Set the x-axis labels using the supplied labels
$c->xAxis->setLabels($timestamp);

#Set the x-axis width to 2 pixels
$c->xAxis->setWidth(2);

#Set the y axis title
$c->yAxis->setTitle("USD (K)");

#Set the y-axis width to 2 pixels
$c->yAxis->setWidth(2);

#Create the image and save it in a session variable
session_register("chart");
$HTTP_SESSION_VARS["chart"] = $chart = $c->makeChart2(PNG);
$chartURL = "myimage.php?img=chart&id=".uniqid(session_id())."&".SID;

#Create an image map for the chart
$imageMap = $c->getHTMLImageMap("dbdemo3a.php", "",
    "alt='{xLabel}: USD {value|0}K'");
?>
<html>
<body>
<h1>Database Clickable Charts</h1>
<p style="width:500px;">The example demonstrates creating a clickable chart
using data from a database. Click on a bar below to "drill down" onto a
particular year.</p>

<p><a href="viewsource.php?file=<?=$HTTP_SERVER_VARS["SCRIPT_NAME"]?>">
    View source code
</a></p>


<map name="map1">
<?=$imageMap?>
</map>

</body>
</html>

```

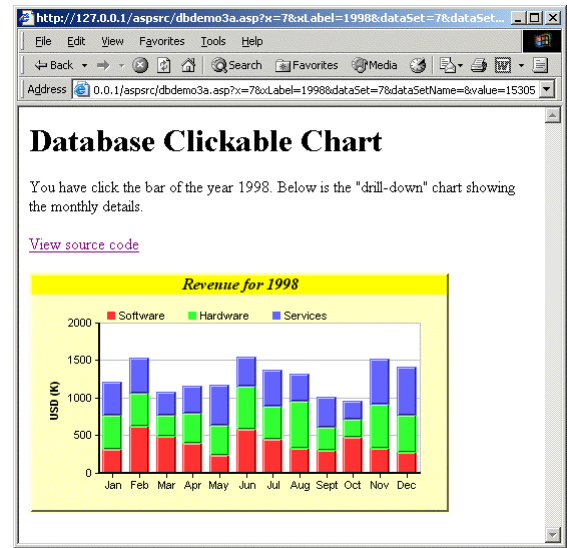
The above code first performs a database query and read the data into arrays. It then uses the data to create a bar chart.

Similar to the previous examples, the above code follows the general structure of producing a clickable chart. It creates the chart and saves it to a session variable. It then creates a unique URL to retrieve the chart using “myimage.php”. The image map is produced using `getHTMLImageMap`, with “dbdemo3a.php” as the handler.

When the user clicks on the bar chart, the handler “dbdemo3a.php” will be invoked with a number of HTTP query parameters to indicate which bar the user has clicked. In particular, the `xLabel` parameter will contain the x-axis label for the bar clicked.

Using the `xLabel` parameter, the “dbdemo3a.php” knows which year the user has clicked. It then query the database for the data on that year, and produces the bar chart for that year.

In this example, the “dbdemo3a.php” will produce another clickable chart using “xystub.php” as the handler. (The “xystub.php” is the same as the “xystub.php” in previous examples.)



(The following program is available as “phpdemo\dbdemo3a.php”).

```
<?
include("phpchartdir.php");

#
#Retrieve the selected year from the query variable "xLabel"
#
$SelectedYear = $_HTTP_GET_VARS["xLabel"];
if (!$SelectedYear) $SelectedYear = 2001;

#
#Create an SQL statement to get the revenues of each month for the
#selected year. The ArrayIndex will be from 0 - 11, representing Jan - Dec.
#
$SQLstatement = "Select Month(TimeStamp) - 1, " .
    "Software, Hardware, Services " .
    "From revenue Where Year(TimeStamp)=" . $SelectedYear;

#
#Read in the revenue data into arrays
#
mysql_connect("localhost", "test", "test");
$result = mysql_db_query("sample", $SQLstatement);
$software = array_pad(array(), 12, 0);
$hardware = array_pad(array(), 12, 0);
$services = array_pad(array(), 12, 0);
```

```

while ($row = mysql_fetch_row($result)) {
    $software[$row[0]] = $row[1];
    $hardware[$row[0]] = $row[2];
    $services[$row[0]] = $row[3];
}

#
#Now we obtain the data into arrays, we can start to draw the chart
#using ChartDirector
#

#Create a XYChart of size 420 pixels x 240 pixels
$c = new XYChart(420, 240);

#Set the chart background to pale yellow (0xffffc0) with a 2 pixel 3D border
$c->setBackground(0xffffc0, 0xffffc0, 2);

#Set the plotarea at (70, 50) and of size 320 x 150 pixels. Set background
#color to white (0xffffffff). Enable both horizontal and vertical grids by
#setting their colors to light grey (0xc0c0c0)
$c->setPlotArea(70, 50, 320, 150, 0xffffffff, 0xffffffff, 0xc0c0c0, 0xc0c0c0);

#Add a title to the chart
$title = $c->addTitle("Revenue for " . $SelectedYear, "timesbi.ttf");
$title->setBackground(0xffff00);

#Add a legend box at the top of the plotarea
$legend = $c->addLegend(70, 30, 0, "", 8);
$legend->setBackground(Transparent);

#Add a stacked bar chart layer using the supplied data
$layer = $c->addBarLayer2(Stack);
$layer->addDataSet($software, -1, "Software");
$layer->addDataSet($hardware, -1, "Hardware");
$layer->addDataSet($services, -1, "Services");
$layer->setBorderColor(Transparent, 1);

#Set the x axis labels. In this example, the labels must be Jan - Dec.
$labels = array("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
    "Sept", "Oct", "Nov", "Dec");
$c->xAxis->setLabels($labels);

#Set the x-axis width to 2 pixels
$c->xAxis->setWidth(2);

#Set the y axis title
$c->yAxis->setTitle("USD (K)");

#Set the y-axis width to 2 pixels
$c->yAxis->setWidth(2);

#Create the image and save it in a session variable
session_register("chart");
$HTTP_SESSION_VARS["chart"] = $chart = $c->makeChart2(PNG);
$chartURL = "myimage.php?img=chart&id=".uniqid(session_id())."&".SID;

#Create an image map for the chart

```

```

$imageMap = $c->getHTMLImageMap("xystub.php", "",
    "alt='{dataSetName} @ {xLabel} = USD {value|0}K'");
?>
<html>
<body>
<h1>Database Clickable Chart</h1>
<p style="width:500px;">You have click the bar of the year
<?=$HTTP_GET_VARS["xLabel"]?>. Below is the "drill-down" chart
showing the monthly details.</p>

<p><a href="viewsource.php?file=<?=$HTTP_SERVER_VARS["SCRIPT_NAME"]?>">
    View source code
</a></p>


<map name="map1">
<?=$imageMap?>
</map>

</body>
</html>

```

## Using Different Handlers for Different Data Points

In previous examples, all data points (that is, bars in chart, lines in line chart, sectors in pie chart,...) use the same image map handler.

For example, in a pie chart, if the image map is generated using `getHTMLImageMap` with “piestub.php” as the handler, then no matter what sectors the user clicks, the same handler “piestub.php” is used. The different sectors are distinguished by different HTTP query parameters.

If for some reason, it is desirable that different sectors use different handlers, the followings are some of the methods:

- Use the same handler for all sectors. In the handler, redirect the request to the “real handler” based on what sectors the user has clicked.
- Use a client side Javascript function as the handler, where “{sector}” is used as the argument to indicate which sector is clicked. For example, “javascript:callMe({sector});” can be used as the handler. In the “callMe” function, the client side Javascript can instruct the browser to load the “real handler”.
- Do not use the `getHTMLImageMap` method, but use the `getImageCoor` and `getLabelCoor` methods of the Sector object to obtain the image map coordinates of individual sectors and sector labels. With the image map coordinates, you may create `<area>` tags for the individual sectors and put whatever image map attributes you like. So you can set different “href” (that is, handler) for different sectors.

Although we use a pie chart as an example in the above description, these methods can apply to other chart types as well.

In general, if an entity has multiple clickable regions (e.g. all bars in a bar chart), the “getHTMLImageMap” method of that entity may be used to generate the <area> tags for all regions at once. These <area> tags will have similar structure.

On the other hand, if an entity only has one clickable region (e.g. a single sector or a text box), it is more convenient to enter the handler and tool tip text directly into the <area> tag for that object. In this case, the “getImageCoor” method of that entity may be used for getting the image map coordinates of the clickable region.

## Making Other Objects Clickable

Currently, most “dynamic objects” in ChartDirector have methods for generating image maps. On the other hand, ChartDirector may not provide image map methods for “static objects”. It is because they are not necessary.

“Static objects” are objects where the position and size are constants in a chart. For example, the plot area in an XYChart is a “static object”, because its position and size is pre-defined using the setPlotArea method. Thus the coordinates of the plot area are always. If you would like to response to mouse clicks on any point on the plot area, you can always produce an <area> tag for the plot area yourself.

Another example is if you have a custom shape (such as a company logo) on the chart and you want to make it clickable. Since the position and size of the custom shape is probably known in advanced, you can always produce an <area> tag for the custom shape yourself.

Note that ChartDirector image map support is “open-ended”. You can always insert your own <area> tags inside the image map.

---

# ChartDirector API Reference

## Important Notes

This ChartDirector API Reference is a common chapter for ChartDirector for PHP, Perl, Python and C++. Because of the differences among the languages, the followings should be noted when using this reference.

### Notes for Perl Developers

In ChartDirector for Perl, all constants are all under the namespace “perlchartdir”. For example, the constant “Transparent” should be used as “\$perlchartdir::Transparent” in scripts.

Many ChartDirector functions and methods accept “arrays” as arguments. However, in Perl, it is impossible to pass an “array” as argument without losing its structure. Therefore, these “array arguments” actually means “references to arrays” in ChartDirector for Perl.

### Notes for C++ Developers

#### *Method Overloading in C++*

The ChartDirector C++ API employs method overloading in many places. For example, there are three “addBarLayer” methods in the ChartDirector API.

However, method overloading is not supported in other ChartDirector supported languages, such as PHP, Perl and Python. In these languages, each method must have a different name. For example, the three “addBarLayer” methods in the C++ API are called “addBarLayer”, “addBarLayer2” and “addBarLayer3” in the PHP/Perl/Python APIs.

In this documentation, the method headings are named using the PHP/Perl/Python convention. This is for ease of referencing in this documentation since each heading is different.

Therefore, when calling the methods in C++, the method names in the “C++ Prototype” should be used rather than the names in the method heading. For example, the “addBarLayer3” method should be called as “addBarLayer” when using C++, as described in the “C++ Prototype” for the “addBarLayer3” method.

#### *Arrays in C++*

In the languages PHP, Perl and Python, an array object contains a property that describes its own length. However, in C++, an array is simply a pointer to the first position of the data block. It does not contain information about the size of the array.



To make the API consistent among languages, in ChartDirector's C++ API, special classes are used to represent the arrays. The classes "IntArray", "DoubleArray", "StringArray" and "MemBlock" represent an array of integers, an array of doubles, and array of strings, and an array of bytes (char).

Each class contains only two properties – a pointer to the actually C++ array, and the length of the array. The classes are defined as follows.

```
class IntArray
{
public :
    int len;
    const int *data;
    IntArray() : data(0), len(0) {}
    IntArray(const int *data, int len) : data(data), len(len) {}
};

class DoubleArray
{
public :
    int len;
    const double *data;
    DoubleArray() : data(0), len(0) {}
    DoubleArray(const double *data, int len) : data(data), len(len) {}
};

class StringArray
{
public :
    int len;
    const char * const *data;
    StringArray() : data(0), len(0) {}
    StringArray(const char * const *data, int len) : data(data), len(len) {}
};

class MemBlock
{
public :
    int len;
    const char *data;
    MemBlock() : data(0), len(0) {}
    MemBlock(const char *data, int len) : data(data), len(len) {}
};
```

You can easily use the constructor of the array classes to create a ChartDirector array object from from a C++ array. For example:

```
int data[100];
IntArray a = IntArray(data, 100);
```

The above will create an "IntArray" object representing the C++ array "data".

# International Character Sets Support

ChartDirector supports international character sets by supporting UTF8 encoding. UTF8 is an ASCII compatible standard for encoding Unicode characters.

## What is Unicode and What is UTF8?

Whereas ASCII is a standard of representing common alphanumeric character as a number in the range 0 – 127, Unicode is a standard of representing characters from all known languages as a number. Because there are many known languages in the world, a lot of numbers are needed.

One common way of encoding Unicode is to use two bytes (16 bits) for a single Unicode character. However, this scheme is incompatible with ASCII.

On the other hand, the UTF8 standard employs a variable number of bytes to represent one Unicode character. If the character is in the range 0 – 191, it just employs one byte. Since an ASCII string consists of one byte character in the range 0 – 127, it is also a valid UTF8 string. Thus UTF8 is compatible to ASCII.

## Writing Code in UTF8

If you write some code that contains international characters (that is, non-ASCII characters, such as Greek or Chinese characters), and you pass them to ChartDirector, you should save your source code in UTF8 format.

As UTF8 is a very popular format, many text editors support saving in UTF8 format. For example, the Notepad from Windows 2000 support saving file as UTF8.

If your code contains only ASCII characters, you can save it just as an ASCII file. It is because UTF8 is compatible with ASCII.

## ISO-8859-1 (Latin1) Western European Character Set

ChartDirector by default will assume text strings are of UTF8 format. However, if a string contains invalid UTF8 characters, ChartDirector will assume it is of ISO-8859-1 format.

ISO-8859-1 (also called Latin1 or Western European Character Set) is an extension of ASCII to include many Western European Characters in the range 128 - 255. This is the default encoding used in the English version of Windows.

Thus if you write some code that contains Latin1 characters, and you just save it as ASCII (or ANSI), it is quite probable ChartDirector can read it correctly. It is because a Latin1 string will probably contains invalid UTF8 characters, so ChartDirector will use the ISO-8859-1 interpretation for the string.

## Displaying International Characters

After ChartDirector understands the international characters, it needs to display it in chart images. To do that, it needs the proper fonts.

By default, the ChartDirector on Windows uses the Arial font (“arial.ttf”), which contains a lot of international characters already. If you are using a language that is not included in the Arial font, you

need to specify a font that contains your language. For example, if you are using the “Traditional Chinese” language, you may need to use the “minglui.ttc” font instead.

For ChartDirector on Linux/FreeBSD/Solaris, it will use the “arail.ttf” font if it is available. Otherwise, it will use the fonts that come with ChartDirector on Linux/FreeBSD/Solaris. These fonts only contain the ISO-8859-1 character set. If you need other fonts, you may need to download it yourself.

Please refer to the section on [Font Specification](#) on how ChartDirector handle fonts.

## Data Types

### Color Specification

Many functions in the ChartDirector API accept color as a parameter. ChartDirector supports colors specified in ARGB format, in which ARGB refers to the Alpha transparency, Red, Green and Blue components of the color.

In addition to ARGB colors, ChartDirector supports “dynamic” colors. A dynamic color is a color that changes depending on the position of the pixels. The “dynamic” colors that ChartDirector supports include “gradient colors”, “pattern colors” and “dash line colors”.

ChartDirector supports specifying colors indirectly using “palette colors”. When a “palette color” is used, the color is specified as an index to a palette. The actual color is looked up from the palette. The palette can contain both ARGB colors and “dynamic” colors. Palette colors and non-palette colors can be used simultaneously on the same ChartDirector image.

### ARGB Color

ARGB color is coded as a 32-bit number in 0xaaarrgbb format. The Alpha transparency component occupies the most significant 8 bits, the red component occupies the next 8 bits, the green component occupies the next 8 bits, and the blue component occupies the least significant 8 bits.

Each component ranges from 0 – 255, representing its intensity. For example, pure red color is 0x00ff0000, pure green color is 0x0000ff00, and pure blue color is 0x000000ff. White color is 0x00ffffff, while black color is 0x00000000.

Most software developers are probably familiar with the RGB color format. In ChartDirector, an additional alpha transparent component is added to support semi-transparent colors. If the alpha transparency component is zero, the color is fully opaque, and this becomes the same as the RGB format.

If alpha transparency is non-zero, the color is semi-transparent. That means if you use that color to draw something, the underlying background can still be seen. The larger the alpha transparency, the more transparent the color will become.

For example, 0x80ff0000 is a semi-transparent red color, while 0x00ff0000 is a fully opaque red color.

If alpha transparency is 255, the color is totally transparent. That means the color is invisible. It does not matter what the RGB components are. That means all totally transparent colors are the same. Therefore in ChartDirector, only one legal totally transparent color is used – 0xff000000. All other colors of the

form 0xffnnnnnn are reserved to represent palette colors and dynamic colors, and should not be interpreted as the normal ARGB colors.

The totally transparent color 0xff000000 is often used in ChartDirector to disable drawing something. For example, if you want to disable drawing the border of a rectangle, you can set the border color to totally transparent.

The ChartDirector API pre-defines a constant called “Transparent”, with is equivalent to 0xff000000.

## Pattern Color

A “pattern color” represents a repeated pattern. If you use a “pattern color” to fill a rectangle, the rectangle will be filled with the repeated pattern like tiling with a wallpaper.

Pattern colors are created using the [patternColor](#) method or the [patternColor2](#) method of the BaseChart object or DrawArea object. The first method creates a pattern color using an array of integers as a bitmap. The second method creates a pattern color by loading the pattern from an image file.

Both methods return a 32-bit integer. This number can be used in any ChartDirector API that expects a color as its input.

## Gradient Color

A “gradient color” is a color that changes progressively across a direction.

Gradient colors are created using the [gradientColor](#) method or the [gradientColor2](#) method of the BaseChart or DrawArea object. The first method creates a simple gradient color that contains a starting color and an ending color. The second method creates a multiple section gradient color that can have intermediate colors along the gradient direction.

Both methods return a 32-bit integer. This number can be used in any ChartDirector API that expects a color as its input.

Multiple section gradient colors are often used to create metallic or shiny effects. ChartDirector comes with several pre-defined constants for metal like gradient colors - gold, silver, red metallic, green metallic and blue metallic. For more details, please refer to the [gradientColor2](#) method.

## Dash Line Colors

A “dash line color” is a color that switches on and off periodically. When used to draw a line, the line will appear as a dash line.

Dash line colors are created using the [dashLineColor](#) method of the BaseChart object or DrawArea object. This method accepts a line color and a dash pattern code. The dash pattern code defines the style of the dash line.

The `dashLineColor` method returns a 32-bit integer. In theory, this number can be used in any ChartDirector API that expects a color as its input. In practice, however, it is usually only meaningful to use this number as line or border colors.

## Palette Colors

Palette colors are colors of the format 0xffffnnnn, where the least significant 16 bits (nnnn) are the index to the palette. A palette is simply a table of colors. For a palette color, the actual color is obtained by looking up the palette using the index. For example, the color 0xffff0001 is the second color in the palette color table (first color is index 0).

The colors in the palette can be normal colors of ARGB format, or any “dynamic” colors (pattern, gradient and dash line colors).

The first eight palette colors have special significance. The first three palette colors are the background color, default line color, and default text color of the chart. The 4<sup>th</sup> to 7<sup>th</sup> palette colors are reserved for future use. The 8<sup>th</sup> color is a special dynamic color that is equal to the fill color of the “associated data set”. Please refer to the table below for more details.

The 9<sup>th</sup> color (index = 8) onwards are used as the default colors for drawing the data sets. The 9<sup>th</sup> color is the default color for the 1<sup>st</sup> data set, the 10<sup>th</sup> color is for the 2<sup>nd</sup> data set, etc.

The ChartDirector API pre-defines several constants to facilitate using the color tables.

Name	Value	Description
Palette	0xffff0000	The starting point of the color palette table. The first palette color is “Palette + 0”, and the nth palette color is “Palette + n – 1”.
BackgroundColor	0xffff0000	The background color. ChartDirector always use the first palette color (Palette + 0) as the background color.
LineColor	0xffff0001	The default line color. ChartDirector uses the second palette color (Palette + 1) as the default line color.
TextColor	0xffff0002	The default text color. ChartDirector uses the third palette color (Palette + 1) as the default text color.
<Reserved>	0xffff0003 – 0xffff0006	These palette positions are reserved. Future versions of ChartDirector may use these palette positions for colors that have special significance.
SameAsMainColor	0xffff0007	A dynamic color that is equal to the fill color of the “associated data set”. This color is useful for objects that are associated with data sets. For example, in a pie chart, if the sector label background color is SameAsMainColor, its color will be the same as the corresponding sector fill color.

DataColor	0xffff0008	The starting point of the default data colors. Data colors are colors used to plot the data sets (e.g. the colors of the bars in bar charts). ChartDirector uses the ninth palette color (Palette + 8) as the starting point of the default data color. The first default data color is (DataColor + 0), and the nth default data color is (DataColor + n - 1).
-----------	------------	---

When a chart is created, it has a default palette color table. You may modify the color tables using the [setColor](#), [setColors](#), or [setColors2](#) methods of the BaseChart object

The advantages of using palette colors are that you can change the color schemes of the chart in one place. The ChartDirector comes with several pre-built palette color table for drawing charts on a white background, charts on a black background, and charts with “transparent” look and feel.

Name	Description
defaultPalette	The default palette color table. This palette is designed for drawing charts on white backgrounds (or lightly colored backgrounds).
whiteOnBlackPalette	A palette color table that is useful for drawing charts on black backgrounds (or darkly colored backgrounds).
transparentPalette	A palette color table for drawing charts on white backgrounds (or lightly colored backgrounds). The data colors in this palette are all semi-transparent colors.

## Font Specification

### Font File

In the ChartDirector API, a font is specified by specifying the file name that contains the font. For example, under the Windows platform, the “Arial” font is specified as “arial.ttf”, while the “Arial Bold” font is specified as “arialbd.ttf”.

ChartDirector on Windows does not come with any font files. Instead, it will automatically use the operating system’s font files in the “\windows\Fonts” subdirectory (where “\windows” is the directory where the operating system is installed). To see what fonts are installed, simply use the File Explorer to view that subdirectory.

For ChartDirector on Linux/FreeBSD/Solaris, it assumes the fonts files are in the “fonts” subdirectory under the subdirectory where the ChartDirector shared object “libchartdir.so” is installed. The ChartDirector on Linux/FreeBSD/Solaris distribution already includes some font files in that subdirectory.

If you need additional fonts, you may download them from numerous places in the web. A recommended location for downloading free high quality fonts is from Microsoft’s web site at <http://www.microsoft.com/OpenType/fontpack/default.htm>.

ChartDirector currently supports True Type fonts (.ttf) and Type 1 fonts (.pfa and .pfb), as well as Windows bitmap fonts (.fon).

Instead of using the default font directories mentioned above, you may also put your fonts in other directories and uses the full path name when specifying font files.

If you want the ChartDirector to search other subdirectories as the default font directories, you may define an environment variable called “FONTPATH” and list the subdirectories you want ChartDirector to search.

## Font Index

In practice, most font files contain only one font. However, in theory, a font file may contain more than one font. In this case, the font index can be used to specify which font to use.

By default, the font index is 0, which means the first font in the font file will be used.

## Font Size, Font Height and Font Width

The font size decides how big a font will appear in the image. The font size is expressed in a font unit called points. This is the same unit used in common word processors.

By default, when you specify a font size, both the font height and font width will be scaled by the same factor. The ChartDirector API also supports using different point size for font height and font width to create special effects. For example, the [setFontSize](#) method of the TextBox object allows you to specify different font height and font width.

## Font Color

This is the color to draw the font. (See [Color Specification](#) on how to specified colors.)

## Font Angle

This is the angle by which the font should be rotated anti-clockwise. By default, the angle is 0 degree, which means to draw the characters upright and layout the characters horizontally. An angle of 90 degrees would mean the characters are drawn sideways.

## Vertical Layout

This is a boolean flag to indicate whether the font should be layout horizontally or vertically.

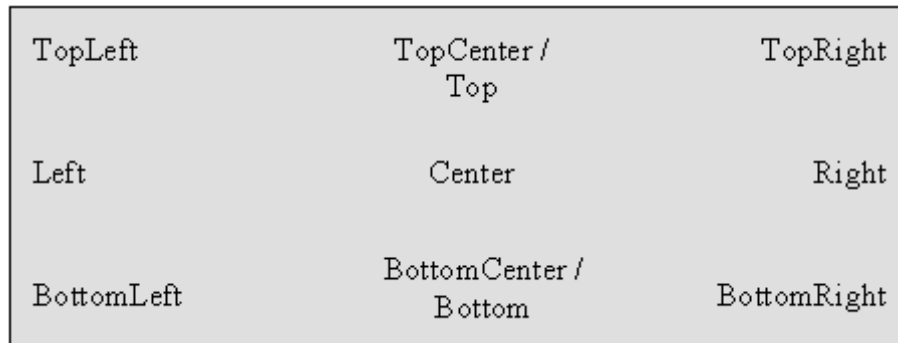
In horizontal layout, each additional character will be drawn on the right of the previous character. After all characters are drawn, the whole text will be rotated according to the font angle.

In vertical layout, each additional character will be drawn on the bottom of the previous character. After all characters are drawn, the whole text will be rotated according to the font angle.

Vertical layout is most often used for Oriental languages such as Chinese, Japanese and Korean languages.

## Alignment Specification

In a number of ChartDirector objects, you may specify the alignment of the object's content relative to its boundary. For example, for a TextBox object, you may specify the text's alignment relative to the box boundary by using the [setAlignment](#) method. The following diagram illustrates the location where the text will appear for different alignment options.



The ChartDirector API pre-defines several constants for the alignment options.

Name	Value	Description
BottomLeft	1	The leftmost point on the bottom line.
BottomCenter	2	The center point on the bottom line.
BottomRight	3	The rightmost point on the bottom line.
Left	4	The leftmost point on the middle horizontal line.
Center	5	The center point on the middle horizontal line.
Right	6	The rightmost point on the middle horizontal line.
TopLeft	7	The leftmost point on the top line.
TopCenter	8	The center point on the top line.
TopRight	9	The rightmost point on the top line.
Bottom	2	The center point on the bottom line. Same as BottomCenter.
Top	8	The center point on the top line. Same as TopCenter.

## No Value Specification

To draw charts, you need to supply data. In ChartDirector, data are supplied as an array of numbers.

In many practice cases, sometimes a data point will be missing. For example, suppose you are drawing a line chart with one data point per hour. It is possible that due to various reasons, some data points could be missing.

ChartDirector supports a special constant called "NoValue". You may use this constant to specify that a data point has no value (that is, missing).



By default, when ChartDirector sees that a data point has NoValue, it will draw nothing for that point. For a line chart, that means the line will become discontinuous.

ChartDirector line charts support an alternative method to handle NoValue data points. You can use a join line to jump over the NoValue point. The join line can be the same color and style as the line chart, or it can be of a different color and/or style (e.g. a dash line).

## Data and Text Formatting

ChartDirector charts often contain a lot of text strings. For example, the sector labels in pie charts, the axis labels, labels for the data points, the HTML image maps, etc., are all text strings.

### Parameter Substitution

ChartDirector uses “parameter substitution” to allow you to configure precisely the information contained in the text and their format.

For example, when drawing a pie chart in circular layout, the default sector label format in ChartDirector is:

```
“{label}\n{percent}%”
```

The “{label}” is a place holder for the name of the sector, while “{percent}” is a place holder for the percentage of the sector. In actually drawing the sector labels, ChartDirector will replace “{label}” with the actual sector name, and “{percent}” with the actual sector percentage.

So the above label format means the sector label will consist of two lines. The first line is the name of the sector, while the second line is the percentage of the sector with a “%” sign.

If you want to change the sector label format, you just need to change the above format string. For example, you can change it to:

- “{label}: US\${value}K ({percent}%)”

The above means that the sector label will consists of the name of the sector, followed by a colon and a space, followed by the value of the sector in US\$vvvvK format, followed by a space, followed by the percentage of the sector in brackets.

In general, for ChartDirector API that supports parameter substitution, the fields to be substituted should be enclosed in curly brackets “{” and “}”.

The following tables describe the fields available for parameter substitution for various kinds of chart objects.

#### Parameters for PieChart

Parameter	Description
sector	The sector number. The first sector is 0, while the n <sup>th</sup> sector is (n-1).

label	The text label of the sector.
value	The data value of the sector. (Supports <a href="#">numeric formatting</a> .)
percent	The percentage value of the sector. (Supports <a href="#">numeric formatting</a> .)

### Parameters for Bar, Line, Area and Scatter Chart Layers

Parameter	Description
x	The x position of the data point. For an enumerated x-axis (that is, data sets that do not have explicit x values), the first data point is 0, while the n <sup>th</sup> data point is (n-1). (Supports <a href="#">numeric formatting</a> .)
xLabel	The label of the data point on the bottom x-axis.
x2Label	The label of the data point on the top x-axis.
value	The value of the data point. (Supports <a href="#">numeric formatting</a> .)
accValue	The accumulative value of the data point. This is for charts that stack data up, such as stacked bar chart and stacked area chart. (Supports <a href="#">numeric formatting</a> .)
totalValue	The total value of all data points. This is for charts that stack data up, such as stacked bar chart and stacked area chart. (Supports <a href="#">numeric formatting</a> .)
dataSet	The data set number to which the data point belongs. The first data set is 0, while the n <sup>th</sup> data set is (n-1).
dataSetName	The data set name to which the data point belongs.
dataItem	The data point number within the data set. The first data point is 0, while the n <sup>th</sup> data point is (n-1). (Supports <a href="#">numeric formatting</a> .)

### Parameters for HLOC and CandleStick Chart Layers

Parameter	Description
x	The x position of the data point. For an enumerated x-axis (that is, data sets that do not have explicit x values), the first data point is 0, while the n <sup>th</sup> data point is (n-1). (Supports <a href="#">numeric formatting</a> .)
xLabel	The label of the data point on the bottom x-axis.
x2Label	The label of the data point on the top x-axis.
high	The high value of the data point. (Supports <a href="#">numeric formatting</a> .)
low	The low value of the data point. (Supports <a href="#">numeric formatting</a> .)
open	The open value of the data point. (Supports <a href="#">numeric formatting</a> .)
close	The close value of the data point. (Supports <a href="#">numeric formatting</a> .)

dataItem	The data point number within the data set. The first data point is 0, while the n <sup>th</sup> data point is (n-1). (Supports <a href="#">numeric formatting</a> .)
----------	--

### Parameters for Trend Chart Layers

Parameter	Description
dataSetName	The name of the trend layer.

### Parameters for Axis (XAxis and YAxis)

Parameter	Description
value	The axis value at the tick position. (Supports <a href="#">numeric formatting</a> .)

## Numeric Formatting

Many ChartDirector text labels contain numbers. For example, the sector labels in pie charts contain numbers representing the percentages and/or the values of the sectors.

ChartDirector supports formatting of the numbers by including numeric formatting options in the parameter substitution field. You can specify the precision, decimal point characters, thousand separator character and the negative sign character for a number.

For example, if you want a field “{value}” to have a precision of two decimal points, using ‘.’ (dot) as the decimal point, and using ‘,’ (comma) as the thousand separator, you may use the format “{value|2.,}”. The value 123456.789 will be displayed as “123,456.79”.

In general, the numeric formatting option has the following syntax:

{<value>|<a><b><c><d>}

where:

- <value> = the parameter name
- <a> = A number specifying the number of decimal points. Default is automatic.
- <b> = The decimal point character. Default is “.”.
- <c> = The thousand separator character. Default is none.
- <d> = The negative sign character. Default is “-”. The “~” character has a special meaning that no negative sign character should be used. In this case, negative numbers will look the same as positive numbers. This is occasionally useful for axis labeling.

Note that the separator character “|” and <a>, <b>, <c>, <d> are all optional. If you do not specify an option, the default value will be used. Note that you cannot leave out an optional character in the middle. For example, if you specify <a> and <c>, you must also specify <b>.

## Draw Objects

### Box

The class Box, as its name implies, represents a box. It is used as the base class for more complex objects, such as the [TextBox](#) and the [LegendBox](#).

Method	Description
<a href="#">setPos</a>	Set the coordinates of the top left corner of the box.
<a href="#">setSize</a>	Set the width and height of the box.
<a href="#">setBackground</a>	Set the background color, border color and 3D border effect of the box.
<a href="#">getImageCoord</a>	Get the coordinates of the box as HTML image map.

### setPos

PHP/Perl/Python Usage

setPos(x, y)

C++ Prototype

virtual void setPos(int x, int y) = 0;

Description

Set the coordinates of the top left corner of the box.

Arguments

Argument	Default Value	Description
x	(Mandatory)	The x coordinate of the left of the box.
y	(Mandatory)	The y coordinate of the top of the box.

Return Value

None

### setSize

PHP/Perl/Python Usage

setSize(w, h)

C++ Prototype

virtual void setSize(int w, int h) = 0;

Description

Set the width and height of the box.

#### Arguments

Argument	Default Value	Description
w	(Mandatory)	The width of the box in pixels.
h	(Mandatory)	The height of the box in pixels.

#### Return Value

None

## setBackground

PHP/Perl/Python Usage

setBackground(color [, edgeColor [, raisedEffect]])

#### C++ Prototype

virtual void setBackground(int color, int edgeColor = -1, int raisedEffect = 0) = 0;

#### Description

Set the background color, border color and 3D border effect of the box. By default, both the background and border colors are transparent, which means the box is invisible. See [Color Specification](#) for how colors are specified in ChartDirector.

#### Arguments

Argument	Default Value	Description
color	(Mandatory)	The background color of the box.
edgeColor	-1	The border color of the box.
raisedEffect	0	The width of the 3D border of the box. For positive values, the box will appear raised. For negative values, the box will appear depressed. A zero value means no 3D border should be drawn, that is, the box will appear as flat.

#### Return Value

None

## getImageCoor

PHP/Perl/Python Usage

getImageCoor()

#### C++ Prototype

virtual const char \*getImageCoor() = 0;

#### Description

Get the image map coordinates of the box as HTML image map attributes in the following format:

```
shape="rect" cords="<x1>,<y1>,<x2>,<y2>"
```

where (x1, y1) and (x2, y2) are opposite corners of the box. The format is specially designed so that it can easily be included into HTML image maps.

This method should be called only after you have completely drawn the chart using the [makeChart](#), [makeChart2](#) or [makeChart3](#) method. It is because ChartDirector needs to actually draw the chart to compute the coordinates of the box in some cases. For example, for a [TextBox](#) that is auto-sized, the size of the Box will automatically be adjusted to fix the text. ChartDirector needs to actually draw the text before knowing the exact size of the Box.

Arguments

None

Return Value

Return a text string representing the coordinates of the box in HTML image map attribute format.

## TextBox

The class TextBox, as its name implies, represents a text box. The TextBox class inherits from the Box class.

By default, the width and height of the text box are zero, which means they are automatically adjusted to fit the text. You can change the width and height by using the setSize method.

Method	Description
<a href="#">setText</a>	Sets the text to be shown in the text box.
<a href="#">setFontStyle</a>	Set the font of the text.
<a href="#">setFontSize</a>	Set the font size of the text.
<a href="#">setFontAngle</a>	Set the rotation angle of the text.
<a href="#">setFontColor</a>	Set the color of the text.
<a href="#">setMargin</a>	Set the left, right, top and bottom margins of the bounding box of the text.
<a href="#">setMargin2</a>	Short cut to set all four bounding box margins to the same value.
<a href="#">setAlignment</a>	Sets the alignment of the text relative to the container box
<b>Methods inherited from Box</b>	
<a href="#">setPos</a>	Set the coordinates of the top left corner of the box.
<a href="#">setSize</a>	Set the width and height of the box.
<a href="#">setBackground</a>	Set the background color and edge color of the box.
<a href="#">getImageCoor</a>	Get the coordinates of the box as HTML image map.

## setText

PHP/Perl/Python Usage

setText(text)

C++ Prototype

```
virtual void setText(const char *text) = 0;
```

Description

Sets the text to be shown in the text box. The text may contain multiple lines separated using the new line character “\n”.

Arguments

Argument	Default Value	Description
text	(Mandatory)	The text to be displayed in the text box.

Return Value

None

## setFontStyle

PHP/Perl/Python Usage

setFontStyle(font [, fontIndex])

C++ Prototype

```
virtual void setFontStyle(const char *font, int fontIndex = 0) = 0;
```

Description

Set the font of the text by specifying the file that contains the font. See [Font Specification](#) for details on various font attributes.

Arguments

Argument	Default Value	Description
font	(Mandatory)	The path name or file name of the <a href="#">font file</a> that contains the font.
fontIndex	0	The <a href="#">font index</a> of the font to use for font files that contains more than one font. An index of 0 means the first font.

Return Value

None

## setFontSize

PHP/Perl/Python Usage

setFontSize(fontHeight [, fontWidth])

C++ Prototype

```
virtual void setFontSize(double fontHeight, double fontWidth = 0) = 0;
```

Description

Set the font height and width. In most cases, only the fontHeight needs to be specified. The default value of the fontWidth is 0, which means the font width will be set to the same as the font height. See [Font Specification](#) for details on various font attributes.

Arguments

Argument	Default Value	Description
fontHeight	(Mandatory)	The <a href="#">font height</a> in a font unit called points.
fontWidth	0	The <a href="#">font width</a> in a font unit called points. If the font width is zero, it means the font width is the same as the font height.

Return Value

None

## setFontAngle

PHP/Perl/Python Usage

```
setFontAngle(angle [, vertical])
```

C++ Prototype

```
virtual void setFontAngle(double angle, bool vertical = false) = 0;
```

Description

Set the rotation of the text and the layout direction. By default, the rotation is 0 degrees and the layout direction is horizontal. See [Font Specification](#) for details on various font attributes.

Arguments

Argument	Default Value	Description
angle	(Mandatory)	The <a href="#">font rotation angle</a> . Rotation is measured in counter-clockwise direction in degrees.
vertical	0	Determine whether the font is <a href="#">layout</a> horizontally (from left to right) or vertically (from top to down). Vertical layout is common for oriental languages such as Chinese, Japanese and Korean. A “true” value (that is, any non-zero value) means vertical layout, while a “false” value means horizontal layout.

Return Value

None



## setFontColor

PHP/Perl/Python Usage  
setFontColor(color)

C++ Prototype  
virtual void setFontColor(int color) = 0;

### Description

Set the color of the text. By default, the color is the default TextColor as defined by the palette color table. See [Color Specification](#) for how colors are represented in ChartDirector.

### Arguments

Argument	Default Value	Description
color	(Mandatory)	The <a href="#">font color</a> .

### Return Value

None

## setMargin

PHP/Perl/Python Usage  
setMargin(m)

C++ Prototype  
virtual void setMargin(int m) = 0;

### Description

Sets all margins (left, right, top, and bottom) of the bounding box to the same value in pixels.

### Arguments

Argument	Default Value	Description
m	(Mandatory)	The left, right, top and bottom margins in pixels.

### Return Value

None

## setMargin2

PHP/Perl/Python Usage  
setMargin2(leftMargin, rightMargin, topMargin, bottomMargin)

C++ Prototype  
virtual void setMargin(int leftMargin, int rightMargin, int topMargin, int bottomMargin) = 0;

### Description

Set the margins of the bounding box in pixels. The margins are the distances between the borders of the boundary box to the text.

By default, the left and right margins are approximately half the font size, and the top and bottom margins are approximately ¼ of the font size.

#### Arguments

Argument	Default Value	Description
leftMargin	(Mandatory)	The left margin in pixels.
rightMargin	(Mandatory)	The right margin in pixels.
topMargin	(Mandatory)	The top margin in pixels.
bottomMargin	(Mandatory)	The bottom margin in pixels.

Return Value

None

## setAlignment

PHP/Perl/Python Usage

setAlignment(a)

C++ Prototype

```
virtual void setAlignment(Alignment a) = 0;
```

Description

Sets the alignment of the text relative to the container box.

#### Arguments

Argument	Default Value	Description
a	(Mandatory)	The alignment specification. See <a href="#">Alignment Specification</a> for possible alignment types.

Return Value

None

## Line

A Line, as its name implies, represents a straight line.

Method	Description
<a href="#">setPos</a>	Set the end points (x1, y1) and (x2, y2) of the line.
<a href="#">setColor</a>	Set the color of the line.
<a href="#">setWidth</a>	Set the width of the line in pixels.

## setPos

PHP/Perl/Python Usage

setPos(x1, y1, x2, y2)

C++ Prototype

virtual void setPos(int x1, int y1, int x2, int y2) = 0;

Description

Set the end points (x1, y1) and (x2, y2) of the line.

Arguments

Argument	Default Value	Description
x1	(Mandatory)	The x coordinate of the first end-point of the line.
y1	(Mandatory)	The y coordinate of the first end-point of the line.
x2	(Mandatory)	The x coordinate of the second end-point of the line.
y2	(Mandatory)	The y coordinate of the second end-point of the line.

Return Value

None

## setColor

PHP/Perl/Python Usage

setColor(c)

C++ Prototype

virtual void setColor(int c) = 0;

Description

Set the color of the line. By default, the color is the default LineColor as defined by the palette color table. See [Color Specification](#) for details of the palette color table.

Arguments

Argument	Default Value	Description
c	(Mandatory)	The color of the line.

Return Value

None

## setWidth

PHP/Perl/Python Usage

setWidth(w)

C++ Prototype

```
virtual void setWidth(int w) = 0;
```

Description

Set the width of the line in pixels. By default, the width is 1 pixel.

Arguments

Argument	Default Value	Description
w	(Mandatory)	The width (thickness) of the line in pixels.

Return Value

None

## BaseChart

BaseChart is an abstract class containing methods that are common to all chart types.

Method	Description
<a href="#"><u>destroy</u></a>	Destroy the chart. (Used in ChartDirector for C++ only).
<a href="#"><u>setSize</u></a>	Set the size of the chart to the specified width and height in pixels.
<a href="#"><u>setBackground</u></a>	Set the background color, border color and 3D border effect of the chart.
<a href="#"><u>setBorder</u></a>	Set the border color of the chart.
<a href="#"><u>setWallpaper</u></a>	Specify an image as the background wallpaper of the chart.
<a href="#"><u>setBgImage</u></a>	Specify an image as the background image of the chart.
<a href="#"><u>setTransparentColor</u></a>	Set the transparent color for the image when writing the image to an image file.
<a href="#"><u>addTitle</u></a>	Add a title to the chart on the <a href="#"><u>TopCenter</u></a> position of the chart
<a href="#"><u>addTitle2</u></a>	Add a title to the chart at the top, bottom, left or right position of the chart.
<a href="#"><u>addLegend</u></a>	Add a legend box to the chart.
<a href="#"><u>getDrawArea</u></a>	Returns the <a href="#"><u>DrawArea</u></a> object that the chart is drawn with to allow drawing custom text, line or shapes.
<a href="#"><u>addDrawObj</u></a>	Add a custom-developed <a href="#"><u>DrawObj</u></a> to the chart.
<a href="#"><u>addText</u></a>	Add a text box to the chart.
<a href="#"><u>addLine</u></a>	Add a line to the chart.
<a href="#"><u>setColor</u></a>	Change the color of the specified position in the palette color table.
<a href="#"><u>setColors</u></a>	Change the colors of the color table starting from the first color.
<a href="#"><u>setColors2</u></a>	Change the colors of the color table starting with the specified position in the palette color table.

<a href="#"><u>getColor</u></a>	Get the color of the specified position in the palette color table.
<a href="#"><u>dashLineColor</u></a>	Create a dash line color that is suitable for drawing dash lines.
<a href="#"><u>patternColor</u></a>	Create a pattern color that can be used to fill areas with repeating patterns. The pattern is defined using an array of colors as the bitmap.
<a href="#"><u>patternColor2</u></a>	Create a pattern color that can be used to fill areas with repeating patterns. The pattern is defined by loading from an image file.
<a href="#"><u>gradientColor</u></a>	Create a gradient color that can be used to fill areas with color gradients. The gradient color is defined using a gradient line segment with a starting color and an ending color.
<a href="#"><u>gradientColor2</u></a>	Create a gradient color that can be used to fill areas with color gradients. The gradient color is defined using a gradient line segment with multiple color points.
<a href="#"><u>layout</u></a>	Perform auto-scaling of the axis and compute the position of the various objects of the chart, without actually drawing the chart. This allows additional custom text or shapes to be added to the chart based on the positions of other objects.
<a href="#"><u>makeChart</u></a>	Generate the chart and save it into a file.
<a href="#"><u>makeChart2</u></a>	Generate the chart in memory.
<a href="#"><u>makeChart3</u></a>	Generate the chart in internal format and return a <a href="#"><u>DrawArea</u></a> object to allow adding custom drawings on top of the chart.
<a href="#"><u>getHTMLImageMap</u></a>	Generate an HTML image map for the chart contents. For a pie chart, this includes all the pie sectors. For an XYChart, it includes the image maps of all the chart layers (bar layers, line layers, area layers and HLOC layers).

## destroy

C++ Prototype

```
virtual void destroy() = 0;
```

Description

This method is for the C++ version of ChartDirector only. It is not required in other versions of ChartDirector.

Destroy the chart object. This method should be the last method to call for every chart object created to free up memory tied up by the object. After calling this method, the object is deleted and must not be any more.

Arguments

None

Return Value  
None

## setSize

PHP/Perl/Python Usage  
setSize(width, height)

C++ Prototype  
virtual void setSize(int width, int height) = 0;

Description  
Set the size of the chart to the specified width and height in pixels.

Arguments

Argument	Default Value	Description
width	(Mandatory)	The width of the chart in pixels.
height	(Mandatory)	The height of the chart in pixels.

Return Value  
None

## setBackground

PHP/Perl/Python Usage  
setBackground(color [, edgeColor [, raisedEffect]])

C++ Prototype  
virtual void setBackground(int bgColor, int edgeColor = -1, int raisedEffect = 0) = 0;

Description  
Set the background color, border color and 3D border effect of the chart. By default, the background color is white and the border color is transparent (which means no border). See [Color Specification](#) for how colors are represented in ChartDirector.

Arguments

Argument	Default Value	Description
color	(Mandatory)	The background color of the chart.
edgeColor	<a href="#">Transparent</a>	The border color of the chart.
raisedEffect	0	The width of the 3D border of the chart. For positive values, the chart will appear raised. For negative values, the chart will appear depressed. A zero value means no 3D border should be drawn, that is, the chart will appear as flat.

Return Value  
None

## setBorder

PHP/Perl/Python Usage  
setBorder(color)

C++ Prototype  
virtual void setBorder(int color) = 0;

Description  
Set the border color of the chart. By default, the border color is Transparent, which means the border is invisible.

Arguments

Argument	Default Value	Description
color	(Mandatory)	The border color of the chart.

Return Value  
None

## setWallpaper

PHP/Perl/Python Usage  
setWallpaper(img)

C++ Prototype  
virtual void setWallpaper(const char \*img) = 0;

Description  
Use the image loaded the specified file as the background wallpaper of the chart. The method will auto-detect the image file format using the file name extension, which must either be png, jpg, jpeg, gif, wbmp or wmp (case insensitive). If the image is smaller than the chart, the method will draw the image repetitively to fill up the whole chart.

Arguments

Argument	Default Value	Description
img	(Mandatory)	The image file that is used as the background wallpaper of the chart.

Return Value  
None

## setBgImage

PHP/Perl/Python Usage

setBgImage(img [, align])

C++ Prototype

```
virtual void setBgImage(const char *img, Alignment align = Center) = 0;
```

Description

Use the image loaded the specified file as the background image of the chart. The method will auto-detect the image file format using the file name extension, which must either be png, jpg, jpeg, gif, wbmp or wmp (case insensitive). The alignment of the image is controlled by the optional “align” argument. The default value of the “align” argument is Center. All [Alignment](#) values are supported.

Unlike the [setWallpaper](#) method, this method will not repetitively draw the image. Instead, it will only draw it once at the position determined by the “align” argument.

Arguments

Argument	Default Value	Description
img	(Mandatory)	The image file that is used as the background image of the chart.
align	Center	The alignment of the background image relative to the chart. See <a href="#">Alignment Specification</a> for possible alignment types.

Return Value

None

## setTransparentColor

PHP/Perl/Python Usage

setTransparentColor(c)

C++ Prototype

```
virtual void setTransparentColor(int c) = 0;
```

Description

Set the transparent color for the image when writing the image to an image file. This only applies if the image file format supports transparent color (such as GIF and PNG).

Arguments

Argument	Default Value	Description
c	(Mandatory)	The color that is designated as the transparent color.

Return Value

None



## addTitle

PHP/Perl/Python Usage

```
addTitle(text [, font [, fontSize [, fontColor [, bgColor [, edgeColor]]]])
```

C++ Prototype

```
virtual TextBox *addTitle(const char *text, const char *font = 0, double fontSize = 12, int fontColor = TextColor, int bgColor = Transparent, int edgeColor = Transparent) = 0;
```

Description

Add a title to the chart on the [TopCenter](#) position of the chart. For other positions, use the alternative form of [addTitle2](#) method.

The “text” argument contains the title text. Titles with multiple lines are supported by separating the lines with the new line character (‘\n’).

By default, the title will be drawn using the Arial bold font at font size of 12 points using the default TextColor. These can be changed by using the optional “font”, “fontSize” and “fontColor” arguments. (See [Font Specification](#))

The title is contained within a box, of which the width is the same as the width of the chart, and the height is variable depending on the font size and the number of lines the title has. By default, the box has a transparent background color and a transparent edge color, so it is invisible. These can be change by using the optional “bgColor” and “edgeColor” arguments. (See [Color Specification](#))

Arguments

Argument	Default Value	Description
text	(Mandatory)	The text for the title.
font	0	The font to be used for the title text.
fontSize	12	The font size in points for the title text.
fontColor	TextColor	The color of the title text.
bgColor	Transparent	The background color of the title box.
edgeColor	Transparent	The border color of the title box.

Return Value

This method returns a [TextBox](#) object representing the title to allow fine-tuning of the title appearance.

## addTitle2

PHP/Perl/Python Usage

```
addTitle2(alignment, text [, font [, fontSize [, fontColor [, bgColor [, edgeColor]]]])
```

C++ Prototype

```
virtual TextBox *addTitle(Alignment alignment, const char *text, const char *font = 0, double fontSize = 12, int fontColor = TextColor, int bgColor = Transparent, int edgeColor = Transparent) = 0;
```

#### Description

Add a title to the chart. This method is similar as the [addTitle](#) method, except that the first argument “[alignment](#)” can be used to control where the title is drawn. You can add more than one title to the chart.

If the “[alignment](#)” is set to “Left “ or “Right”, the title is rotated 90 degrees (that is, drawn vertically).

This method returns a [TextBox](#) object representing the title to allow fine-tuning of the title appearance.

For the explanations of the arguments, please refer to the [addTitle](#) method.

#### Arguments

Argument	Default Value	Description
alignment	(Mandatory)	The position of the title relative to the chart. The supported alignments are Top, Bottom, Left and Right. See <a href="#">Alignment Specification</a> for the meaning of the various alignment types.
Text	(Mandatory)	The text for the title.
Font	0	The font to be used for the title text.
fontSize	12	The font size in points for the title text.
fontColor	TextColor	The color of the title text.
bgColor	Transparent	The background color of the title box.
edgeColor	Transparent	The border color of the title box.

#### Return Value

This method returns a [TextBox](#) object representing the title to allow fine-tuning of the title appearance.

## addLegend

#### PHP/Perl/Python Usage

```
addLegend(x, y [, vertical [, font [, fontSize]]])
```

#### C++ Prototype

```
virtual LegendBox *addLegend(int x, int y, bool vertical = true, const char *font = 0, double fontSize = 10) = 0;
```

#### Description

Add a legend box to the chart. The (x, y) arguments specify the top left corner of the legend box. The optional “vertical” argument is a boolean flag to indicate whether the keys in the legend box will be layout vertically or horizontally. The optional “font” and “fontSize” arguments specify the font and font size. The default font is Arial with font size of 10 points.

By default, if you specify the vertical layout, the legend box will have a boundary drawn using the default LineColor, while there will be no such boundary for horizontal layout.

This method returns a [LegendBox](#) object, which you may use to fine-tune the appearance of the legend box.

#### Arguments

Argument	Default Value	Description
x	(Mandatory)	The x coordinate of the left of the legend box.
y	(Mandatory)	The y coordinate of the top of the legend box.
vertical	1	Determines whether the keys inside the legend box are layout vertically or horizontally. A “true” (that is, non-zero) value means the keys are layout vertically. A “false” value means the keys are layout horizontally.
font	""	The font file for the legend font. See <a href="#">Font Specification</a> for how fonts are specified in ChartDirector.
fontSize	10	The font size of the legend font.

#### Return Value

This method returns the [LegendBox](#) object, which you may use to fine-tune the appearance of the legend box.

## getDrawArea

PHP/Perl/Python Usage

getDrawArea()

C++ Prototype

```
virtual DrawArea *getDrawArea() = 0;
```

#### Description

Return the [DrawArea](#) object. The DrawArea object provides basic graphics operations such as drawing text, lines, circles, polygons, etc. It is the tool used by ChartDirector to draw all the charts.

This object is made accessible to allow adding custom drawings to the chart or even developing custom chart types.

The custom drawings will be drawn before drawing the chart. That means the custom drawings will be at the bottom of the chart. If you want the custom drawings to be on top of the chart, get the DrawArea object using the [makeChart3](#) method instead.

Note that when you call getDrawArea, the original background of the chart will be “frozen”. For example, you cannot change the background color of the chart after you called getDrawArea. It is because changing the background will overwrite anything that you would have drawn using the DrawArea object. To avoid this, the background is automatically “frozen” when the getDrawArea method is called.

Arguments  
None.

Return Value

This method returns a [DrawArea](#) object that can be used to add custom text and shapes to the chart.

## addDrawObj

PHP/Perl/Python Usage

addDrawObj(obj)

C++ Prototype

```
virtual DrawObj *addDrawObj(DrawObj *obj) = 0;
```

Description

Add a custom-developed [DrawObj](#) to the chart.

Arguments

Argument	Default Value	Description
obj	(Mandatory)	The DrawObj to be added to the chart.

Return Value

This method returns the same [DrawObj](#) that is passed in as the argument.

## addText

PHP/Perl/Python Usage

addText(x, y, text [, font [, fontSize [, fontColor [, alignment [, angle [, vertical]]]]]])

C++ Prototype

```
virtual TextBox *addText(int x, int y, const char *text, const char *font = 0, double fontSize = 8, int  
fontColor = TextColor, Alignment alignment = TopLeft, double angle = 0, bool vertical = false) = 0;
```

Description

Add a text box to the chart. Return a [TextBox](#) object that represents the text box added. By default, only the text is visible, the box is transparent and therefore invisible. You may use the methods of the returned TextBox object to change the appearance of the text box.

Arguments

Argument	Default Value	Description
x	(Mandatory)	The x coordinate of the top left corner of the text box.
y	(Mandatory)	The y coordinate of the top left corner of the text box.
text	(Mandatory)	The text to shown in the text box.

font	""	The font used to draw the text. An empty string means using the default font (Arial). See <a href="#">Font Specification</a> for how fonts are specified in ChartDirector.
fontSize	8	The font size used to draw the text.
fontColor	TextColor	The color used to draw the text.
alignment	TopLeft	The alignment of the text within the text box.
angle	0	The rotation angle of the text within the text box.
vertical	0	A flag to indicate whether the text should be layout vertically or horizontally (default).

#### Return Value

This method returns the [TextBox](#) object represented the text box added. You may use the methods of this object to fine-tune the appearance of the text box.

## addLine

PHP/Perl/Python Usage

```
addLine(x1, y1, x2, y2 [, color [, lineWidth]])
```

C++ Prototype

```
virtual Line *addLine(int x1, int y1, int x2, int y2, int color = LineColor, int lineWidth = 1) = 0;
```

#### Description

Add a line to the chart. Return a [Line](#) object that represents the line added. You may use the methods of the returned Line object to change the appearance of the line.

#### Arguments

Argument	Default Value	Description
x1	(Mandatory)	The x coordinate of the first endpoint of the line.
y1	(Mandatory)	The y coordinate of the first endpoint of the line.
x2	(Mandatory)	The x coordinate of the second endpoint of the line.
y2	(Mandatory)	The y coordinate of the second endpoint of the line.
color	LineColor	The color of the line.
lineWidth	1	The width of the line.

#### Return Value

This method returns the [Line](#) object represented the line added. You may use the methods of this object to fine-tune the appearance of the line.

## setColor

PHP/Perl/Python Usage  
setColor(paletteEntry, color)

C++ Prototype  
virtual void setColor(int paletteEntry, int color) = 0;

Description  
Change the color of the specified position in the palette color table. (See the section on [Color Specification](#) on the details of the palette color table.)

### Arguments

Argument	Default Value	Description
paletteEntry	(Mandatory)	The index to the palette color table.
color	(Mandatory)	The color to change to.

Return Value  
None

## setColors

PHP/Perl/Python Usage  
setColors(colors)

C++ Prototype  
virtual void setColors(const int \*colors) = 0;

Description  
Change the colors of the palette color table starting from the first color. This method is typically used to change the entire palette color table. (See the section on [Color Specification](#) on the details of the palette color table.)

### Arguments

Argument	Default Value	Description
colors	(Mandatory)	A list of colors to change to.

Return Value  
None

## setColors2

PHP/Perl/Python Usage  
setColors2(paletteEntry, colors)

C++ Prototype  
virtual void setColors(int paletteEntry, const int \*colors) = 0;

#### Description

Change the colors of the color table starting with the specified position in the palette color table. See the section on [Color Specification](#) on the details of the palette color table.

#### Arguments

Argument	Default Value	Description
paletteEntry	(Mandatory)	The index to the color table.
colors	(Mandatory)	A list of colors to change to.

#### Return Value

None

## getColor

PHP/Perl/Python Usage

getColor(paletteEntry)

C++ Prototype

```
virtual int getColor(int paletteEntry) = 0;
```

#### Description

Get the color of the specified position in the palette color table. (See the section on [Color Specification](#) on the details of the color table.)

#### Arguments

Argument	Default Value	Description
paletteEntry	(Mandatory)	The index to the color table.

#### Return Value

Return the color of the specified position in the color table.

## dashLineColor

PHP/Perl/Python Usage

dashLineColor(color, patternCode)

C++ Prototype

```
virtual int dashLineColor(int color, int patternCode) = 0;
```

#### Description

A “dash line color” is a color that switches on and off periodically. When used to draw a line, the line will appear as a dash line.

This method is the same as the [dashLineColor](#) method of the DrawArea object. Please refer to the dashLineColor method of the DrawArea object for detail information on how to use the patternCode to specify the style of the dash line.

#### Arguments

Argument	Default Value	Description
color	(Mandatory)	The color to draw the dash line.
patternCode	(Mandatory)	A 32-bit integer representing the style of the dash line.

#### Return Value

Return a 32-bit integer acting as a “handle” to the dash line color.

## patternColor

PHP/Perl/Python Usage

patternColor(colorArray, height [, startX, startY])

#### C++ Prototype

virtual int patternColor([IntArray](#) c, int h, int startX = 0, int startY = 0) = 0;

#### Description

A pattern color is a “dynamic” color that changes according to a 2D periodic pattern. When it is used to fill an area, the area will look like being tiled with a wallpaper pattern.

The patternColor method accepts an array of colors as the bitmap pattern. If you would like to use a pattern from an image file, use the [patternColor2](#) method instead.

This method is the same as the [patternColor](#) method of the DrawArea object. Please refer to the patternColor method of the DrawArea object for detail information on how to use this method.

#### Arguments

Argument	Default Value	Description
colorArray	(Mandatory)	An array of 32-bit integers representing the colors of the pixels of the bitmap pattern. The color of the pixel at (x, y) should correspond to index (x + y * width - 1) of the array.
height	(Mandatory)	The height of the bitmap pattern in pixels.
startX	0	The x coordinate of a reference point to align the pattern.
startY	0	The y coordinate of a reference point to align the pattern.

#### Return Value

Return a 32-bit integer acting as a “handle” to the pattern color.

## patternColor2

PHP/Perl/Python Usage

patternColor(filename [, startX, startY])

#### C++ Prototype

virtual int patternColor(const char \*filename, int startX = 0, int startY = 0) = 0;



#### Description

A pattern color is a “dynamic” color that changes according to a 2D periodic pattern. When it is used to fill an area, the area will look like being tiled with a wallpaper pattern.

The `patternColor2` method is used to create the pattern color using an image file as the pattern. If you would like to use a pattern from memory, use the [patternColor](#) method instead.

This method is the same as the [patternColor2](#) method of the `DrawArea` object. Please refer to the `patternColor2` method of the `DrawArea` object for detail information on how to use this method.

#### Arguments

Argument	Default Value	Description
filename	(Mandatory)	An image file providing the pattern.
startX	0	The x coordinate of a reference point to align the pattern.
startY	0	The y coordinate of a reference point to align the pattern.

#### Return Value

Return a 32-bit integer acting as a “handle” to the pattern color.

## gradientColor

PHP/Perl/Python Usage

`gradientColor(startX, startY, endX, endY, startColor, endColor)`

C++ Prototype

`virtual int gradientColor(int startX, int startY, int endX, int endY, int startColor, int endColor) = 0;`

#### Description

A gradient color is a “dynamic” color that changes progressively from one color to other colors along a direction. A simple gradient color contains two colors (a starting color and an ending color) along the direction, while a complex gradient color may contain many intermediate colors along the direction.

The `gradientColor` method creates a simple gradient color. To create a complex gradient color, please refer to the [gradientColor2](#) method.

This method is the same as the [gradientColor](#) method of the `DrawArea` object. Please refer to the `gradientColor` method of the `DrawArea` object for detail information on how to use this method.

#### Arguments

Argument	Default Value	Description
startX	(Mandatory)	The x coordinate of the starting point of the gradient line segment.
startY	(Mandatory)	The y coordinate of the starting point of the gradient line segment.

endX	(Mandatory)	The x coordinate of the ending point of the gradient line segment.
endY	(Mandatory)	The y coordinate of the ending point of the gradient line segment.
startColor	(Mandatory)	The starting color.
endColor	(Mandatory)	The ending color.

Return Value

Return a 32-bit integer acting as a “handle” to the gradient color.

## gradientColor2

PHP/Perl/Python Usage

gradientColor2(colorArray [, angle [, scale [, startX, startY]]])

C++ Prototype

```
virtual int gradientColor(const int *colorArray, double angle = 90, double scale = 1, int startX = 0, int startY = 0) = 0;
```

Description

A gradient color is a “dynamic” color that changes progressively from one color to other colors along a direction. A simple gradient color contains two colors (a starting color and an ending color) along the direction, while a complex gradient color may contain many intermediate colors along the direction.

The gradientColor2 method creates a complex gradient color. To create a simple gradient color, please refer to the [gradientColor](#) method.

This method is the same as the [gradientColor2](#) method of the DrawArea object. Please refer to the gradientColor2 method of the DrawArea object for detail information on how to use this method.

Arguments

Argument	Default Value	Description
colorArray	(Mandatory)	An array defining the positions and colors of the pixels along the gradient line. See above for description.
angle	90	The direction of the gradient line segment in degrees. The default direction is horizontal from left to right (90 degrees).
scale	1.0	The scaling factor for the gradient line segment. By default, the gradient line segment is 256 pixels in length. The scaling factor can be use to stretch or compress the gradient line segment during drawing.
startX	0	The x coordinate of the starting point of the gradient line segment.

startY	0	The y coordinate of the starting point of the gradient line segment.
--------	---	--

#### Return Value

Return a 32-bit integer acting as a “handle” to the gradient color.

## layout

PHP/Perl/Python Usage

layout()

C++ Prototype

```
virtual void layout() = 0;
```

#### Description

Perform auto-scaling of the axis and compute the position of the various objects of the chart.

This method is typically used when custom objects needs to be added to the chart, and the position of the custom objects depends on the scale of the axis. In this case, the layout method needs to be called first to determine the scale of the axis.

An example is to draw a custom label on the maximum value point of a data line. The application knows the maximum value (since the data set is supplied by the caller), but it does not know the coordinate of the maximum value. To calculate the coordinate correctly, it needs to call the layout method to auto-scale the axis first, and then call the getXCoor and getYCoor methods of the Layer object to get the coordinates.

After drawing the custom objects, the application can call [makeChart](#) to generate the chart.

If you just want to generate the chart, you do not need to call the layout method. You can call the makeChart method directly. The makeChart method will automatically call the layout method if it is not already called.

#### Arguments

None

#### Return Value

None

## makeChart

PHP/Perl/Python Usage

makeChart(filename)

C++ Prototype

```
virtual bool makeChart(const char *filename) = 0;
```

#### Description

Generate the chart and save it into a file. The formats supported are PNG, JPG, JPEG, alternative GIF and WBMP. The actual format used depends on the extension of the filename, which should be png, jpg, jpeg, gif, wbmp or wmp (case insensitive). If the extension is none of the above, the PNG format will be used.

If you want to generate the chart in memory (e.g. for directly output to the network), use the [makeChart2](#) method instead.

#### Arguments

Argument	Default Value	Description
filename	(Mandatory)	The name of the file to save the image.

#### Return Value

Return 1 (true) if no error, otherwise returns 0 (false).

## makeChart2

PHP/Perl/Python Usage

`makeChart2(format)`

#### C++ Prototype

virtual [MemBlock](#) makeChart(int format) = 0;

#### Description

Generate the chart in memory. The formats supported are PNG, JPG, JPEG, alternative GIF and WBMP.

If you want to generate the chart to a file, use the [makeChart](#) method instead.

#### Arguments

Argument	Default Value	Description
format	(Mandatory)	The format of the image. It must be one of the followings constants: <ul style="list-style-type: none"><li>• PNG</li><li>• GIF</li><li>• JPG</li><li>• WMP</li></ul>

#### Return Value

Returns a block of memory (as a string in PHP/Perl/Python, and as a MemBlock in C++) that contains the binary image of the chart in the requested format.

## makeChart3

PHP/Perl/Python Usage

makeChart3()

C++ Prototype

```
virtual DrawArea *makeChart() = 0;
```

Description

Generate the chart in internal format and return a [DrawArea](#) object to allow you to add custom drawings on top of the chart. The DrawArea object provides basic graphics operations such as drawing text, lines, circles, polygons, etc. The resulting chart can then be output using the [makeChart](#) or [makeChart2](#) method.

If you want to add custom drawings at the bottom of the chart, use the [getDrawArea](#) method to obtain the DrawArea object instead.

Arguments

None

Return Value

This method returns a [DrawArea](#) object that can be used to add custom text and shapes to the chart.

## getHTMLImageMap

PHP/Perl/Python Usage

getHTMLImageMap(url [, queryFormat [, extraAttr]])

C++ Prototype

```
virtual const char *getHTMLImageMap(const char *url, const char *queryFormat = 0, const char *extraAttr = 0) = 0;
```

Description

Generate an HTML image map for the chart contents. For a pie chart, this includes all the pie sectors. For an XYChart, it includes the image maps of all the chart layers.

This method does not include the image maps for the legend box, title box or custom text box. To generate an HTML image map for the legend box, use the [getHTMLImageMap](#) method of the LegendBox object. To generate an HTML image map for the title box or custom text box, please refer to the [getImageCoor](#) method of the TextBox object. If you just want to generate an image map for a particular layer for an XYChart, use the [getHTMLImageMap](#) method of the Layer object instead.

The getHTMLImageMap method should be called only after you have completely drawn the chart using the [makeChart](#), [makeChart2](#) or [makeChart3](#) method. It is because ChartDirector needs to actually draw the chart to compute the coordinates of the various entities in the image map.

The getHTMLImageMap method accepts a URL as its argument. It generates an image map using the URL appended with parameters to indicate which part of the chart the user has clicked. The parameters are appended using standard URL syntax, that is, in the following format:

attr1=value1&attr2=value2&attr3=value3....

The parameters will be appended to the URL using the “?” character as the separator. However, if the given URL already contains the “?” character (that is, it already have some parameters), then the “&” character will be used as the separator.

The following is an example of the HTML image map generated for a bar chart with five bars.

```
<area shape="rect" coords="34,219,63,139"
  href="myurl.php?x=0&xLabel=Mon&dataSet=0&dataSetName=Server #1&value=100">

<area shape="rect" coords="74,219,103,119"
  href="myurl.php?x=1&xLabel=Tue&dataSet=0&dataSetName=Server #1&value=125">

<area shape="rect" coords="114,219,143,22"
  href="myurl.php?x=2&xLabel=Wed&dataSet=0&dataSetName=Server #1&value=245.78">

<area shape="rect" coords="154,219,183,101"
  href="myurl.php?x=3&xLabel=Thu&dataSet=0&dataSetName=Server #1&value=147">

<area shape="rect" coords="194,219,223,165"
  href="myurl.php?x=4&xLabel=Fri&dataSet=0&dataSetName=Server #1&value=67">
```

The image map consists of multiple <area> tags, one for each bar in the chart. The “href” attribute of the image map consists of a URL appended with parameters to indicate which bar the user has clicked, and to provide additional information regarding the clicked bar. The image map does not include the “<map>” or “</map>” tag. This is intentional so that you can add additional <area> tags to the image map.

The format of the appended URL parameters is determined using the optional queryFormat argument of the getHTMLImageMap method. If no queryFormat argument is specified, the default queryFormat depends on the chart and layer types as shown in the following table.

### Default Query Format

Chart / Layer Type	Default Query Format
Pie chart	sector={sector}&label={label}&value={value}&percent={percent}
Bar, Line, Area and Scatter chart layers	x={x}&xLabel={xLabel}&dataSet={dataSet}&dataSetName={dataSetName}&value={value}
HLOC and CandleStick chart layers	x={x}&xLabel={xLabel}&high={high}&low={low}&open={open}&close={close}
Trend chart layer	dataSetName={dataSetName}

The texts in curly brackets (e.g. {sector}, {dataSet}, etc.) will be replaced by the actual values when generating the image map using “parameter substitution”. For example, {sector} will be replaced by the sector number of the sector clicked by the user.

Please refer to the section on [Data and Text Formatting](#) for the available parameters and their meanings. The url, queryFormat and extraAttr arguments (described below) all support parameter substitution.

Using the queryFormat argument, you may add additional URL parameters to provide more information, remove unused URL parameters to reduce the image map size, or change the names of the parameters. For example, if you use “x={x}&v={value}” as the queryFormat for an XYChart, only the x position and the data value of the clicked bar will be included as URL parameters, and they will be called “x” and “v” respectively.

In addition to customizing the URL parameters, ChartDirector supports additional HTML attributes in the <area> tags using the extraAttr argument of the getHTMLImageMap method. For example, you can add an “alt” HTML attribute to provide “tool tip” when the mouse moves over the bars. The following extraAttr:

```
alt='{xlabel}:{value}'
```

will add an “alt” attribute to the image map. It will cause a “tool tip” to appear when the mouse moves over the data representation (that is, the bars in a bar chart, or the lines in a line chart, etc.). The “tool tip” will contain the x-axis label and the data value of the data point separated by a colon.

Another common usage of the extraAttr argument is to add “onmouseover” and “onmouseout” HTML attributes to handle user interaction using Javascript on the browser.

#### Arguments

Argument	Default Value	Description
url	(Mandatory)	The URL to be included as the “href” attribute of the image map. ChartDirector will append additional parameters to the URL to indicate which data point the user has clicked. If you do not use any URL, you may simply use an empty string.
queryFormat	""	A text string to specify the format of the parameters to be appended to the URL. If this argument is an empty string, a default queryFormat will be used. If you do not need any query parameter, use a space character as the queryFormat.
extraAttr	""	A text string to specify additional attributes to add to the <area> tag.

#### Return Value

Return a text string containing the image map generated.

## LegendBox

The class LegendBox represents a legend box. It is a subclass of [TextBox](#), which in turn is a subclass of [Box](#).

To create a legend box and add it to a chart, use the [addLegend](#) method of the BaseChart class. It will return a LegendBox object representing the legend box being created. You may use the LegendBox object to fine-tune the appearance of the legend box.

ChartDirector will automatically add every named data set in the chart to the legend box. You may add additional entry to the legend box by using the [addKey](#) method of the LegendBox object.

Method	Description
<a href="#">addKey</a>	Add an additional entry to the legend box.
<a href="#">setText</a>	Set the format of the legend key text – override setText in TextBox superclass (applicable for pie chart only)
<a href="#">setKeySize</a>	Set the size of the color box and gap between the color box and the text.
<a href="#">getImageCoor2</a>	Get the image map coordinates of a legend key as HTML image map attributes.
<a href="#">getHTMLImageMap</a>	Generate an HTML image map for add the keys in the legend box.
<b>Methods inherited from TextBox</b>	
<a href="#">setFontStyle</a>	Set the font of the text.
<a href="#">setFontSize</a>	Set the font size of the text.
<a href="#">setFontAngle</a>	Set the rotation angle of the text.
<a href="#">setFontColor</a>	Set the color of the text.
<a href="#">setMargin</a>	Set the left, right, top and bottom margins of the bounding box of the text.
<a href="#">setMargin2</a>	Short cut to set all four bounding box margins to the same value.
<a href="#">setAlignment</a>	Set the alignment of the text relative to the container box
<b>Methods inherited from Box</b>	
<a href="#">setPos</a>	Set the coordinates of the top left corner of the box.
<a href="#">setSize</a>	Set the width and height of the box.
<a href="#">setBackground</a>	Set the background color and edge color of the box.
<a href="#">getImageCoor</a>	Get the coordinates of the box as HTML image map.

## addKey

PHP/Perl/Python Usage

addKey(text, color)

C++ Prototype

virtual void addKey(const char \*text, int color) = 0;



#### Description

Add an additional entry to the legend box.

#### Arguments

Argument	Default Value	Description
text	(Mandatory)	The text of the legend box entry.
color	(Mandatory)	The color of the legend box entry.

#### Return Value

None

## setText

PHP/Perl/Python Usage

setText(text)

#### C++ Prototype

```
virtual void setText(const char *text) = 0;
```

#### Description

Set the format of the legend key text. This method is useful for pie charts only. It is ignored in other chart types.

The default legend key text for a pie chart is "{label}", which means the legend key text for a sector is the text label for that sector.

The legend key text can be modified to include more information. For example, "{label}: {value}" means the legend key text for a sector will consist of its text label, followed by a color character and a space character, followed by its data value.

Please refer to the Pie Chart parameter table on the section [Data and Text Formatting](#) for the supported fields and their meanings.

#### Arguments

Argument	Default Value	Description
text	(Mandatory)	The legend key text format specification.

#### Return Value

None

## setKeySize

PHP/Perl/Python Usage

setKeySize(width [, height [, gap]])

C++ Prototype

```
virtual void setKeySize(int width, int height = -1, int gap = -1) = 0;
```

Description

Each legend entry consists of a color box (or a data symbol) and a text description. By default, the size of the color box and the gap between the color box and the text are determined automatically based on the font size of the legend text. Use this method to manually configure the size of the color box and gap between the color box and the text.

Arguments

Argument	Default Value	Description
width	(Mandatory)	The width of the color box in pixels.
height	-1	The height of the color box in pixels. A value of -1 means the height is determined automatically.
gap	-1	The gap between the color box and the text in pixels. A value of -1 means the gap is determined automatically.

Return Value

None

## getImageCoor2

PHP/Perl/Python Usage

```
getImageCoor2(dataItem)
```

C++ Prototype

```
virtual const char *getImageCoor(int dataItem) = 0;
```

Description

Get the image map coordinates of a legend key as HTML image map attributes in the following format:

```
shape="rect" cords="<x1>, <y1>, <x2>, <y2>"
```

where (x1, y1) and (x2, y2) are opposite corners of the box that enclosed the legend key. The legend key includes the colored box followed by the text label for that color. The format is specially designed so that it can easily be included into HTML image maps.

The legend key to use is specified using the dataItem argument. The first legend key is 0, while the n<sup>th</sup> legend key is (n-1).

If you want to get the complete image map for the entire legend box, use the [getHTMLImageMap](#) method of the LegendBox object instead.

This method should be called only after you have completely drawn the chart using the [makeChart](#), [makeChart2](#) or [makeChart3](#) method. It is because ChartDirector needs to actually draw the chart to compute the coordinates of the legend keys.

## Arguments

Argument	Default Value	Description
dataItem	(Mandatory)	The text of the legend box entry.

## Return Value

Return a text string representing the coordinates of the box in HTML image map attribute format.

## getHTMLImageMap

PHP/Perl/Python Usage

getHTMLImageMap(url [, queryFormat [, extraAttr]])

## C++ Prototype

```
virtual const char *getHTMLImageMap(const char *url, const char *queryFormat = 0, const char *extraAttr = 0) = 0;
```

## Description

Generate an HTML image map for add the keys in the legend box.

If you want to generate an HTML image map for the chart contents, use the [getHTMLImageMap](#) method of the BaseChart object instead.

The getHTMLImageMap method should be called only after you have completely drawn the chart using the [makeChart](#), [makeChart2](#) or [makeChart3](#) method. It is because ChartDirector needs to actually draw the chart to compute the coordinates of the various entities in the image map.

The getHTMLImageMap method accepts a URL as an argument. It generates an image map using the URL appended with parameters to indicate which legend key the user has clicked. The parameters are appended using standard URL syntax, that is, in the following format:

```
attr1=value1&attr2=value2&attr3=value3....
```

The parameters will be appended to the URL using the “?” character as the separator. However, if the given URL already contains the “?” character (that is, it already have some parameters), then the “&” character will be used as the separator.

The following is an example of the HTML image map generated for a legend box with 3 keys.

```
<area shape="rect" coords="310,20,375,33"
      href='test.php?dataSet=0&dataSetName=Web'>

<area shape="rect" coords="310,36,377,49"
      href='test.php?dataSet=1&dataSetName=Email'>

<area shape="rect" coords="310,52,377,65"
      href='test.php?dataSet=2&dataSetName=FTP'>
```

The image map consists of multiple <area> tags, one for each key in the legend box. The “href” attribute of the image map consists of a URL appended with parameters to indicate legend key the user

has clicked. The image map does not include the “<map>” or “</map>” tag. This is intentional so that you can add additional <area> tags to the image map.

The format of the appended URL parameters is determined using the optional queryFormat argument of the getHTMLImageMap method. If no queryFormat argument is specified, the default is:

```
dataSet={dataSet}&dataSetName={dataSetName}
```

The texts in curly brackets (i.e. {dataSet}, {dataSetName}) will be replaced by the actual values when generating the image map using “parameter substitution”. For example, {dataSetName} will be replaced by the label of the legend key clicked by the user.

The available parameters for the LegendBox object are described in the following table. You may use them in the url, queryFormat and extraAttr arguments (described below) by enclosing them in curly brackets.

Parameter	Description
dataSet	The entry number in the legend box. The first legend key is 0, while the n <sup>th</sup> legend key is (n-1).
dataSetName	The label of the legend key.

By using the queryFormat argument, you may remove unused URL parameters to reduce the image map size, or change the names of the URL parameters. For example, if you use “key={dataSetName}” as the queryString, only the label of the legend key will be included as URL parameters, and it will be called “key”.

In addition to customizing the URL parameters, ChartDirector supports additional HTML attributes in the <area> tags using the extraAttr argument of the getHTMLImageMap method. For example, you can add an “alt” HTML attribute to provide “tool tip” when the mouse moves over the bars. For example, the following extraAttr:

```
alt='Click me for details on {dataSetName}'
```

will add an “alt” attribute to the <area> tags in the image map, which will cause a “tool tip” to appear when the mouse moves over the legend keys.

Another common usage of the extraAttr argument is to add “onmouseover” and “onmouseout” HTML attributes to handle user interaction using Javascript on the browser.

#### Arguments

Argument	Default Value	Description
----------	---------------	-------------

url	(Mandatory)	The URL to be included as the “href” attribute of the image map. ChartDirector will append additional parameters to the URL to indicate which legend key the user has clicked.  If you do not use any URL, you may simply use an empty string.
queryFormat	""	A text string to specify the format of the parameters to be appended to the URL. If this argument is an empty string, a default queryFormat will be used. If you do not need any query parameter, use a space character as the queryFormat.
extraAttr	""	A text string to specify additional attributes to add to the <area> tag.

Return Value

Return a text string containing the image map generated.

## PieChart

The PieChart class, as its name implies, represents pie charts. It is a subclass of [BaseChart](#).

You can use the methods in this class to create a blank pie chart, add data to it, design its appearance and layout, and finally draws the pie chart.

Method	Description
<a href="#">PieChart Constructor</a>	Create a PieChart object.
<a href="#">setPieSize</a>	Set the position and size of the pie within the pie chart.
<a href="#">set3D</a>	Add 3D effects to the pie.
<a href="#">setStartAngle</a>	Set the angle of the first sector in the pie, and the direction (clockwise or anticlockwise) to layout subsequent sectors.
<a href="#">setLabelFormat</a>	Set the format of the all sector labels.
<a href="#">setLabelStyle</a>	Set the style used to draw all sector labels.
<a href="#">setLabelLayout</a>	Set the layout method and location of the sector labels.
<a href="#">setLineColor</a>	Set the sector edge color and join line color. The join line is the line that connects the sector labels to the sector perimeter.
<a href="#">setLabelPos</a>	Set the location of the sector labels, and specify whether join lines are used to connect the sector labels to the sector perimeter,
<a href="#">setData</a>	Set the data used to draw the pie chart.
<a href="#">sector</a>	Retrieve the <a href="#">Sector</a> object representing the specified sector in the pie chart.
<b>Methods inherited from BaseChart</b>	
<a href="#">setSize</a>	Set the size of the chart to the specified width and height in pixels.

<a href="#"><u>setBackground</u></a>	Set the background color, border color and 3D border effect of the chart.
<a href="#"><u>setBorder</u></a>	Set the border color of the chart.
<a href="#"><u>setWallpaper</u></a>	Specify an image as the background wallpaper of the chart.
<a href="#"><u>setBgImage</u></a>	Specify an image as the background image of the chart.
<a href="#"><u>setTransparentColor</u></a>	Set the transparent color for the image when writing the image to an image file.
<a href="#"><u>addTitle</u></a>	Add a title to the chart on the <a href="#"><u>TopCenter</u></a> position of the chart
<a href="#"><u>addTitle2</u></a>	Add a title to the chart at the top, bottom, left or right position of the chart.
<a href="#"><u>addLegend</u></a>	Add a legend box to the chart.
<a href="#"><u>getDrawArea</u></a>	Returns the <a href="#"><u>DrawArea</u></a> object that the chart is drawn with to allow drawing custom text, line or shapes.
<a href="#"><u>addDrawObj</u></a>	Add a custom-developed <a href="#"><u>DrawObj</u></a> to the chart.
<a href="#"><u>addText</u></a>	Add a text box to the chart.
<a href="#"><u>addLine</u></a>	Add a line to the chart.
<a href="#"><u>setColor</u></a>	Change the color of the specified position in the palette color table.
<a href="#"><u>setColors</u></a>	Change the colors of the color table starting from the first color.
<a href="#"><u>setColors2</u></a>	Change the colors of the color table starting with the specified position in the palette color table.
<a href="#"><u>getColor</u></a>	Get the color of the specified position in the palette color table.
<a href="#"><u>dashLineColor</u></a>	Create a dash line color that is suitable for drawing dash lines.
<a href="#"><u>patternColor</u></a>	Create a pattern color that can be used to fill areas with repeating patterns. The pattern is defined using an array of colors as the bitmap.
<a href="#"><u>patternColor2</u></a>	Create a pattern color that can be used to fill areas with repeating patterns. The pattern is defined by loading from an image file.
<a href="#"><u>gradientColor</u></a>	Create a gradient color that can be used to fill areas with color gradients. The gradient color is defined using a gradient line segment with a starting color and an ending color.
<a href="#"><u>gradientColor2</u></a>	Create a gradient color that can be used to fill areas with color gradients. The gradient color is defined using a gradient line segment with multiple color points.
<a href="#"><u>layout</u></a>	Perform auto-scaling of the axis and compute the position of the various objects of the chart, without actually drawing the chart. This allows additional custom text or shapes to be added to the chart based on the positions of other objects.
<a href="#"><u>makeChart</u></a>	Generate the chart and save it into a file.

<a href="#"><u>makeChart2</u></a>	Generate the chart in memory.
<a href="#"><u>makeChart3</u></a>	Generate the chart in internal format and return a <a href="#"><u>DrawArea</u></a> object to allow adding custom drawings on top of the chart.
<a href="#"><u>getHTMLImageMap</u></a>	Generate an HTML image map for the chart contents. This includes all the pie sectors and their labels.

## PieChart Constructor

PHP/Perl Usage

```
new PieChart(width, height [, bgColor [, edgeColor]])
```

Python Usage

```
PieChart(width, height [, bgColor [, edgeColor]])
```

C++ Prototype

```
static PieChart *create(int width, int height, int bgColor = BackgroundColor, int edgeColor = Transparent);
```

Description

Create a PieChart object.

Arguments

Argument	Default Value	Description
width	(Mandatory)	The width of the chart in pixels.
height	(Mandatory)	The height of the chart in pixels.
bgColor	<a href="#"><u>BackgroundColor</u></a>	The background color of the chart.
edgeColor	<a href="#"><u>Transparent</u></a>	The border color of the chart.

Return Value

Return the PieChart object created.

## setPieSize

PHP/Perl/Python Usage

```
setPieSize(x, y, r)
```

C++ Prototype

```
virtual void setPieSize(int x, int y, int r) = 0;
```

Description

Set the position and size of the pie within the pie chart.

#### Arguments

Argument	Default Value	Description
x	(Mandatory)	The x coordinate of the pie center.
y	(Mandatory)	The y coordinate of the pie center.
r	(Mandatory)	The radius of the pie.

Return Value

None

## set3D

PHP/Perl/Python Usage

```
set3D([depth [, angle [, shadowMode]])
```

C++ Prototype

```
virtual void set3D(int depth = -1, double angle = -1, bool shadowMode = false) = 0;
```

Description

Add 3D effects to the pie. By default, if this method is not called, a 2D pie will be drawn.

#### Arguments

Argument	Default Value	Description
depth	-1	The 3D depth of the pie in pixels. A value of -1 means the depth is automatically determined. (The current version uses the height of the chart divided by 20 as the depth.)
angle	-1	The view angle in degrees. Must be 0 – 90 for standard 3D mode, and 0 – 360 in shadow 3D mode. A value of -1 means the angle is automatically determined. (The current version uses 45 degrees if the depth is non-zero. If the depth is zero, it uses 0 degree.)
shadowMode	0	Flag to indicate whether the pie is in standard 3D mode or shadow 3D mode. A true (non-zero) value means shadow 3D mode, while a false value means standard 3D mode.

Return Value

None

## setStartAngle

PHP/Perl/Python Usage

```
setStartAngle(startAngle [, clockwise])
```

C++ Prototype

```
virtual void setStartAngle(double startAngle, bool clockWise = true) = 0;
```



#### Description

Set the angle of the first sector in the pie, and the direction (clockwise or anticlockwise) to layout subsequent sectors. By default, the `startAngle` is 0 degree (the 12 o'clock position), and subsequent sectors are drawn clockwise.

#### Arguments

Argument	Default Value	Description
<code>startAngle</code>	(Mandatory)	The angle to start drawing the first sector. The angle is measured starting from the 12 o'clock position in the clockwise direction. For example, 3 o'clock is 90 degrees, 6 o'clock is 180 degrees and 9 o'clock is 270 degrees.
<code>clockWise</code>	1	A flag to control the layout direction of the sectors. A true (non-zero) value indicates clockwise, while a false value indicates counter-clockwise.

#### Return Value

None

## setLabelFormat

PHP/Perl/Python Usage

`setLabelFormat(formatString)`

#### C++ Prototype

```
virtual void setLabelFormat(const char *formatString) = 0;
```

#### Description

Set the format of the all sector labels. If you just want to set the label format for one particular sector only, use the [setLabelFormat](#) method of the Sector object.

By default, if the circular label layout method is used, the sector label format is:

- “{label}\n{percent}%”

If the side label layout method is used, the sector label format is:

- “{label} ({percent}%)”

Please refer to the section on [Data and Text Formatting](#) for the meaning of the format string, and how to customize it.

#### Arguments

Argument	Default Value	Description
<code>formatString</code>	(Mandatory)	The format string. See above for description.

Return Value  
None

## setLabelStyle

PHP/Perl/Python Usage

setLabelStyle([font [, fontSize [, fontColor]])

C++ Prototype

```
virtual TextBox *setLabelStyle(const char *font, double fontSize = 8, int fontColor = TextColor) = 0;
```

Description

Set the style used to draw all sector labels. If you just want to set the style for one particular sector only, use the [setLabelStyle](#) method of the Sector object.

For details about how to specify font style, please refer to the section on [Font Specification](#).

Arguments

Argument	Default Value	Description
font	""	The font used to draw the sector labels. An empty string means using the default font (Arial).
fontSize	8	The font size used to draw the sector labels.
fontColor	<a href="#">TextColor</a>	The color used to draw the sector labels.

Return Value

A [TextBox](#) object that represents the Usage of the sector labels. You may use the methods of the TextBox object to fine tune the appearance of the sector labels.

## setLabelLayout

PHP/Perl/Python Usage

setLabelLayout (layoutMethod [, pos [, topBound [, bottomBound ]]])

C++ Prototype

```
virtual void setLabelLayout(int layoutMethod, int pos = -1, int topBound = -1, int bottomBound = -1) = 0;
```

Description

Set the layout method and location of the sector labels.

This method affects all sectors. If you just want to set the sector label layout method and location for one sector, use the [setLabelLayout](#) method of the Sector object.

ChartDirector supports two sector label layout methods – “circular layout” and “side layout”.

In the “circular layout” method, the sector labels are positioned around the perimeter of the pie.

In the “side layout” method, it can be imagined that there is an invisible rectangle containing the pie, and the sector labels are positioned on the left and right sides just outside the rectangle. Because the labels can be quite far away from the sectors, join lines are typically used to connect the labels to the sectors.

The “circular layout” method usually uses less space and is the default layout method. However, if the pie chart contains a lot of small sectors, the labels may overlap with each others. It is because there may be insufficient space on the pie perimeter to position the labels.

The “side layout” method has the advantages that it can avoid label overlapping unless under the most extreme cases (e.g. even the whole chart does not have enough space for all the labels). In the “side layout” method, labels will automatically shift up and down to avoid overlapping.

One common issue in pie charts is if they contain a lot of small sectors, and the data are sorted, the small sectors will be crowded together. Although the side layout method can avoid label overlapping by shifting the labels up and down, some labels may need to be shifted great distances.

In these cases, label layout can often be further improved (that is, distributed more evenly, and shifted less) by choosing an appropriate “start angle”.

The “start angle” is the angle of the first sector in the pie. Subsequent sectors are layout in clockwise direction by default. You can control the start angle and the layout direction by using the [setStartAngle](#) method.

Label layout can be improved if the small sectors near the horizontal axis. It is because the vertical distances between the sector labels are maximized for sectors near the horizontal axis. As a result, the extent of label overlapping is minimized.

If the data is in descending order (small sectors at the end), you may use a start angle of 135 degrees with clockwise sector layout. If the data is in ascending order (small sectors at the beginning), you may use a start angle of 45 degrees with clockwise sector layout. This will position the small sectors near the 90 degrees (horizontal) position.

#### Arguments

Argument	Default Value	Description
layoutMethod	(Mandatory)	Specify the layout method. Use the predefined constant “CircleLayout” for circular layout, and “SideLayout” for side layout.

pos	-1	<p>For circular layout, it is the distance between the sector perimeter and the sector label in number of pixels. If this parameter is negative (but not -1), that means the sector label will be drawn in the interior of the sector (that is, on the sector surface).</p> <p>For side layout, it is the distance between the pie and the left and right edges of the invisible containing rectangle.</p> <p>A value of -1 means the distance is automatically determined.</p>
topBound	-1	<p>This parameter is applicable only when the side layout method is used.</p> <p>In the side layout method, the sector labels can shift up or down to avoid overlapping. This parameter controls the top bound of the labels (the minimum y coordinate), thereby limiting the extent which the labels can be shifted up.</p> <p>This parameter is useful to avoid the labels from moving up too much and overlap with other chart objects.</p>
bottomBound	-1	<p>This parameter is applicable only when the side layout method is used.</p> <p>In the side layout method, the sector labels can shift up or down to avoid overlapping. This parameter controls the bottom bound of the labels (the maximum y coordinate), thereby limiting the extent which the labels can be shifted down.</p> <p>This parameter is useful to avoid the labels from moving down too much and overlap with other chart objects.</p>

Return Value

None

## setLineColor

PHP/Perl/Python Usage

setLineColor (edgeColor [, joinLineColor])

C++ Prototype

virtual void setLineColor(int edgeColor, int joinLineColor = -1) = 0;

Description

Set the sector edge color and join line color. The join line is the line that connects the sector labels to the sector perimeter.

This method affects all sectors. If you just want to set the sector label position or join line color for one sector, use the [setColor](#) method of the Sector object.

#### Arguments

Argument	Default Value	Description
edgeColor	(Mandatory)	The colors for the edges of the sectors. By default, the edge color is “SameAsMainColor”, which means the edge color is the same as the fill color of the sector. In this case, the sector will appear borderless.
joinLineColor	-1	The color of the line that join the sector perimeter with the sector label.  By default, if the circular label layout method is used, no join line will be used (color = Transparent). If the side label layout method is used, the join line will be the same color as the sector fill color (color = SameAsMainColor).

#### Return Value

None

## setLabelPos

PHP/Perl/Python Usage

setLabelPos(pos [, joinLineColor])

#### C++ Prototype

virtual void setLabelPos(int pos, int joinLineColor = Transparent) = 0;

#### Description

Set the location of the sector labels using circular layout, and specify whether join lines are used to connect the sector labels to the sector perimeter. This method affects all sectors. If you just want to set the sector label position or join line color for one sector, use the [setLabelPos](#) method of the [Sector](#) object.

#### Arguments

Argument	Default Value	Description
pos	(Mandatory)	The distance between the sector perimeter and the sector label in number of pixels. If this parameter is negative, that means the sector label will be drawn in the interior of the sector (that is, on the sector surface).
joinLineColor	Transparent	The color of the line that joins the sector perimeter with the sector label. The default value is Transparent, which means the line is not drawn. Note that join line does not apply if the sector label is inside the sector.

#### Return Value

None

## setData

PHP/Perl/Python Usage

setData(data [, labels])

C++ Prototype

virtual void setData([DoubleArray](#) data, [StringArray](#) labels = [StringArray\(\)](#)) = 0;

Description

Set the data used to draw the pie chart.

Arguments

Argument	Default Value	Description
data	(Mandatory)	A list of numbers representing the data points.
labels	empty array	A list of text strings that represents the labels of the sectors. An empty array means that the sectors will have no label. Otherwise, the length of labels should be the same as the length of data.

Return Value

None

## sector

PHP/Perl/Python Usage

sector(sectorNo)

C++ Prototype

virtual Sector \*sector(int sectorNo) = 0;

Description

Retrieve the [Sector](#) object representing a single sector in the pie chart. You may use the methods of this object to fine tune the sector appearance.

Arguments

Argument	Default Value	Description
sectorNo	(Mandatory)	The number of the sector that you want to retrieve. The sector number is the index of the data point in the <a href="#">setData</a> method when the sectors are created. Notes that the first sector is 0, the second sector is 1, and so on.

Return Value

The required [Sector](#) object.

# Sector

The Sector object represents a single sector in a pie chart. The Sector object is obtained by using the sector method of the PieChart object. You may use the Sector object to fine-tune the appearance of the sector.

Method	Description
<u>setExplode</u>	Explode the sector.
<u>setLabelFormat</u>	Set the format of the sector label.
<u>setLabelStyle</u>	Set the style used to draw the sector label.
<u>setLabelLayout</u>	Set the layout method and location of the sector labels.
<u>setLabelPos</u>	Set the location of the sector label, and specify whether a join line is used to connect the sector label to the sector perimeter.
<u>setColor</u>	Set the fill color, edge color and join line color of the sector.
<u>getImageCoor</u>	Get the image map coordinates of the sector as HTML image map attributes.
<u>getLabelCoor</u>	Get the image map coordinates of the sector label as HTML image map attributes.

## setExplode

PHP/Perl/Python Usage

setExplode([distance])

C++ Prototype

virtual void setExplode(int distance = -1) = 0;

Description

Explode the sector.

Arguments

Argument	Default Value	Description
distance	-1	The distance between the exploded sector and the center of the pie in number of pixels. A large value means that the exploded sector will be moved farther away from the pie. A value of -1 means the distance is automatically determined.

Return Value

None

## setLabelFormat

PHP/Perl/Python Usage  
setLabelFormat(formatString)

C++ Prototype  
virtual void setLabelFormat(const char \*formatString) = 0;

### Description

Set the format of the sector label. If you want to set the sector label format for all sectors, use the [setLabelFormat](#) method of the PieChart object.

### Arguments

Argument	Default Value	Description
formatString	(Mandatory)	The format string. See the <a href="#">setLabelFormat</a> method of the PieChart object for details.

### Return Value

None

## setLabelStyle

PHP/Perl/Python Usage  
setLabelStyle([font [, fontSize [, fontColor]]])

C++ Prototype  
virtual TextBox \*setLabelStyle(const char \*font, double fontSize = 8, int fontColor = TextColor) = 0;

### Description

Set the style used to draw the sector label. If you just want to set the style for all sector labels, use the [setLabelStyle](#) method of the PieChart object.

For details about how to specify font style, please refer to the section on [Font Specification](#).

### Arguments

Argument	Default Value	Description
font	""	The font used to draw the sector label. An empty string means using the default font (Arial).
fontSize	8	The font size used to draw the sector label.
fontColor	<a href="#">TextColor</a>	The color used to draw the sector label.

### Return Value

A [TextBox](#) object that represents the Usage of the sector label. You may use the methods of the TextBox object to fine tune the appearance of the sector label.



## setLabelLayout

PHP/Perl/Python Usage

setLabelLayout (layoutMethod [, pos ])

C++ Prototype

```
virtual void setLabelLayout(int layoutMethod, int pos = -1) = 0;
```

Description

Set the layout method and location of the sector labels. If you just want to set the sector label layout method and location for all sectors, use the [setLabelLayout](#) method of the PieChart object.

ChartDirector supports two sector label layout methods – “circular layout” and “side layout”. For detail descriptions of these label layout methods, please refer to the [setLabelLayout](#) method of the PieChart object.

Arguments

Argument	Default Value	Description
layoutMethod	(Mandatory)	Specify the layout method. Use the predefined constant “CircleLayout” for circular layout, and “SideLayout” for side layout.
pos	-1	For circular layout, it is the distance between the sector perimeter and the sector label in number of pixels. If this parameter is negative (but not -1), that means the sector label will be drawn in the interior of the sector (that is, on the sector surface).  For side layout, it is the distance between the pie and the left and right edges of the invisible containing rectangle.  A value of -1 means the distance is automatically determined.

Return Value

None

## setLabelPos

PHP/Perl/Python Usage

setLabelPos(pos [, joinLineColor])

C++ Prototype

```
virtual void setLabelPos(int pos, int joinLineColor = Transparent) = 0;
```

Description

Set the location of the sector label using circular layout, and specify whether a join line is used to connect the sector label to the sector perimeter. If you want to set the sector label position or join line color for all sectors, use the [setLabelPos](#) method of the PieChart object.

#### Arguments

Argument	Default Value	Description
pos	(Mandatory)	The distance between the sector perimeter and the sector label in number of pixels. If this parameter is negative, that means the sector label will be drawn in the interior of the sector (that is, on the sector surface).
joinLineColor	Transparent	The color of the line that joins the sector perimeter with the sector label. The default value is Transparent, which means the line is not drawn. Note that join line does not apply if the sector label is inside the sector.

#### Return Value

None

## setColor

PHP/Perl/Python Usage

setColor(color [, edgeColor [, joinLineColor ]])

C++ Prototype

virtual void setColor(int color, int edgeColor = -1, int joinLineColor = -1) = 0;

#### Description

Set the fill color, edge color and join line color of the sector.

#### Arguments

Argument	Default Value	Description
color	(Mandatory)	The color of the sector.
edgeColor	-1	The edge color of the sector. The default value of -1 means the edge color will be unchanged. If you have never set the edge color before, the default edge color is <a href="#">LineColor</a> .
joinLineColor	-1	The color of the line that join the sector perimeter with the sector label.  By default, if the circular label layout method is used, no join line will be used (that is, color = Transparent). If the side label layout method is used, the join line will be the same color as the sector fill color (that is, color = SameAsMainColor).

#### Return Value

None

## getImageCoor

PHP/Perl/Python Usage

getImageCoor()

C++ Prototype

```
virtual const char *getImageCoor() = 0;
```

Description

Get the image map coordinates of the sector as HTML image map attributes in the following format:

```
shape="poly" cords="<x1>,<y1>,<x2>,<y2> ..."
```

where (x1, y1), (x2, y2) ... are vertices of a polygon that approximate the sector. The format is specially designed so that it can easily be included into HTML image maps.

If you want to get the complete image map for all the sectors and sector labels of a pie chart, use the [getHTMLImageMap](#) method of the PieChart object instead.

This method should be called only after you have completely drawn the chart using the [makeChart](#), [makeChart2](#) or [makeChart3](#) method. It is because ChartDirector needs to actually draw the chart to compute the coordinates of the sector.

Arguments

None

Return Value

Return a text string representing the coordinates of the sector in HTML image map attribute format.

## getLabelCoor

PHP/Perl/Python Usage

getLabelCoor()

C++ Prototype

```
virtual const char *getLabelCoor() = 0;
```

Description

Get the image map coordinates of the sector label as HTML image map attributes in the following format:

```
shape="rect" cords="<x1>,<y1>,<x2>,<y2>"
```

where (x1, y1) and (x2, y2) are opposite corners of the box that enclosed the sector label. The format is specially designed so that it can easily be included into HTML image maps.

If you want to get the complete image map for all the sectors and sector labels of a pie chart, use the [getHTMLImageMap](#) method of the PieChart object instead.

This method should be called only after you have completely drawn the chart using the [makeChart](#), [makeChart2](#) or [makeChart3](#) method. It is because ChartDirector needs to actually draw the chart to compute the coordinates of the sector labels.

Arguments

None

Return Value

Return a text string representing the coordinates of the sector label in HTML image map attribute format.

## XYChart

The XYChart class represents an XYChart object. In ChartDirector, all chart types that have an X and Y axis are implemented as layers contained in an XYChart object. The currently supported layers include bar chart layer, area chart layer, line chart layer and high-low-open-close chart layer. You may include multiple layers in an XYChart to create “combo” charts.

The XYChart class is a subclass of [BaseChart](#).

Method	Description
<a href="#">XYChart Constructor</a>	Create an XYChart object.
<a href="#">xAxis</a>	Retrieve the XAxis object representing primary x-axis object of the XYChart.
<a href="#">xAxis2</a>	Retrieve the XAxis object representing secondary x-axis object of the XYChart.
<a href="#">yAxis</a>	Retrieve the YAxis object representing primary y-axis object of the XYChart.
<a href="#">yAxis2</a>	Retrieve the YAxis object representing secondary y-axis object of the XYChart.
<a href="#">syncYAxis</a>	Specify that the secondary y-axis be derived from the primary y-axis through a formula $y2 = y1 * \text{slope} + \text{intercept}$ .
<a href="#">setYAxisOnRight</a>	Specify whether the position of the primary y-axis is on the right or on the left of the chart. (The secondary y-axis will be on the opposite position).
<a href="#">swapXY</a>	Swap the position of the x-axis and y-axis, that is, the x-axis will be vertical and the y-axis will be horizontal.
<a href="#">setPlotArea</a>	Sets the position, size, background colors, edge color and grid colors of the plot area.
<a href="#">addBarLayer</a>	Add a bar chart layer to the XYChart, and specify the data set to use for drawing the bars.
<a href="#">addBarLayer2</a>	Add an empty bar chart layer to the XYChart.

<a href="#"><u>addBarLayer3</u></a>	Add a multi-color bar chart layer to the XYChart, and specify the data set to use for drawing the bars.
<a href="#"><u>addLineLayer</u></a>	Add a line chart layer to the XYChart, and specify the data set to use for drawing the line.
<a href="#"><u>addLineLayer2</u></a>	Add an empty line chart layer to the XYChart.
<a href="#"><u>addScatterLayer</u></a>	Add a scatter chart layer to the XYChart.
<a href="#"><u>addTrendLayer</u></a>	Add a trend chart layer to the XYChart, and specify the data set to use for drawing the trend line.
<a href="#"><u>addTrendLayer2</u></a>	Add a trend chart layer to the XYChart, and specify the x and y values for drawing the trend line.
<a href="#"><u>addAreaLayer</u></a>	Add an area chart layer to the XYChart, and specify the data set to use for drawing the area.
<a href="#"><u>addAreaLayer2</u></a>	Add an empty area chart layer to the XYChart.
<a href="#"><u>addHLOCLayer</u></a>	Add a high-low-open-close (HLOC) chart layer to the XYChart, and specify the data sets to use for drawing the layer.
<a href="#"><u>addHLOCLayer2</u></a>	Add an empty high-low-open-close (HLOC) chart layer to the XYChart.
<a href="#"><u>addCandleStickLayer</u></a>	Add a candlestick chart layer to the XYChart, and specify the data sets to use for drawing the layer.
<b>Methods inherited from BaseChart</b>	
<a href="#"><u>setSize</u></a>	Set the size of the chart to the specified width and height in pixels.
<a href="#"><u>setBackground</u></a>	Set the background color, border color and 3D border effect of the chart.
<a href="#"><u>setBorder</u></a>	Set the border color of the chart.
<a href="#"><u>setWallpaper</u></a>	Specify an image as the background wallpaper of the chart.
<a href="#"><u>setBgImage</u></a>	Specify an image as the background image of the chart.
<a href="#"><u>addTitle</u></a>	Add a title to the chart on the TopCenter position of the chart
<a href="#"><u>addTitle2</u></a>	Add a title to the chart at the top, bottom, left or right position of the chart.
<a href="#"><u>addLegend</u></a>	Add a legend box to the chart.
<a href="#"><u>getDrawArea</u></a>	Return the <a href="#"><u>DrawArea</u></a> object that the chart is drawn with to allow drawing custom text, line or shapes.
<a href="#"><u>addDrawObj</u></a>	Add a custom-developed <a href="#"><u>DrawObj</u></a> to the chart.
<a href="#"><u>addText</u></a>	Add a text box to the chart.
<a href="#"><u>addLine</u></a>	Add a line to the chart.
<a href="#"><u>setColor</u></a>	Change the color of the specified position in the palette color table.
<a href="#"><u>setColors</u></a>	Change the colors of the color table starting from the first color.

<a href="#"><u>setColors2</u></a>	Change the colors of the color table starting with the specified position in the palette color table.
<a href="#"><u>getColor</u></a>	Get the color of the specified position in the palette color table.
<a href="#"><u>dashLineColor</u></a>	Create a dash line color that is suitable for drawing dash lines.
<a href="#"><u>patternColor</u></a>	Create a pattern color that can be used to fill areas with repeating patterns. The pattern is defined using an array of colors as the bitmap.
<a href="#"><u>patternColor2</u></a>	Create a pattern color that can be used to fill areas with repeating patterns. The pattern is defined by loading from an image file.
<a href="#"><u>gradientColor</u></a>	Create a gradient color that can be used to fill areas with color gradients. The gradient color is defined using a gradient line segment with a starting color and an ending color.
<a href="#"><u>gradientColor2</u></a>	Create a gradient color that can be used to fill areas with color gradients. The gradient color is defined using a gradient line segment with multiple color points.
<a href="#"><u>layout</u></a>	Perform auto-scaling of the axis and compute the position of the various objects of the chart, without actually drawing the chart. This allows additional custom text or shapes to be added to the chart based on the positions of other objects.
<a href="#"><u>makeChart</u></a>	Generate the chart and save it into a file.
<a href="#"><u>makeChart2</u></a>	Generate the chart in memory.
<a href="#"><u>makeChart3</u></a>	Generate the chart in internal format and return a <a href="#"><u>DrawArea</u></a> object to allow you to add custom drawings on top of the chart.
<a href="#"><u>getHTMLImageMap</u></a>	Generate an HTML image map for the chart contents. It includes the image maps of all the chart layers (bar layers, line layers, area layers and HLOC layers).

## XYChart Constructor

PHP/Perl Usage

```
new XYChart(width, height [, bgColor [, edgeColor]])
```

Python Usage

```
XYChart(width, height [, bgColor [, edgeColor]])
```

C++ Prototype

```
static XYChart *create(int width, int height, int bgColor = BackgroundColor, int edgeColor = Transparent);
```

Description

Create an XYChart object.

#### Arguments

Argument	Default Value	Description
width	(Mandatory)	The width of the chart in pixels.
height	(Mandatory)	The height of the chart in pixels.
bgColor	<u>BackgroundColor</u>	The background color of the chart. The default value is the BackgroundColor entry of the color palette.
edgeColor	Transparent	The edge color of the chart. The default value is Transparent, which means the chart will have no edge.

#### Return Value

Returns the XYChart object created.

## xAxis

#### PHP Usage

xAxis

Note: xAxis is implemented as a property of the XYChart in PHP. It is not a method and therefore does not need to have the “()”.

#### Perl/Python Usage

xAxis()

#### C++ Prototype

```
virtual XAxis *xAxis() = 0;
```

#### Description

Retrieve the XAxis object representing primary x-axis object of the XYChart. The primary x-axis is the x-axis on the bottom of the chart, while the secondary x-axis is the x-axis on the top of the chart.

#### Arguments

None

#### Return Value

Returns an XAxis object representing the primary x-axis of the XYChart.

## xAxis2

#### PHP Usage

xAxis2

Note: xAxis2 is implemented as a property of the XYChart in PHP. It is not a method and therefore does not need to have the “()”.

#### Perl/Python Usage

xAxis2()

C++ Prototype

```
virtual XAxis *xAxis2() = 0;
```

Description

Retrieve the [XAxis](#) object representing secondary x-axis object of the XYChart. The primary x-axis is the x-axis on the bottom of the chart, while the secondary x-axis is the x-axis on the top of the chart.

Arguments

None

Return Value

Returns an [XAxis](#) object representing the secondary x-axis of the XYChart.

## yAxis

PHP Usage

yAxis

Note: yAxis is implemented as a property of the XYChart in PHP. It is not a method and therefore does not need to have the "()".

Perl/Python Usage

yAxis()

C++ Prototype

```
virtual YAxis *yAxis() = 0;
```

Description

Retrieve the [YAxis](#) object representing primary y-axis object of the XYChart. By default, the primary y-axis is the y-axis on the left side of the chart, while the secondary y-axis is the y-axis on the right side of the chart. You may interchange the positions of the two y-axes using the setYAxisOnRight method.

Arguments

None

Return Value

Returns a [YAxis](#) object representing the primary y-axis of the XYChart.

## yAxis2

PHP Usage

yAxis2

Note: yAxis2 is implemented as a property of the XYChart in PHP. It is not a method and therefore does not need to have the "()".

Perl/Python Usage

yAxis2()



C++ Prototype

```
virtual YAxis *yAxis2() = 0;
```

Description

Retrieve the [YAxis](#) object representing secondary y-axis object of the XYChart. By default, the primary y-axis is the y-axis on the left side of the chart, while the secondary y-axis is the y-axis on the right side of the chart. You may interchange the positions of the two y-axes using the [setYAxisOnRight](#) method.

Arguments

None

Return Value

Returns a [YAxis](#) object representing the secondary y-axis of the XYChart.

## syncYAxis

PHP/Perl/Python Usage

```
syncYAxis([slope [, intercept]])
```

C++ Prototype

```
virtual void syncYAxis(double slope = 1, double intercept = 0) = 0;
```

Description

Specify that the secondary y-axis be derived from the primary y-axis through a formula:

$$y2 = y1 * \text{slope} + \text{intercept}$$

This method is usually used if the two y-axes represent the same measurement but are using different units. For example, if the primary y-axis represents temperature in Celsius, and the secondary y-axis represents temperature in Fahrenheit, then:

$$y2 = y1 * 1.8 + 32$$

In this case, you can sync up the axes by using:

```
syncYAxis(1.8, 32);
```

If you call the `syncYAxis` without any parameters, the default is to use 1 as the slope and 0 as the intercept. This essentially means the two y-axes are of the same scale.

Arguments

Argument	Default Value	Description
slope	1	The slope parameter of the formula linking the secondary y-axis to the primary y-axis.
intercept	0	The intercept parameter of the formula linking the secondary y-axis to the primary y-axis.

Return Value  
None.

## setYAxisOnRight

PHP/Perl/Python Usage  
setYAxisOnRight([ b ])

C++ Prototype  
virtual void setYAxisOnRight(bool b = true) = 0;

Description  
Specify the position of the primary y-axis and the secondary y-axis.

By default, the primary y-axis is the y-axis on the left side of the chart, while the secondary y-axis is the y-axis on the right side of the chart. If the setYAxisOnRight method is called passing “true” as the argument, the position of the y-axes will be interchanged, that is, the primary y-axis will be on the right, while the secondary y-axis will be on the left. Calling setYAxisOnRight with a “false” argument will set the primary y-axis on the left and secondary y-axis on the right.

### Arguments

Argument	Default Value	Description
b	1	If this argument is “true” (non-zero), the primary y-axis will be on the right, and the secondary y-axis will be on the left. If this argument is “false”, the primary y-axis will be on the left, and the secondary y-axis will be on the right.

Return Value  
None.

## swapXY

PHP/Perl/Python Usage  
swapXY([b])

C++ Prototype  
virtual void swapXY(bool b = true) = 0;

Description  
Swap the position of the x-axis and y-axis, that is, the x-axis will be vertical and the y-axis will be horizontal.

In the current version of ChartDirector, only bar chart layers support swapping of x and y axes. After swapping x and y axes, the bars will be horizontal instead of vertical.

For other chart layer types, nothing will be drawn if the axes are swapped.

#### Arguments

Argument	Default Value	Description
b	1	If this argument is “true” (non-zero), the x-axis and y-axis will be swapped, that is, the x-axis will be the vertical axis and the y-axis will be horizontal. If this argument is “false” (zero), the x-axis and y-axis will return to normal.

#### Return Value

None

## setPlotArea

PHP/Perl/Python Usage

setPlotArea(x, y, width, height [, bgColor [, altBgColor [, edgeColor [, hGridColor [, vGridColor]]]])

#### C++ Prototype

```
virtual PlotArea *setPlotArea(int x, int y, int width, int height, int bgColor = Transparent, int altBgColor = -1, int edgeColor = LineColor, int hGridColor = 0xc0c0c0, int vGridColor = Transparent) = 0;
```

#### Description

Sets the position, size, background colors, edge color and grid colors of the plot area.

#### Arguments

Argument	Default Value	Description
x	(Mandatory)	The x coordinate of the top left corner of the plot area.
y	(Mandatory)	The y coordinate of the top left corner of the plot area.
width	(Mandatory)	The width of the plot area in pixels.
height	(Mandatory)	The height of the plot area in pixels.
bgColor	Transparent	The background color of the plot area.
altBgColor	-1	The second background color of the plot area. The default value (-1) means there is no second background color. If there is a second background color, the two background colors will use alternatively as horizontal bands on the background grid.
edgeColor	<u>LineColor</u>	The color used to draw the border of the plot area. The default value is the LineColor entry of the color palette.
hGridColor	0xc0c0c0	The horizontal grid color. The default value is light gray (0xc0c0c0).
vGridColor	Transparent	The vertical grid color. By default, it is Transparent, meaning that the vertical grid is invisible.

Return Value

A [PlotArea](#) object representing the plot area.

## addBarLayer

PHP/Perl/Python Usage

`addBarLayer(data [, color [, name [, depth]]])`

C++ Prototype

```
virtual BarLayer *addBarLayer(DoubleArray data, int color = -1, const char *name = 0, int depth = 0) = 0;
```

Description

Add a bar chart layer to the [XYChart](#), and specify the data set to use for drawing the bars. This method is typically used if you have only one data set in the bar layer. If you have multiple data sets (e.g. in a stacked bar chart), use the [addBarLayer2](#) method instead.

All bars in this bar chart layer have the same color. If you want each bar to have a different color, use the [addBarLayer3](#) method instead.

Arguments

Argument	Default Value	Description
data	(Mandatory)	A list of numbers representing the data set. Note that <a href="#">ChartDirector</a> supports the special constant <a href="#">NoValue</a> as values in the array to specify that certain data points have no value.
color	-1	The color to draw the bars. The default value of -1 means that the color is automatically selected from the color palette.
name	""	The name of the data set. The name will be used in the legend box, if one is available. The default value "" (empty string) means that there is no name.
depth	0	The depth of the bar layer. The default of zero means that the bar layer is flat. A non-zero depth means that the bar layer is in 3D.

Return Value

A [BarLayer](#) object representing the bar layer created.

## addBarLayer2

PHP/Perl/Python Usage

`addBarLayer2([dataCombineMethod [, depth]])`

C++ Prototype

```
virtual BarLayer *addBarLayer(int dataCombineMethod = Side, int depth = 0) = 0;
```

## Description

Add a bar chart layer to the XYChart. This method returns a [BarLayer](#) object representing the bar layer. You may use then add one or more data sets to the bar layer using the methods of the BarLayer object.

If you only have one data set in the bar layer, you may also use the [addBarLayer](#) method to add a bar layer to the chart.

## Arguments

Argument	Default Value	Description
dataCombineMethod	Side	<p>The method to combine the data sets together in a single bar layer. The followings are the supported methods:</p> <ul style="list-style-type: none"><li>▪ Side: The data sets are combined by plotting the bars side by side.</li><li>▪ Stack: The data sets are combined by stacking up the bar.</li><li>▪ Overlay: The data sets are represented by stacked bars similar to Stack. However, in Overlay, one data set is assumed to include the other data set.</li></ul> <p>For example, if the data sets are “average loading” and “peak loading”, we cannot simply stack the peak loading on top of the average loading, because the peaking loading already contains the average loading. Instead, the Overlay style should be used.</p> <p>In general, if there are multiple data sets, the Overlay will sort the data sets by their values, and assume the larger data values include the smaller data values.</p>
depth	0	<p>The depth of the bar layer. The default of zero means that the bar layer is flat. A non-zero depth means that the bar layer is in 3D.</p>

## Return Value

A [BarLayer](#) object representing the bar layer created.

## addBarLayer3

PHP/Perl/Python Usage

addBarLayer3(data [, colors [, names [, depth]]])

C++ Prototype

```
virtual BarLayer *addBarLayer(DoubleArray data, IntArray colors, StringArray names = StringArray(),  
int depth = 0) = 0;
```

Description

Add a multi-color bar chart layer to the XYChart, and specify the data set to use for drawing the bars.

A multi-color bar chart is a bar chart in which each bar has a different color. In a normal bar chart, each data set has a different color, but the bars in the same data set have the same color.

If you have multiple data sets (e.g. in a stacked bar chart), use the [addBarLayer2](#) method instead. If you only have one data set, but you want all the bars to have the same color, use the [addBarLayer](#) method instead.

Arguments

Argument	Default Value	Description
data	(Mandatory)	An array of numbers representing the data set.
colors	empty array	An array of colors to draw the bars. The size of the array should be the same the number of data points. An empty array means the colors are automatically selected from the color palette.
names	empty array	An array of text strings as the names of the bars. The names will be used in the legend box, if one is available. The size of the array should be the same the number of data points. An empty array means that bars have no name.
depth	0	The depth of the bar layer. The default of zero means that the bar layer is flat. A non-zero depth means that the bar layer is in 3D.

Return Value

A [BarLayer](#) object representing the bar layer created.

## addLineLayer

PHP/Perl/Python Usage

```
addLineLayer(data [, color [, name [, depth]]])
```

C++ Prototype

```
virtual LineLayer *addLineLayer(DoubleArray data, int color = -1, const char *name = 0, int depth = 0)  
= 0;
```

#### Description

Add a line chart layer to the XYChart, and specify the data set to use for drawing the line. This method is typically used if you have only one data set in the line layer. If you have multiple data sets, use the [addLineLayer2](#) method instead.

#### Arguments

Argument	Default Value	Description
data	(Mandatory)	A list of numbers representing the data set.
color	-1	The color to draw the line. The default value of -1 means that the color is automatically selected from the color palette.
name	""	The name of the data set. The name will be used in the legend box, if one is available. The default value of "" (empty string) means that there is no name.
depth	0	The depth of the line layer. The default of zero means that the line layer is flat. A non-zero depth means that the line layer is in 3D.

#### Return Value

A [LineLayer](#) object representing the line layer created.

## addLineLayer2

PHP/Perl/Python Usage

```
addLineLayer2([dataCombineMethod [, depth]])
```

#### C++ Prototype

```
virtual LineLayer *addLineLayer(int dataCombineMethod = Overlay, int depth = 0) = 0;
```

#### Description

Add a line chart layer to the XYChart. This method returns a [LineLayer](#) object representing the line layer. You may use then add one or more data sets to the line layer using the methods of the LineLayer object.

If you only have one data set in the line layer, you may also use the [addLineLayer](#) method to add a line layer to the chart.

#### Arguments

Argument	Default Value	Description
dataCombineMethod	Overlay	In this version of the ChartDirector, the only supported method is Overlay. That means the lines are plot on the chart without any further manipulations.

depth	0	The depth of the line layer. The default of zero means that the line layer is flat. A non-zero depth means that the line layer is in 3D.
-------	---	--

Return Value

A [LineLayer](#) object representing the line layer created.

## addScatterLayer

PHP/Perl/Python Usage

addScatterLayer(xData, yData [, name [, symbol [, symbolSize [, fillColor [, edgeColor ]]]]])

C++ Prototype

virtual ScatterLayer \*addScatterLayer([DoubleArray](#) xData, [DoubleArray](#) yData, const char \*name = 0, SymbolType symbol = SquareSymbol, int symbolSize = 5, int fillColor = -1, int edgeColor = -1) = 0;

Description

Add a scatter chart layer to the XYChart.

A scatter chart can be considered as a special kind of line chart, in which both x and y values are used to plot the data points, and in which the data symbols are enabled and the line width is set to zero. Therefore only the data symbols are visible.

ChartDirector supports setting the line width to non-zero using the [setLineWidth](#) method of the Layer object. In this case, there will be lines joining the data points.

Arguments

Argument	Default Value	Description
xData	(Mandatory)	An array of numbers representing the x values of the data points.
yData	(Mandatory)	An array of numbers representing the y values of the data points.
name	""	The name of the data set. The name will be used in the legend box, if one is available. The default value of "" (empty string) means that there is no name.
symbol	SquareSymbol	One of the pre-defined symbol constants (see the <a href="#">setDataSymbol</a> method of the DataSet object for the symbol table) to specify the symbol to use.
symbolSize	5	The width and height of the symbol in pixels. The default symbol size is 5 pixels. It is recommended to use an odd number for the symbol size, as many symbols will draw better if the center point has integer coordinates.



fillColor	-1	The color used to fill the symbol. The default value of -1 means that the color is automatically selected from the color palette.
edgeColor	-1	The edge color used to draw the edge of the symbol. The default value of -1 means using the LineColor entry of the color palette.

Return Value

A [ScatterLayer](#) object representing the scatter layer created.

## addTrendLayer

PHP/Perl/Python Usage

addTrendLayer(data [, color [, name [, depth ]]])

C++ Prototype

virtual TrendLayer \*addTrendLayer([DoubleArray](#) data, int color = -1, const char \*name = 0, int depth = 0) = 0;

Description

Add a trend chart layer to the XYChart, and specify the data set to use for drawing the trend line. This method is typically used if you only need to use the y values to draw the chart. (The data points assumed to be equally spaced on the x-axis.) If your data points are not equally spaced on the x-axis and you need to specify both the x and y values, use the [addTrendLayer2](#) method instead.

Arguments

Argument	Default Value	Description
data	(Mandatory)	An array of numbers representing the data set.
color	-1	The color to draw the line. The default value of -1 means that the color is automatically selected from the color palette.
name	""	The name of the data set. The name will be used in the legend box, if one is available. The default value of "" (empty string) means that there is no name.
depth	0	The depth of the line layer. The default of zero means that the line layer is flat. A non-zero depth means that the trend line layer is in 3D.

Return Value

A [TrendLayer](#) object representing the trend layer created.

## addTrendLayer2

PHP/Perl/Python Usage

addTrendLayer2(xData, yData [, color [, name [, depth ]]])

C++ Prototype

```
virtual TrendLayer *addTrendLayer(DoubleArray xData, DoubleArray yData, int color = -1, const char *name = 0, int depth = 0) = 0;
```

Description

Add a trend chart layer to the XYChart, and specify the x and y values for drawing the trend line. This method is typically used if your data points are not equally spaced on the x-axis and you need to specify both the x and y values to draw the charts. If your data points are equally spaced on the x-axis, usually you just need to specify the y values in drawing the chart. In this case, you can use the [addTrendLayer](#) method instead.

Arguments

Argument	Default Value	Description
xData	(Mandatory)	An array of numbers representing the x values of the data points.
yData	(Mandatory)	An array of numbers representing the y values of the data points.
color	-1	The color to draw the line. The default value of -1 means that the color is automatically selected from the color palette.
name	""	The name of the data set. The name will be used in the legend box, if one is available. The default value of "" (empty string) means that there is no name.
depth	0	The depth of the line layer. The default of zero means that the line layer is flat. A non-zero depth means that the trend line layer is in 3D.

Return Value

A [TrendLayer](#) object representing the trend layer created.

## addAreaLayer

PHP/Perl/Python Usage

```
addAreaLayer(data [, color [, name [, depth]]])
```

C++ Prototype

```
virtual AreaLayer *addAreaLayer(DoubleArray data, int color = -1, const char *name = 0, int depth = 0) = 0;
```

Description

Add an area chart layer to the XYChart, and specify the data set to use for drawing the area. This method is typically used if you have only one data set in the area layer. If you have multiple data sets (e.g. in a stacked area chart), use the [addAreaLayer2](#) method instead.

#### Arguments

Argument	Default Value	Description
data	(Mandatory)	A list of numbers representing the data set.
color	-1	The color to draw the area. The default value of -1 means that the color is automatically selected from the color palette.
name	""	The name of the data set. The name will be used in the legend box, if one is available. The default value "" (empty string) means that there is no name.
depth	0	The depth of the area layer. The default of zero means that the area layer is flat. A non-zero depth means that the area layer is in 3D.

#### Return Value

An [AreaLayer](#) object representing the area layer created.

## addAreaLayer2

PHP/Perl/Python Usage

`addAreaLayer2([dataCombineMethod [, depth]])`

#### C++ Prototype

`virtual AreaLayer *addAreaLayer(int dataCombineMethod = Stack, int depth = 0) = 0;`

#### Description

Add an area chart layer to the XYChart. This method returns an [AreaLayer](#) object representing the area layer. You may use then add one or more data sets to the area layer using the methods of the AreaLayer object.

If you only have one data set in the area layer, you may also use the [addAreaLayer](#) method to add an area layer to the chart.

#### Arguments

Argument	Default Value	Description
dataCombineMethod	Stack	In this version of the ChartDirector, the only supported method is Stack. That means the areas are plot on the chart by stacking on top of one another.
depth	0	The depth of the area layer. The default of zero means that the area layer is flat. A non-zero depth means that the area layer is in 3D.

#### Return Value

An [AreaLayer](#) object representing the area layer created.

## addHLOCLayer

PHP/Perl/Python Usage

```
addHLOCLayer(highData, lowData [, openData [, closeData [, color]]])
```

C++ Prototype

```
virtual HLOCLayer *addHLOCLayer(DoubleArray highData, DoubleArray lowData, DoubleArray  
openData, DoubleArray closeData, int color = -1) = 0;;
```

Description

Add a high-low-open-close (HLOC) chart layer to the XYChart, and specify the data sets to use for drawing the layer.

HLOC charts are commonly used in stock price charts to representing the highest price, lowest price, opening price and the closing price. This chart, of course, can be used for many other purposes as well.

In the ChartDirector HLOC chart, the high and low data sets are mandatory, while the open and close data sets are optional.

Arguments

Argument	Default Value	Description
highData	(Mandatory)	A list of numbers representing the highest value data set.
lowData	(Mandatory)	A list of numbers representing the lowest value data set.
openData	empty array	A list of numbers representing the opening value data set. An empty array means there is no opening value data set available.
closeData	empty array	A list of numbers representing the closing value data set. An empty array means there is no closing value data set available.
color	-1	The color to draw the line. The default value of -1 means that the color is automatically selected from the color palette.

Return Value

A [HLOCLayer](#) object representing the HLOC layer created.

## addHLOCLayer2

PHP/Perl/Python Usage

```
addHLOCLayer2()
```

C++ Prototype

```
virtual HLOCLayer *addHLOCLayer() = 0;
```

#### Description

Add a high-low-open-close (HLOC) chart layer to the XYChart. This method returns a [HLOCLayer](#) object representing the HLOC layer. You may use then add data sets to the HLOC using the methods of the HLOCLayer object.

HLOC charts are commonly used in stock price charts to representing the highest price, lowest price, opening price and the closing price. This chart, of course, can be used for many other purposes as well.

In the ChartDirector HLOC chart, the high and low data sets are mandatory, while the open and close data sets are optional. Therefore you need to add at least two data sets to the HLOC layer for the high and low data sets. The third data set, if added, will be the open data set. The fourth data set, if added, will be the close data set. If you just have high, low and close data sets but do not have the open data set, you must still add an empty open data set (that is, a data set with no data points) to the HLOC layer.

#### Arguments

None.

#### Return Value

A [HLOCLayer](#) object representing the HLOC layer created.

## addCandleStickLayer

PHP/Perl/Python Usage

addCandleStickLayer(highData, lowData, openData, closeData [, riseColor [, fallColor [, edgeColor ]]])

#### C++ Prototype

```
virtual CandleStickLayer *addCandleStickLayer(DoubleArray highData, DoubleArray lowData, DoubleArray openData, DoubleArray closeData, int riseColor = 0xffffffff, int fallColor = 0x0, int edgeColor = LineColor) = 0;
```

#### Description

Add a candlestick chart layer to the XYChart, and specify the data sets to use for drawing the layer.

Candlestick charts are commonly used in stock price charts to representing the highest price, lowest price, opening price and the closing price. This chart, of course, can be used for many other purposes as well.

#### Arguments

Argument	Default Value	Description
highData	(Mandatory)	An array of numbers representing the highest value data set.
lowData	(Mandatory)	An array of numbers representing the lowest value data set.
openData	(Mandatory)	An array of numbers representing the opening value data set.
closeData	(Mandatory)	An array of numbers representing the closing value data set.
riseColor	0xffffffff	The color used to fill the candle if the closing value is higher than the opening value.

fallColor	0x0	The color used to fill the candle if the opening value is higher than the closing value.
edgeColor	LineColor	The color used to draw the vertical line and the border of the candle in a candlestick chart. The default value is the LineColor entry of the color palette.

Return Value

A [CandleStickLayer](#) object representing the CandleStick layer created.

## PlotArea

The PlotArea object represents the plot area within an [XYChart](#). The PlotArea object is obtained by using the [setPlotArea](#) method of the XYChart.

Method	Description
<a href="#">setBackground</a>	Set the background colors and the border color of the plot area.
<a href="#">setBackground2</a>	Specify an image as the background image of the plot area.
<a href="#">setGridColor</a>	Set the horizontal and vertical grid colors of the plot area.
<a href="#">setGridWidth</a>	Set the horizontal and vertical grid line width.

## setBackground

PHP/Perl/Python Usage

setBackground(color [, altBgColor [, edgeColor]])

C++ Prototype

virtual void setBackground(int color, int altBgColor = -1, int edgeColor = LineColor) = 0;

Description

Set the background colors and the border color of the plot area. The plot area can have one or two background colors. If it has two background colors, they are drawn alternatively as horizontal bands on the background grid.

Note that you may also specify the background and edge colors when you define the plot area using the [setPlotArea](#) method of the XYChart object.

Arguments

Argument	Default Value	Description
color	(Mandatory)	The background color.

altBgColor	-1	The second background color. The default value (-1) means there is no second background color. If there is a second background color, the two background colors will use alternatively as horizontal bands on the background grid.
edgeColor	LineColor	The color used to draw the border of the plot area. The default value is the LineColor entry of the color palette.

Return Value

None

## setBackground2

PHP/Perl/Python Usage

setBackground2(img [, align])

C++ Prototype

virtual void setBackground(const char \*img, Alignment align = Center) = 0;

Description

Use the image loaded the specified file as the background image of the plot area. The method will auto-detect the image file format using the file name extension, which must either be png, jpg, jpeg, gif, wbmp or wmp (case insensitive). The alignment of the image is controlled by the optional “align” argument. The default value of the “align” argument is Center. All [alignment](#) values are supported.

Arguments

Argument	Default Value	Description
img	(Mandatory)	The image file that is used as the background image of the plot area.
align	Center	The alignment of the background image relative to the plot area. See <a href="#">Alignment Specification</a> for possible alignment types.

Return Value

None

## setGridColor

PHP/Perl/Python Usage

setGridColor(hGridColor [, vGridColor])

C++ Prototype

virtual void setGridColor(int hGridColor, int vGridColor = Transparent) = 0;

#### Description

Sets the horizontal and vertical grid colors of the plot area. To disable the grids, simply set their colors to Transparent.

Note that you may also specify the grid colors when you define the plot area using the [setPlotArea](#) method of the XYChart object.

#### Arguments

Argument	Default Value	Description
hGridColor	(Mandatory)	The horizontal grid color.
vGridColor	Transparent	The vertical grid color. By default, it is Transparent, meaning that the vertical grid is invisible.

#### Return Value

None

## setGridWidth

PHP/Perl/Python Usage

setGridWidth(hGridWidth [, vGridWidth])

#### C++ Prototype

virtual void setGridWidth(int hGridWidth, int vGridWidth = -1) = 0;

#### Description

Set the horizontal and vertical grid line width.

#### Arguments

Argument	Default Value	Description
hGridWidth	(Mandatory)	The horizontal grid line width.
vGridWidth	-1	The vertical grid line width. The default value of -1 means the vertical grid line width is the same as the horizontal grid line width.

#### Return Value

None

## BaseAxis

The BaseAxis is the base class of the [XAxis](#) and [YAxis](#) in ChartDirector. It represents methods that are common to both axis classes.

Method	Description
<a href="#">setLabelStyle</a>	Set the font style used to for the axis labels.



<a href="#"><u>setLabelGap</u></a>	Set the distance between the axis labels and the ticks on the axis.
<a href="#"><u>setTitle</u></a>	Add a title to the axis.
<a href="#"><u>setTitlePos</u></a>	Set the title position relative to the axis.
<a href="#"><u>setColors</u></a>	Set the axis color, axis label color, axis title color and axis tick color.
<a href="#"><u>setWidth</u></a>	Set the line width of the axis.
<a href="#"><u>setTickLength</u></a>	Set the axis ticks length in pixels.
<a href="#"><u>setTickLength2</u></a>	Set the major and minor axis ticks lengths in pixels.
<a href="#"><u>addMark</u></a>	Add a mark line to the chart.
<a href="#"><u>addZone</u></a>	Add a zone to the chart.

## setLabelStyle

PHP/Perl/Python Usage

setLabelStyle([font [, fontSize [, fontColor [, fontAngle]]]])

C++ Prototype

```
virtual TextBox *setLabelStyle(const char *font = 0, double fontSize = 8, int fontColor = TextColor,
double fontAngle = 0) = 0;
```

Description

Set the font style used to for the axis labels. For details about how to specify font style, please refer to the section on [Font Specification](#).

Arguments

Argument	Default Value	Description
font	""	The font used to draw the labels. An empty string "" means using the default font (Arial).
fontSize	8	The font size used to draw the labels.
fontColor	TextColor	The color used to draw the labels.
fontAngle	0	Set the rotation of the font.

Return Value

A [TextBox](#) object that represents the Usage of the axis labels. You may use the methods of the TextBox object to fine-tune the appearance of the axis labels.

## setLabelGap

PHP/Perl/Python Usage

setLabelGap(d)

C++ Prototype

```
virtual void setLabelGap(int d) = 0;
```

Description

Set the distance between the axis labels and the ticks on the axis.

Arguments

Argument	Default Value	Description
d	(Mandatory)	The distance between the axis label and the tick in pixels.

Return Value

None

## setTitle

PHP/Perl/Python Usage

```
setTitle(text [, font [, fontSize [, fontColor]]])
```

C++ Prototype

```
virtual TextBox *setTitle(const char *text, const char *font = "arialbd.ttf", double fontSize = 8, int  
fontColor = TextColor) = 0;
```

Description

Add a title to the axis. For details about how to specify font style, please refer to the section on [Font Specification](#).

Arguments

Argument	Default Value	Description
text	(Mandatory)	The title text.
font	“arialbd.ttf”	The font used to draw the title. Default is Arial Bold (arialbd.ttf).
fontSize	8	The size of the font. Default is 8 points font.
fontColor	TextColor	The color of the font. Default is the <a href="#">TextColor</a> from the color palette.

Return Value

A [TextBox](#) object that represents the axis title. You may use the methods of the TextBox object to fine-tune the appearance of the axis title.

## setTitlePos

PHP/Perl/Python Usage

```
setTitlePos(alignment [, titleGap])
```

C++ Prototype

```
virtual void setTitlePos(Alignment alignment, int titleGap = 6) = 0;
```

Description

Set the title position relative to the axis.

By default, the axis title will be drawn at the middle of the axis outside the plot area. You may change the location of the title. For example, instead of drawing the x-axis title at the middle of the axis, you may want draw at the end of the axis.

The current version of ChartDirector supports the following alignment positions when drawing axis titles.

Axis	Default Position	Supported Position
Bottom x-axis	BottomCenter	TopLeft, TopRight, TopCenter, Left, Right, BottomLeft , BottomRight, BottomCenter :
Top x-axis	TopCenter	TopLeft, TopRight, TopCenter, Left, Right, BottomLeft , BottomRight, BottomCenter
Left y-axis	Left	Left, TopLeft
Right y-axis	Right	Right, TopRight

Besides deciding where the put the title, this method also specifies the distance between the axis title and the “whole axis”, where the “whole axis” in this case includes the labels and ticks.

Arguments

Argument	Default Value	Description
alignment	(Mandatory)	The position of the title relative to the axis. Please refer to the table above for the valid values.
titleGap	6	The distance between the axis title and the “whole axis” in pixels.

Return Value

None

## setColors

PHP/Perl/Python Usage

```
setColors(axisColor [, labelColor [, titleColor [, tickColor]]])
```

C++ Prototype

```
virtual void setColors(int axisColor, int labelColor = TextColor, int titleColor = -1, int tickColor = -1) = 0;
```

Description

Set the axis color, axis label color, axis title color and axis tick color.

By default, the axis and axis ticks are drawn using the [LineColor](#) in the color palette, while the axis label and axis title are drawn using the [TextColor](#) in the color palette. You may use this method to change the colors.

#### Arguments

Argument	Default Value	Description
axisColor	(Mandatory)	The color of the axis itself.
labelColor	<a href="#">TextColor</a>	The color of the axis labels.
titleColor	-1	The color of the axis title. The default value of -1 means the axis title color is the same as the axis label color.
tickColor	-1	The color of the axis ticks. The default value of -1 means the axis ticks color is the same as the axis color.

#### Return Value

None

## setWidth

PHP/Perl/Python Usage

setWidth(width)

C++ Prototype

```
virtual void setWidth(int width) = 0;
```

#### Description

Set the line width of the axis.

#### Arguments

Argument	Default Value	Description
width	(Mandatory)	The line width of the axis in pixels.

#### Return Value

None

## setTickLength

PHP/Perl/Python Usage

setTickLength(majorTickLen)

C++ Prototype

```
virtual void setTickLength(int majorTickLen) = 0;
```

#### Description

Set the axis ticks length in pixels. A positive value means the ticks are drawn outside the plot area. A negative value means the ticks are drawn inside the plot area.

#### Arguments

Argument	Default Value	Description
majorTickLen	(Mandatory)	The length of the major ticks in pixels. The length of the minor ticks will be set to half the length of the major ticks. Note that in the current version of ChartDirector, only the x-axis supports minor ticks.

#### Return Value

None

## setTickLength2

PHP/Perl/Python Usage

```
setTickLength2(majorTickLen, minorTickLen)
```

#### C++ Prototype

```
virtual void setTickLength(int majorTickLen, int minorTickLen) = 0;
```

#### Description

Set the major and minor axis ticks lengths in pixels. A positive value means the ticks are drawn outside the plot area. A negative value means the ticks are drawn inside the plot area.

#### Arguments

Argument	Default Value	Description
majorTickLen	(Mandatory)	The length of the major ticks in pixels.
minorTickLen	(Mandatory)	The length of the minor ticks in pixels. Note that in the current version of the ChartDirector, only the x-axis supports minor ticks.

#### Return Value

None

## addMark

PHP/Perl/Python Usage

```
addMark(value, lineColor [, text [, font [, fontSize]]])
```

#### C++ Prototype

```
virtual Mark *addMark(double value, int lineColor, const char *text = 0, const char *font = 0, double  
fontSize = 8) = 0;
```

#### Description

Add a mark line to the chart.

A mark line is a line drawn on the plot area. This line is usually used to indicate some special values, such as a “target value”, “threshold value”, “target date”, etc.

A mark line drawn using the x-axis will be vertical across the plot area. A mark line drawn using the y-axis will be horizontal across the plot area. In either case, the mark line label will be added to the axis at the mark line position.

The location of the mark line label can be changed by using the [setAlignment](#) method of the returned Mark object. For example, by setting the alignment to TopCenter, the mark line label will be drawn on the top center of the mark line.

By default, the mark line is drawn at the front of the chart layers. You may change it to draw at the back of the plot area (that is, like grid lines) by using the [setDrawOnTop](#) method of the Mark object returned by the addMark method.

#### Arguments

Argument	Default Value	Description
value	(Mandatory)	The value on the axis to draw the mark line. For x-axis, the first data point is value 0, while the n <sup>th</sup> data point is value (n-1). For y-axis, it is simply the value on the y-axis.
lineColor	(Mandatory)	The color of the mark line.
text	""	The text label for the mark line. Only applicable for y-axis based mark lines. The default value of "" (empty string) means there is no text label. By default, the text label and the tick on the y-axis will be drawn using the same color as the mark line. You can modify the colors by using the <a href="#">setMarkColor</a> method of the returned Mark object.
font	""	The font used to draw the text label. Only applicable for y-axis based mark lines. The default value of "" (empty string) means using the default font (Arial).
fontSize	8	The font size used to draw the text label. Only applicable for y-axis based mark lines. Default is 8 points.

#### Return Value

A [Mark](#) object representing the mark line added. You may use this object to fine-tune the appearance of the mark, such as its line width and colors.

## addZone

PHP/Perl/Python Usage

`addZone(startValue, endValue, color)`

C++ Prototype

`virtual void addZone(double startValue, double endValue, int color) = 0;`

## Description

Add a zone to the chart.

A zone is a range of values. For example, “10 to 20” is a zone. Typically, a zone is used to classify the data values. For example, you may classify 0 – 60 as the normal zone, 60 – 90 as the warning zone, and 90 – 100 as the critical zone.

A zone based on the x-axis will be drawn as a vertical band, while a zone based on the y-axis will be drawn as a horizontal band. Zones are always drawn at the back of the plot area.

## Arguments

Argument	Default Value	Description
startValue	(Mandatory)	The start value (the lower bound) on the axis for the zone. For x-axis, the first data point is value 0, while the n <sup>th</sup> data point is value (n-1). For y-axis, it is simply the value on the y-axis.
endValue	(Mandatory)	The end value (the upper bound) on the axis for the zone. For x-axis, the first data point is value 0, while the n <sup>th</sup> data point is value (n-1). For y-axis, it is simply the value on the y-axis.
color	(Mandatory)	The color of the zone.

## Return Value

None

# XAxis

The `XAxis` class represents the x-axes in an `XYChart`. Each `XYChart` has two x-axes, one on the bottom of the plot area, and one on the top of the plot area. The bottom x-axis is the primary x-axis and can be obtained by using the `xAxis` property (member variable) of the `XYChart` object. The top x-axis is the secondary x-axis and can be obtained by using the `xAxis2` property (member variable) of the `XYChart` object.

The `XAxis` class is a subclass of the `BaseAxis` class.

Method	Description
<a href="#"><code>setLabels</code></a>	Set the labels to be used on the x-axis.
<a href="#"><code>setLinearScale</code></a>	Set the numeric scale on the x-axis.
<a href="#"><code>setLinearScale2</code></a>	Set the numeric scale and the text labels to be used on the x-axis.
<a href="#"><code>addLabel</code></a>	Add an extra label on the x-axis.
<a href="#"><code>setIndent</code></a>	Specify where the x-axis should be “indented” or not.
<b>Methods inherited from <code>BaseAxis</code></b>	

<a href="#"><u>setLabelStyle</u></a>	Set the font style used to for the axis labels.
<a href="#"><u>setLabelGap</u></a>	Set the distance between the axis labels and the ticks on the axis.
<a href="#"><u>setTitle</u></a>	Add a title to the axis.
<a href="#"><u>setTitlePos</u></a>	Set the title position relative to the axis.
<a href="#"><u>setColors</u></a>	Set the axis color, axis label color, axis title color and axis tick color.
<a href="#"><u>setTickLength</u></a>	Set the axis ticks length in pixels.
<a href="#"><u>setTickLength2</u></a>	Set the major and minor axis ticks lengths in pixels.
<a href="#"><u>addMark</u></a>	Add a mark line to the chart.
<a href="#"><u>addZone</u></a>	Add a zone to the chart.

## setLabelStyle

PHP/Perl/Python Usage

setLabelStyle(text)

C++ Prototype

```
virtual TextBox *setLabelStyle(int noOfLabels, const char* const* text) = 0;
```

Description

Set the labels to be used on the x-axis.

This method is designed to be used to set the labels for charts using enumerated x values. In a chart using enumerated x values, there is no need to specify the numeric scale on the x-axis. It is only necessary to specify the text labels used on the x-axis. The first text label corresponds for the first data point, the second text label corresponds to the second data point, and so on.

For more details on what are enumerated x values, please refer to the [setXData](#) method of the Layer object.

By default, all labels will be drawn with major axis ticks. If you want to draw a label with a minor axis tick, use the '-' character as the first character of the label. If you want to draw the label without a tick at all, use the '~' character as the first character of the label. The '-' and '~' characters are special characters and will not appear on the actual label. It just tells ChartDirector that the label should be associated with a minor tick or no tick at all.

If you have a label that begins with '-' or '~' and you do not want the ChartDirector to interpret it as special characters, add the '\' character as the first character of the label.

In some cases, it may be desirable to skip some labels (e.g. draw one label per five data points). If you just want to draw a major tick without any labels, use a space character " " as the label. If you just want to draw a minor tick without any labels, use the "-" string. If you just want to leave a label position empty without a tick or a label, use an empty string "".



#### Arguments

Argument	Default Value	Description
text	(Mandatory)	A list of strings containing the text of the labels. By default, all label positions will have a major tick. If the first character of the labels are '-' or '~', the labels will be associated with a minor tick or no tick at all.

#### Return Value

A [TextBox](#) object that represents the Usage of the axis labels. You may use the methods of the TextBox object to fine-tune the appearance of the axis labels.

## setLinearScale

PHP/Perl/Python Usage

setLinearScale(lowerLimit, upperLimit [, tickInc ])

#### C++ Prototype

```
virtual void setLinearScale(double lowerLimit, double upperLimit, double tickInc = 0) = 0;
```

#### Description

Set the numeric scale on the x-axis.

Note that for most charts, it is not necessary to explicitly specify the x-axis scale. ChartDirector will automatically scale the x-axis based on enumerated x values. You just need to specify the labels on the x-axis using the [setLabels](#) method of the XAxis object. For more details on what are enumerated x values, please refer to the [setXData](#) method of the Layer object.

On the other hand, for charts that are not using enumerated x values, you may need to specify the x-axis scale using the [setLinearScale](#) or [setLinearScale2](#) method of the XAxis object.

In the setLinearScale method, the ticks and labels on the x-axis are specified using a tick increment parameter. In the setLinearScale2 method, the ticks and labels on the x-axis are specified as an array of text string.

#### Arguments

Argument	Default Value	Description
lowerLimit	(Mandatory)	The lower bound of the x-axis.
upperLimit	(Mandatory)	The upper bound of the x-axis.

tickInc	0	Specify that a tick with an associated label should be drawn on the x-axis every tickInc units. For example, if the lower bound is 0, the upper bound is 100, and the tickInc is 20, the axis ticks and labels will be at 0, 20, 40, 60, 80 and 100.  The default value of 0 means no tick and label will be added to the x-axis. (You can use the <a href="#">addLabel</a> method of the XAxis object to add ticks and labels to the x-axis.)
---------	---	--

Return Value

None

## setLinearScale2

PHP/Perl/Python Usage

setLinearScale2(lowerLimit, upperLimit, labels)

C++ Prototype

virtual void setLinearScale(double lowerLimit, double upperLimit, StringArray labels) = 0;

Description

Set the numeric scale and the text labels to be used on the x-axis.

Note that for most charts, it is not necessary to explicitly specify the x-axis scale. ChartDirector will automatically scale the x-axis based on enumerated x values. You just need to specify the labels on the x-axis using the [setLabels](#) method of the XAxis object. For more details on what are enumerated x values, please refer to the [setXData](#) method of the Layer object.

On the other hand, for charts that are not using enumerated x values, you may need to specify the x-axis scale using the [setLinearScale](#) or [setLinearScale2](#) method of the XAxis object.

In the setLinearScale method, the ticks and labels on the x-axis are specified using a tick increment parameter. In the setLinearScale2 method, the ticks and labels on the x-axis are specified as an array of text string.

Arguments

Argument	Default Value	Description
lowerLimit	(Mandatory)	The lower bound of the x-axis.
upperLimit	(Mandatory)	The upper bound of the x-axis.
labels	(Mandatory)	An array of text strings to act as the labels on the x-axis. The text strings are assumed to be equally spaced on the x-axis. By default, all text label positions will have a major tick. If the first character of the labels are '-' or '~', the labels will be associated with a minor tick or no tick at all.

Return Value  
None

## addLabel

PHP/Perl/Python Usage  
addLabel(pos, label)

C++ Prototype  
virtual void addLabel(double pos, const char \*label) = 0;

Description  
Add an extra label on the x-axis.

Normally, you can specify all the x-axis labels using the [setLabels](#), [setLinearScale](#) or [setLinearScale2](#) method of the XAxis object.

The addLabel method allows you to add extra labels at any arbitrary position on the x-axis.

### Arguments

Argument	Default Value	Description
pos	(Mandatory)	The x value of the position to add the label. Note that for an enumerated x-axis, the x position of the first data point is at value 0, while the n <sup>th</sup> data point is at value (n-1).
label	(Mandatory)	The text label to be added to the x-axis.

Return Value  
None

## setIndent

PHP/Perl/Python Usage  
setIndent(indent)

C++ Prototype  
virtual void setIndent(bool indent) = 0;

Description  
Specify where the x-axis should be “indented” or not.

Normally, the x-axis is automatically scaled so that x coordinate of first data point is at the beginning of the x-axis (that is, at the bottom left corner of the plot area), while the x coordinate last data point is at the end of the axis (that is, at the bottom right corner of the plot area).

However, for bar charts, if the x-axis is scaled as above, for the first bar and the last bar, half of them will be outside the plot area.

Therefore if any of the `ChartDirector` layers is a bar chart layer, the x-axis scaling will be set to “indented”. In the “indented” mode, the first data point will be shifted to the left, and the last data point will be shifted to the right, so that the first and last bars are completely within the plot area.

The `setIndent` method allows you to override the default x-axis scaling mode.

#### Arguments

Argument	Default Value	Description
indent	(Mandatory)	A “true” (non-zero) value sets the x-axis to indented mode. A “false” value set the x-axis to non-indented mode. By default, the x-axis will be in indented mode if one of the layers is a bar chart layer. Otherwise the x-axis will be in non-indented mode.

#### Return Value

None

## YAxis

The `YAxis` class represents the y-axes in an `XYChart`. Each `XYChart` has two y-axes – the primary y-axis and the secondary y-axis. The primary y-axis can be obtained by using the `yAxis` property (member variable) of the `XYChart` object, while the secondary y-axis can be obtained by using the `yAxis2` property (member variable) of the `XYChart` object.

Normally, the primary y-axis is drawn on the left side of the plot area, while the secondary y-axis is drawn on the right side of the plot area. You may reverse the location of the axes by using the `setYAxisOnRight` method of the `XYChart` object.

The `YAxis` class is a subclass of the `BaseAxis` class.

Method	Description
<a href="#"><code>setLinearScale</code></a>	Set the axis to use linear scaling and manually determine the scale.
<a href="#"><code>setLogScale</code></a>	Specify the type of auto-scaling (log or linear) for the y-axis.
<a href="#"><code>setLogScale2</code></a>	Set the axis to use log scaling and manually determine the scale.
<a href="#"><code>setAutoScale</code></a>	Reserve some space at the top and/or bottom of the plot area by using a larger axis scaling than is necessary, and control whether to include the zero point in auto-scaling.
<a href="#"><code>setTickDensity</code></a>	Set the density of the axis ticks.
<a href="#"><code>setTopMargin</code></a>	Reserve a range at the top of the plot area that is not scaled at all.
<a href="#"><code>setLabelFormat</code></a>	Set the number format for the axis labels.
<b>Methods inherited from <code>BaseAxis</code></b>	
<a href="#"><code>setLabelStyle</code></a>	Set the font style used to for the axis labels.

<u>setLabelGap</u>	Set the distance between the axis labels and the ticks on the axis.
<u>setTitle</u>	Add a title to the axis.
<u>setTitlePos</u>	Set the title position relative to the axis.
<u>setColors</u>	Set the axis color, axis label color, axis title color and axis tick color.
<u>setTickLength</u>	Set the axis ticks length in pixels.
<u>setTickLength2</u>	Set the major and minor axis ticks lengths in pixels.
<u>addMark</u>	Add a mark line to the chart.
<u>addZone</u>	Add a zone to the chart.

## setLinearScale

PHP/Perl/Python Usage

setLinearScale(lowerLimit, upperLimit [, tickInc])

C++ Prototype

virtual void setLinearScale(double lowerLimit, double upperLimit, double tickInc = 0) = 0;

Description

Set the axis to use linear scaling and manually determine the scale.

If you do not call this method and any other axis scaling method, the ChartDirector will use a linear y-axis with auto-scaling.

Arguments

Argument	Default Value	Description
lowerLimit	(Mandatory)	The lower bound of the y-axis.
upperLimit	(Mandatory)	The upper bound of the y-axis.
tickInc	0	The spacing between the y-axis ticks, representing as the difference in y values between two adjacent ticks. A value of 0 means the tick spacing will be automatically determined. In this case, the ChartDirector may adjust the lower bound or upper bound of the axis to in order to find a reasonable tick spacing (e.g. the tick spacing has to be a neat number, no too close or too sparse, and the axis range must be a integer multiple of the tick spacing, and other constraints, etc). If you do not want to lower bound and upper bound to be adjusted, specify the tick spacing explicitly.

Return Value

None.

## setLogScale

PHP/Perl/Python Usage

setLogScale([logScale])

C++ Prototype

```
virtual void setLogScale(bool logScale = true) = 0;
```

Description

Specify the type of auto-scaling (log or linear) for the y-axis.

If you do not call this method and any other axis scaling method, the ChartDirector will use a linear y-axis with auto-scaling.

Arguments

Argument	Default Value	Description
logScale	1	The default value of “true” (non-zero) means the axis will use log scaling. A “false” value means linear scaling.

Return Value

None.

## setLogScale2

PHP/Perl/Python Usage

setLogScale2(lowerLimit, upperLimit [, tickInc])

C++ Prototype

```
virtual void setLogScale(double lowerLimit, double upperLimit, double tickInc = 0) = 0;
```

Description

Set the axis to use log scaling and manually determine the scale.

If you do not call this method and any other axis scaling method, the ChartDirector will use a linear y-axis with auto-scaling.

Arguments

Argument	Default Value	Description
lowerLimit	(Mandatory)	The lower bound of the y-axis.
upperLimit	(Mandatory)	The upper bound of the y-axis.

tickInc	0	The spacing between the y-axis ticks, representing as the ratio in y values between two adjacent ticks. A value of 0 means the tick spacing will be automatically determined. In this case, the ChartDirector may adjust the lower bound or upper bound of the axis to in order to find a reasonable tick spacing (e.g. the tick ratio has to be a neat number, no too close or too sparse, and the axis range must be a integer multiple of the tick spacing, and other constraints, etc). If you do not want to lower bound and upper bound to be adjusted, specify the tick spacing explicitly.
---------	---	--

Return Value  
None.

## setAutoScale

PHP/Perl/Python Usage

setAutoScale([topExtension [, bottomExtension [, zeroAffinity ]]])

C++ Prototype

```
virtual void setAutoScale(double topExtension = 0, double bottomExtension = 0, double zeroAffinity = 0.8) = 0;
```

Description

Control the margin at the top and bottom of the axis when performing auto-scaling, and whether to start the axis from zero.

One common use of setAutoScale is to reserve space at the top portion of the plot area for the legend box, custom text box or other custom chart elements. Another common use is to make sure the data labels (which are drawn on top of the data points) will not fall outside the plot area.

For example, a topExtension of 0.2 will ensure the top 20% of the plot area is empty. ChartDirector achieves this by using a larger y-axis scale than is necessary, so that no data points can fall within the top 20% of the plot area.

For example, if the data is in the range 0 – 100, and we use an axis scaling of 0 – 150, the top portion of the plot area will remain empty because no data are in that range.

Note that there is an alternative way to reserve space at the top of the plot area – the [setTopMargin](#) method.

The bottomExtension is similar to the top extension. It applies to the bottom of the plot area if the lower bound of the axis scale is non-zero. The bottomExtension has no effect if the lower bound of the axis is zero.

Note that for the lower bound of the axis, ChartDirector has “zero affinity”. That means ChartDirector will try to include zero in the axis unless it is “unreasonable” to do so.

For example, if the data is in the range 10000 – 10005, it may be unreasonable to start the axis from zero. On the other hand, if the data is in the range 10 – 15, it is reasonable to start the axis from zero.

In general, if the data range is “too small” compare with the value of the data, it is unreasonable to start the axis from zero. ChartDirector test the “too small” condition by using the formula:

$$\text{maxDataValue} * \text{zeroAffinity} < \text{minDataValue}$$

where zeroAffinity by default is 0.8.

(The above formula is for positive data values. A slightly different formula is used for negative data values. For data that has both positive and negative values, the zero point is always included.)

In other words, if the minimal data value is greater than 80% of the maximum data value, ChartDirector will consider the data range too small, and will not start the axis from zero.

The zeroAffinity parameter of the setAutoScale method allows you to modify the zero affinity when performing auto-scaling. The zero affinity should be between 0 and 1. A large value will encourage ChartDirector to start the axis from zero.

A zero affinity of 1 means the axis always includes the zero point. A zero affinity of 0 means that the axis is scaled purely according to the data range, without any preference for the zero value.

#### Arguments

Argument	Default Value	Description
topExtension	0	The top portion of the y-axis that no data should fall into. The top portion must be between 0 – 1. For example, a value of 0.2 means no data value will fall within the top 20% of the y-axis.
bottomExtension	0	The bottom portion of the y-axis that no data should fall into. The bottom portion must be between 0 – 1. For example, a value of 0.2 means no data value will fall within the bottom 20% of the y-axis.
zeroAffinity	0.8	<p>The tendency of ChartDirector to include zero in the axis during auto-scaling. ChartDirector will include zero in the axis unless the data range is “too small” compare to the data value, where “too small” is defined as:</p> $\text{maxDataValue} * \text{zeroAffinity} < \text{minDataValue}$ <p>The zero affinity should be between 0 and 1. A large value will encourage ChartDirector to start the axis from zero. A zero affinity of 1 means the axis always includes the zero point. A zero affinity of 0 means there is no preference for the zero point during auto-scaling.</p>

Return Value  
None.



## setTickDensity

PHP/Perl/Python Usage

setTickDensity(tickDensity)

C++ Prototype

```
virtual void setTickDensity(int tickDensity) = 0;
```

Description

Set the density of the axis ticks.

Arguments

Argument	Default Value	Description
tickDensity	(Mandatory)	Specify the desired distance between two ticks in pixels. The ChartDirector will auto-scale the axis to try to meet the tick density requirement, but it may not meet it exactly. It is because the ChartDirector has other constraints to consider, such as the ticks and axis range should be neat numbers, and the axis must contain an integral number of ticks, etc. The ChartDirector therefore may use a tick distance that is larger than specified, but never smaller.

Return Value

None.

## setTopMargin

PHP/Perl/Python Usage

setTopMargin(topMargin)

C++ Prototype

```
virtual void setTopMargin(int topMargin) = 0;
```

Description

Reserve a range at the top of the plot area that is not scaled at all. No data will fall within that range. The y-axis at that range will contain no tick, label and grid line.

This method is usually used when you want to reserve some space at the top of the plot area for something, such as a legend box, or some custom text.

Note that there is an alternative way to reserve space at the top and/or bottom of the plot area – the [setAutoScale](#) method.

Arguments

Argument	Default Value	Description
----------	---------------	-------------

topMargin	(Mandatory)	The height of the top of the plot area that is reserved space in pixels.
-----------	-------------	--

Return Value  
None.

## setLabelFormat

PHP/Perl/Python Usage  
setLabelFormat(formatString)

C++ Prototype  
virtual void setLabelFormat(const char \*formatString) = 0;

Description  
Set the number format for the axis labels.

By default, the data label format is “{value}”, which means the label will contain the numeric value at the tick position using the default number format.

You can control the formatting of the axis labels by supplying an appropriate format string. Please refer to the section on [Data and Text Formatting](#) for details.

Arguments

Argument	Default Value	Description
formatString	(Mandatory)	The format string. See above for description.

Return Value  
None

## Mark

The Mark class represents mark lines. A mark line is a line drawn on the plot area. This line is usually used to indicate some special values, such as a “target value”, “threshold value”, “target date”, etc.

A mark line is created using the [addMark](#) method of the BaseAxis object. A mark line drawn using the x-axis will be vertical across the plot area. A mark line drawn using the y-axis will be horizontal across the plot area.

By default, the mark line is drawn at the front of the chart layers. You may change it to draw at the back of the plot area (that is, like grid lines) by using the [setDrawOnTop](#) method.

Method	Description
<a href="#">setValue</a>	Set the value of the mark line.
<a href="#">setMarkColor</a>	Set the line, text and tick colors of the mark line.

<u>setLineWidth</u>	Set the line width of the mark line.
<u>setDrawOnTop</u>	Determine whether the mark line is drawn at the front of the chart layers, or at the back of the plot area (that is, like grid lines).

## setValue

PHP/Perl/Python Usage

setValue(value)

C++ Prototype

virtual void setValue(double value) = 0;

Description

Set the value of the mark line.

Arguments

Argument	Default Value	Description
value	(Mandatory)	The y value of the mark.

Return Value

None

## setMarkColor

PHP/Perl/Python Usage

setMarkColor(lineColor [, textColor [, tickColor]])

C++ Prototype

virtual void setMarkColor(int lineColor, int textColor = -1, int tickColor = -1) = 0;

Description

Set the line, text and tick colors of the mark line.

Arguments

Argument	Default Value	Description
lineColor	(Mandatory)	The color of the mark line.
textColor	-1	The color of the text label that will be shown on the y-axis. The default value of -1 means the text label color is the same as the line color.
tickColor	-1	The color of the tick that will be shown on the y-axis. The default value of -1 means the tick color is the same as the line color.

Return Value  
None

## setLineWidth

PHP/Perl/Python Usage  
setLineWidth(w)

C++ Prototype  
virtual void setLineWidth(int w) = 0;

Description  
Set the line width of the mark line.

Arguments

Argument	Default Value	Description
w	(Mandatory)	The mark line width in pixels.

Return Value  
None

## setDrawOnTop

PHP/Perl/Python Usage  
setDrawOnTop(b)

C++ Prototype  
virtual void setDrawOnTop(bool b) = 0;

Description  
Determine whether the mark line is drawn at the front of the chart layers, or at the back of the plot area (that is, like grid lines).

Arguments

Argument	Default Value	Description
b	(Mandatory)	Set to “true” (non-zero) for drawing the mark line at the front of the chart layers. Set to “false” (zero) for drawing the mark line at the back of the plot area.

Return Value  
None

## Layer

The Layer class is a base class for all [XYChart](#) layer classes.

Method	Description
--------	-------------

<u>set3D</u>	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
<u>setLineWidth</u>	Set the default line width of data lines when drawing data sets on the layer.
<u>setBorderColor</u>	Set the default border color and border effect for drawing all the data sets on the layer.
<u>setDataCombineMethod</u>	Set the method used to combine multiple data sets in a layer.
<u>setLegend</u>	Set the order of the data names when entered into the legend box.
<u>addDataSet</u>	Add a data set to the chart layer.
<u>getDataSet</u>	Get the requested DataSet object.
<u>setXData</u>	Set the x values of the data points in the data sets.
<u>setXData2</u>	Set the x values of the data points in the data sets as evenly distributed in a range.
<u>getXCoor</u>	Get the x coordinate of a point given the x value.
<u>getYCoor</u>	Get the y coordinate of a point given the y value.
<u>setDataLabelFormat</u>	Set the data label format of the all data labels for all data sets in the layer.
<u>setDataLabelStyle</u>	Set the style used to draw data labels for all data sets in the layer.
<u>addCustomDataLabel</u>	Add a custom data label to a data point.
<u>setAggregateLabelFormat</u>	Set the data label format of the aggregate data labels.
<u>setAggregateLabelStyle</u>	Enable and set the style used to draw aggregate data labels in the layer.
<u>addCustomAggregateLabel</u>	Add a custom aggregate data label to an aggregated data point.
<u>getImageCoor</u>	Get the image map coordinates of a data point as drawn on the layer as HTML image map attributes.
<u>getImageCoor2</u>	Get the image map coordinates of all data points at the same x-axis position as drawn on the layer as HTML image map attributes.
<u>getHTMLImageMap</u>	Generate an HTML image map for all data points as drawn on the layer.

## set3D

PHP/Perl/Python Usage

set3D([d [, zGap]])

C++ Prototype

virtual void set3D(int d = -1, int zGap = 0) = 0;

#### Description

Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.

#### Arguments

Argument	Default Value	Description
d	-1	The 3D depth of the layer in pixels. The default value of -1 means the depth is automatically calculated. A value of 0 means the layer will be flat.
zGap	0	The 3D gap between the current layer and the next layer in pixels. The default value of 0 means there is no 3D gap, that is, the back of the current layer will be in touch with the front of the next layer.

#### Return Value

None.

## setLineWidth

PHP/Perl/Python Usage

setLineWidth(w)

#### C++ Prototype

```
virtual void setLineWidth(int w) = 0;
```

#### Description

Set the default line width of data lines when drawing data sets on the layer. This only applies to layers that employ lines to represent data.

If you want a certain data set to have a different line width from other data sets on the same layer, you may use the [setLineWidth](#) method of the [DataSet](#) object to override the default line width.

#### Arguments

Argument	Default Value	Description
w	(Mandatory)	The width of the line in pixels.

#### Return Value

None.

## setBorderColor

PHP/Perl/Python Usage

setBorderColor(color [, raisedEffect])

#### Description

Set the default border color and border effect for drawing all the data sets on the layer. This only applies to layers that draw objects with borders (e.g. bars in a bar chart).

If you want a certain data set to have a different line width from other data sets on the same layer, you may use the [setDataColor](#) method of the DataSet object to override the default colors.

If you do not call this method to setBorderColor, the border color will be the default [LineColor](#) from the current color palette.

#### Arguments

Argument	Default Value	Description
color	(Mandatory)	The border color.
raisedEffect	0	The width of the 3D effect for the border. For positive values, the box will appear raised. For negative values, the box will appear depressed. A zero value means no 3D border should be drawn, that is, the box will appear as flat.

#### Return Value

None.

## setDataCombineMethod

PHP/Perl/Python Usage

setDataCombineMethod(m)

#### Prototype

virtual void setDataCombineMethod(int m) = 0;

#### Description

Set the method used to combine multiple data sets in a layer. Usually this method is used only when a layer contains more than one data set. Note that some layers may not support some data combine methods.

Data Combine Method	Description
Stack	The multiple data sets are stacked up on top of one another. For example, on a bar chart, the data will be represented as stacked bars.
Overlay	The multiple data sets are plotted independently, overlapping each others. For example, on a line chart, the multiple data sets will be represented by multiple lines.
Side	The multiple data sets are plotted side-by-side. Currently, this method only applies to bar charts.

#### Arguments

Argument	Default Value	Description
m	(Mandatory)	The data combine method, which must be Side, Stack or Overlay.

Return Value  
None

## setLegend

PHP/Perl/Python Usage  
setLegend(m)

C++ Prototype  
virtual void setLegend(LegendMode m) = 0;

Description  
Set the order of the data names when entered into the legend box.

By default, if a data set has a name (specified as the third parameter of the [addDataSet](#) method), an entry for that data set will be added to the legend box. If there are multiple data sets in a layer, multiple entries will be added to the legend box, in the same order as the data sets being added to the layer.

In some cases, it may be desirable not to add the data set names to the legend box, or to add them to the legend box in reverse order.

For example, in a stacked bar chart with positive data, the first data set will be at the bottom, and the last data set will be stacked to the top. If the legend box is layout vertically, by default the first legend key will be at the top. So it may be desirable to reverse the order that the data names are added to the legend box.

### Arguments

Argument	Default Value	Description
m	(Mandatory)	One of the following 3 predefined constants: <ul style="list-style-type: none"><li>NormalLegend: Data set names are added to the legend box in normal order.</li><li>ReverseLegend: Data set names are added to the legend box in reverse order.</li><li>NoLegend: Data set names are not added to the legend box at all.</li></ul>

Return Value  
None

## addDataSet

PHP/Perl/Python Usage  
addDataSet(d [, color [, name]])



C++ Prototype

```
virtual DataSet *addDataSet(DoubleArray data, int color = -1, const char *name = 0) = 0;
```

Description Description

Add a data set to the chart layer. A layer can contain multiple data sets.

Arguments

Argument	Default Value	Description
d	(Mandatory)	A list of numbers representing the values in the data set. Note that <a href="#">ChartDirector</a> supports the special constant <a href="#">NoValue</a> as values in the array to specify that certain data points have no value.
color	-1	The color used to draw the data set. A value of -1 means the color is automatic.
name	""	The name of the data set. If a legend box is available in the <a href="#">XYChart</a> , a key will be automatically added to the legend box to describe the data set. A value of "" (empty string) means the data set is unnamed.

Return Value

This method returns the [DataSet](#) object representing the data set added. You may use the methods of the [DataSet](#) object to fine-tune how the data set is drawn on the chart.

## getDataSet

PHP/Perl/Python Usage

```
getDataSet(dataSet)
```

C++ Prototype

```
virtual DataSet *getDataSet(int dataSet) = 0;
```

Description

Get the requested [DataSet](#) object.

Arguments

Argument	Default Value	Description
dataSet	(Mandatory)	A number specifying the <a href="#">DataSet</a> object to get. The first <a href="#">DataSet</a> object is 0, and the n <sup>th</sup> data set is (n-1).

Return Value

This method returns the requested [DataSet](#) object.

## setXData

PHP/Perl/Python Usage

setXData(xData)

C++ Prototype

```
virtual void setXData(DoubleArray xData) = 0;
```

Description

Set the x values of the data points in the data sets.

When a data set is added to ChartDirector using the [addDataSet](#) method (or other similar methods), only the y values are required. In most charts, it is not necessary to specify the x values for the data points.

For example, in a bar chart, you just need to specify the values of the bars (y values). ChartDirector will automatically layout the bars evenly on the x-axis. You can specify the x-axis labels for the bars by as an array of text strings using the [setLabels](#) method of the XAxis object. The same applies to all ChartDirector chart layers.

In the whole process, there is no need to specify the x values. Only the x-axis labels are required.

Internally, if the x values are not specified, ChartDirector uses “enumerated” x values. It assigns an x value of 0 to the first data point, 1 to the second data point, and so on. Of course, these values are not visible on the x-axis. Only the x-axis labels provided as an array of text strings are visible.

However, in some cases, it may be necessary to specify the x values explicitly. For example, if a line chart contains data points that are not evenly distributed on the x-axis, it is necessary to specify the x values explicitly. Otherwise, ChartDirector will use enumerated x values and distribute the data points evenly on the x-axis.

In general, if the data points are evenly distributed on the x-axis (such as a data set with one data point per hour, or one data point per day), it is recommended that the internal enumerated x values be used. In this case, there is no need to specify the x values. Even if some data points are missing, it is not necessary to specify the x values explicitly. Missing data points can be handled by using the [NoValue](#) constant.

On the other hand, if the data points are by its nature not evenly distributed, the x values should be provided explicitly.

Note that each chart layer only has one x values series. All data sets within the layer will use the same x value series. If two data sets have different x values, they should be put in two different layers.

Arguments

Argument	Default Value	Description
xData	(Mandatory)	An array of numbers representing the x values of the data in the data sets.

Return Value  
None

## setXData2

PHP/Perl/Python Usage  
setXData2(minValue, maxValue)

C++ Prototype  
virtual void setXData(double minValue, double maxValue) = 0;

Description  
Set the x values of the data points in the data sets as evenly distributed in a range.

This method is most useful when two layers contain data at different x axis scale. An example is a line chart layer with one data point per minute, and another line chart layer with one data point per 5 minutes.

In the above example, in one hour, the first layer will have 60 data points, while the second layer will have 12 data points. By default, ChartDirector automatically assigns enumerated x values to the data points. (Please refer to the description on [setXData](#) for details on enumerated x values.) The first layer will have x values from 0 – 59, while the second layer will have x values from 0 – 11. When the chart is drawn, it would look like the second layer contains 12 minutes of data, which is not correct.

To solve this problem, the setXData2 method can be used to specify that the x values of the second layer is from 0 – 55. The first data point will be at 0, while the last data point at 55, and other data points will be evenly distributed between 0 – 55. As there are 12 data points, it will be one data point per 5 minutes, which is correct.

### Arguments

Argument	Default Value	Description
minValue	(Mandatory)	The x value of the first point in a data set.
maxValue	(Mandatory)	The x value of the last point in the data set.

Return Value  
None

## getXCoord

PHP/Perl/Python Usage  
getXCoord(v)

C++ Prototype  
virtual int getXCoord(double v) = 0;

#### Description

Get the x coordinate of a point given the x value. This method is usually used when you want to add custom text or drawings to certain location of the

Note that this method must be called after you have called the [layout](#) method of the XYChart object to layout the chart first. It is because the ChartDirector needs to compute the axis auto-scaling and other things first before it can compute the coordinates.

#### Arguments

Argument	Default Value	Description
v	(Mandatory)	The x value. For an enumerated x-axis, the x value of the first data point is 0, and the x-value of the n <sup>th</sup> data point is n-1.

#### Return Value

The x coordinate of the x value.

### getYCoor

PHP/Perl/Python Usage

getYCoor(v [, yAxis])

#### C++ Prototype

```
virtual int getYCoor(double v, bool yAxis = true) = 0;
```

#### Description

Get the y coordinate of a point given the y value. Since there are two y-axes supported by the [XYChart](#) and they can be of different scale, you may need to specify which y-axis to use when computing the y coordinate. The default is to use the primary y-axis.

This method is usually used when you want to add custom text or lines to certain location of the chart. One common example is to add a text label to the highest data point of the data set.

Note that this method must be called after you have called the [layout](#) method of the XYChart object to layout the chart first. It is because the ChartDirector needs to compute the axis auto-scaling and other things first before it can compute the coordinates.

#### Arguments

Argument	Default Value	Description
v	(Mandatory)	The y value.
yAxis	1	Determine whether the y coordinate is computed using the scale on the primary y-axis or the secondary y-axis. The default value of “true” (non-zero) means that the primary y-axis will be used. A “false” value means the secondary y-axis will be used.

Return Value

The y coordinate of the y value.

## setDataLabelFormat

PHP/Perl/Python Usage

setDataLabelFormat(formatString)

C++ Prototype

```
virtual void setDataLabelFormat(const char *formatString) = 0;
```

Description

Set the data label format of the all data labels for all data sets in the layer. If you just want to set the label format for one particular data set only, use the [setDataLabelFormat](#) method of the DataSet object.

Data labels are labels that appear near the data points in the chart. The exact position depends on the chart type. Please refer to the description of the [setDataLabelStyle](#) method.

By default, the data label format is “{value}”, which means the label will contain the data value. You can include more information and control the formatting of the data label by supplying an appropriate format string. Please refer to the section on [Data and Text Formatting](#) for details.

Arguments

Argument	Default Value	Description
formatString	(Mandatory)	The format string. Please refer to the section on <a href="#">Data and Text Formatting</a> for details.

Return Value

None

## setDataLabelStyle

PHP/Perl/Python Usage

setDataLabelStyle(font [, fontSize [, fontColor [, fontAngle]]])

C++ Prototype

```
virtual TextStyle *setDataLabelStyle(const char *font = 0, double fontSize = 8, int fontColor = TextColor, double fontAngle = 0) = 0;
```

Description

Set the style used to draw data labels for all data sets in the layer. If you just want to set the style for one particular data set only, use the [setDataLabelStyle](#) method of the DataSet object.

Data labels are labels that appear near the data points in the chart. With the exception of the bar chart layer and the area chart layer, data labels are drawn on the top of the data point. This can be controlled using the [setAlignment](#) parameter of the TextBox object returned by this method.

For a bar chart layer, the exact locations of the data labels depend on the type of bar chart. For multi-bar charts, the data labels are drawn on the outside of the bar. For example, if the bar is positive and vertical, it will be on the top of the bar. For other kind of bar charts (stacked or overlay), the data labels are drawn internal to the bar. For example, if the bar is positive and vertical, it will be just under the top edge of the bar segments.

For an area chart, the data labels are drawn inside the area. For a positive area chart, the data label will be just under the data point. (The position above the data point is reserved for aggregate data label.)

When using data labels with y-axis auto-scaling, it is recommended that the [setAutoScale](#) or [setTopMargin](#) method of the YAxis object be used to reserve some margin at the top of the plot area. This ensures the data labels are always drawn within the plot area.

For details about how to specify font style, please refer to the section on [Font Specification](#).

#### Arguments

Argument	Default Value	Description
font	""	The font used to draw the labels. A "" (empty string) means using the default font (Arial).
fontSize	8	The font size used to draw the labels.
fontColor	<a href="#">TextColor</a>	The color used to draw the labels.
fontAngle	0	Set the rotation of the font.

#### Return Value

A [TextBox](#) object that represents the Usage of the data labels. You may use the methods of the TextBox object to fine-tune the appearance of the data labels.

## addCustomDataLabel

PHP/Perl/Python Usage

`addCustomDataLabel(dataSet, dataItem, label [, font [, fontSize [, fontColor [, fontAngle ]]])`

C++ Prototype

```
virtual TextBox *addCustomDataLabel(int dataSet, int dataItem, const char *label, const char *font = 0, double fontSize = 8, int fontColor = TextColor, double fontAngle = 0) = 0;
```

#### Description

Add a custom data label to a data point.

Data labels are labels that appear near the data points in the chart. The exact position depends on the chart type. Please refer to the description of the [setDataLabelStyle](#) method.

If data labels are enabled for all data points, the custom data label will override the normal data label for the specified data point.

ChartDirector supports parameter substitution for the label text. For example, you can use “{value}” to represent the value of the data point. Please refer to the section on [Data and Text Formatting](#) for details.

When using data labels with y-axis auto-scaling, it is recommended that the [setAutoScale](#) or [setTopMargin](#) method of the YAxis object be used to reserve some margin at the top of the plot area. This ensures the data labels are always drawn within the plot area.

For details about how to specify font style, please refer to the section on [Font Specification](#).

#### Arguments

Argument	Default Value	Description
dataSet	(Mandatory)	The data set number for the data point. The first data set is 0, while the $n^{\text{th}}$ data set is $(n - 1)$ .
dataItem	(Mandatory)	The data point number for the data point within the data set. The first data point is 0, while the $n^{\text{th}}$ data point is $(n - 1)$ .
label	(Mandatory)	A text string representing the data label. Supports parameter substitution in the text string. Please refer to the section on <a href="#">Data and Text Formatting</a> for details.
font	""	The font used to draw the label. The default value "" (empty string) means using the default font (Arial).
fontSize	8	The font size used to draw the label.
fontColor	<a href="#">TextColor</a>	The color used to draw the label.
fontAngle	0	Set the rotation of the font.

#### Return Value

A [TextBox](#) object that represents the custom data label. You may use the methods of the TextBox object to fine-tune the appearance of the custom data label.

## setAggregateLabelFormat

PHP/Perl/Python Usage

`setAggregateLabelFormat(formatString)`

C++ Prototype

```
virtual void setAggregateLabelFormat(const char *formatString) = 0;
```

#### Description

Set the data label format of the aggregate data labels.

Aggregate data labels only apply to stack and overlay chart types. In these chart types, the aggregate data labels represent the “stacked” data. Aggregate labels are drawn on the top of the stacked data (or bottom for negative stacked data).

By default, the aggregate data label format is “{value}”, which means the label will contain the aggregate data value. You can include more information and control the formatting of the data label by supplying an appropriate format string. Please refer to the section on [Data and Text Formatting](#) for details.

When using data labels with y-axis auto-scaling, it is recommended that the [setAutoScale](#) or [setTopMargin](#) method of the YAxis object be used to reserve some margin at the top of the plot area. This ensures the data labels are always drawn within the plot area.

#### Arguments

Argument	Default Value	Description
formatString	(Mandatory)	The format string. Please refer to the section on <a href="#">Data and Text Formatting</a> for details.

#### Return Value

None

## setAggregateLabelStyle

PHP/Perl/Python Usage

setAggregateLabelStyle([font [, fontHeight [, fontColor [, fontAngle]]]])

#### C++ Prototype

```
virtual TextStyle *setAggregateLabelStyle(const char *font = 0, double fontHeight = 8, int fontColor = TextColor, double fontAngle = 0) = 0;
```

#### Description

Enable and set the style used to draw aggregate data labels in the layer.

Aggregate data labels only apply to stack and overlay chart types. In these chart types, the aggregate data labels represent the “stacked” data. Aggregate labels are drawn on the top of the stacked data (or bottom for negative stacked data).

When using data labels with y-axis auto-scaling, it is recommended that the [setAutoScale](#) or [setTopMargin](#) method of the YAxis object be used to reserve some margin at the top of the plot area. This ensures the data labels are always drawn within the plot area.

For details about how to specify font style, please refer to the section on [Font Specification](#).

#### Arguments

Argument	Default Value	Description
font	""	The font used to draw the labels. A "" (empty string) means using the default font (Arial).
fontSize	8	The font size used to draw the labels.
fontColor	<a href="#">TextColor</a>	The color used to draw the labels.



fontAngle	0	Set the rotation of the font.
-----------	---	-------------------------------

#### Return Value

A [TextBox](#) object that represents the Usage of the aggregate data labels. You may use the methods of the TextBox object to fine-tune the appearance of the aggregate data labels.

## addCustomAggregateLabel

PHP/Perl/Python Usage

addCustomAggregateLabel(dataItem , label [, font [, fontSize [, fontColor [, fontAngle ]]])

C++ Prototype

```
virtual TextBox *addCustomAggregateLabel(int dataItem, const char *label, const char *font = 0,
double fontSize = 8, int fontColor = TextColor, double fontAngle = 0) = 0;
```

#### Description

Add a custom aggregate data label to an aggregated data point.

Aggregate data labels only apply to stack and overlay chart types. In these chart types, the aggregate data labels represent the “stacked” data. Aggregate labels are drawn on the top of the stacked data (or bottom for negative stacked data).

If aggregate data labels are enabled for all aggregated data points, the custom aggregate data label will override the normal aggregate data label for the specified aggregated data point.

ChartDirector supports parameter substitution for the label text. For example, you can use “{value}” to represent the value of the aggregated data point. Please refer to the section on [Data and Text Formatting](#) for details.

When using data labels with y-axis auto-scaling, it is recommended that the [setAutoScale](#) or [setTopMargin](#) method of the YAxis object be used to reserve some margin at the top of the plot area. This ensures the data labels are always drawn within the plot area.

For details about how to specify font style, please refer to the section on [Font Specification](#).

#### Arguments

Argument	Default Value	Description
dataItem	(Mandatory)	The data point number for the aggregated data point. The first aggregated data point is 0, while the n <sup>th</sup> aggregated data point is (n – 1).
label	(Mandatory)	A text string representing the data label. Supports parameter substitution in the text string. Please refer to the section on <a href="#">Data and Text Formatting</a> for details.
font	""	The font used to draw the labels. A "" (empty string) means using the default font (Arial).

fontSize	8	The font size used to draw the labels.
fontColor	<a href="#">TextColor</a>	The color used to draw the labels.
fontAngle	0	Set the rotation of the font.

Return Value

A [TextBox](#) object that represents the custom aggregate data label. You may use the methods of the TextBox object to fine-tune the appearance of the custom aggregate data label.

## getImageCoor

PHP/Perl/Python Usage

`getImageCoor(dataSet, dataItem)`

C++ Prototype

`virtual const char *getImageCoor(int dataSet, int dataItem) = 0;`

Description

Get the image map coordinates of a data point as drawn on the layer as HTML image map attributes in the following format:

```
shape="<shape>" cords="<x1>,<y1>,<x2>,<y2> ..."
```

This format is specially designed so that it can easily be included into HTML image maps.

This method gets the image map for a data point within a data set. For a stacked bar chart, it will get the image map for one segment of a bar. If you want to get the image map for the entire bar, which contains multiple data points at the same x-axis position, use the [getImageCoor2](#) method of the Layer object. The same applies to other kinds of charts (area charts, line charts, etc).

If you want to get an image map for all the data points for all data sets in a layer, use the [getHTMLImageMap](#) method of the Layer object. If you want to get the complete image map for all the layers, use the [getHTMLImageMap](#) method of the BaseChart object instead.

This method should be called only after you have completely drawn the chart using the [makeChart](#), [makeChart2](#) or [makeChart3](#) method. It is because ChartDirector needs to actually draw the chart to compute the coordinates of the data points on the layer.

Arguments

Argument	Default Value	Description
dataSet	(Mandatory)	The data set number for the data set that contains the data point. The first data set is 0, while the n <sup>th</sup> data set is (n-1).
dataItem	(Mandatory)	The position of the data point within the data set. The first data point is 0, while the n <sup>th</sup> data point is (n-1).

#### Return Value

Return a text string representing the coordinates of the data point as drawn on the layer in HTML image map attribute format.

## getImageCoor2

PHP/Perl/Python Usage

getImageCoor2(dataItem)

#### C++ Prototype

```
virtual const char *getImageCoor(int dataItem) = 0;
```

#### Description

Get the image map coordinates of all data points at the same x-axis position as drawn on the layer as HTML image map attributes. The resulting image map will be of the following format:

```
shape="<shape>" cords="<x1>,<y1>,<x2>,<y2> ..."
```

This format is specially designed so that it can easily be included into HTML image maps.

This method gets the image map for all data points within the same x-axis position. For a stacked bar chart, it will get the image map for the entire bar, which contains multiple data points. If you want to get the image map for one particular data point, use the [getImageCoor](#) method of the Layer object. The same applies to other kinds of charts (area charts, line charts, etc).

If you want to get an image map for all the data points for all data sets in a layer, use the [getHTMLImageMap](#) method of the Layer object. If you want to get the complete image map for all the layers, use the [getHTMLImageMap](#) method of the BaseChart object instead.

This method should be called only after you have completely drawn the chart using the [makeChart](#), [makeChart2](#) or [makeChart3](#) method. It is because ChartDirector needs to actually draw the chart to compute the coordinates of the data points on the layer.

#### Arguments

Argument	Default Value	Description
dataItem	(Mandatory)	The position of the data point within the data sets. The first data point is 0, while the n <sup>th</sup> data point is (n-1).

#### Return Value

Return a text string representing the coordinates of the data points as drawn on the layer in HTML image map attribute format.

## getHTMLImageMap

PHP/Perl/Python Usage

getHTMLImageMap(url [, queryFormat [, extraAttr]])

#### C++ Prototype

```
virtual const char *getHTMLImageMap(const char *url, const char *queryFormat = 0, const char *extraAttr = 0) = 0;
```

#### Description

Generate an HTML image map for all data points as drawn on the layer.

If you want to generate an HTML image map for all the layers, use the [getHTMLImageMap](#) method of the BaseChart object instead. If you want to get the image map coordinates for a particular data point, use the [getImageCoor](#) or [getImageCoor2](#) method of the Layer object instead.

This method should be called only after you have completely drawn the chart using the [makeChart](#), [makeChart2](#) or [makeChart3](#) method. It is because ChartDirector needs to actually draw the chart to compute the coordinates of the various entities in the image map.

The description for this method is the same as the [getHTMLImageMap](#) method of the BaseChart object. Please refer to the [getHTMLImageMap](#) method of the BaseChart object for detail information regarding this method.

#### Arguments

Argument	Default Value	Description
url	(Mandatory)	The URL to be included as the “href” attribute of the image map. ChartDirector will append additional parameters to the URL to indicate which data point the user has clicked. If you do not use any URL, you may simply use an empty string.
queryFormat	""	A text string to specify the format of the parameters to be appended to the URL. If this argument is an empty string, a default queryFormat will be used. Please refer to the description above for details.
extraAttr	""	A text string to specify additional attributes to add to the <area> tag. You can include queryFormat parameters to provide information regarding the data point represented by the <area> tag. Please refer to the description above for details.

#### Return Value

Return a text string containing the image map generated.

## BarLayer

The BarLayer class, as its name implies, represents bar chart layers. The BarLayer is a subclass of the [Layer](#) class. The BarLayer is created by using the [addBarLayer](#), [addBarLayer2](#) and [addBarLayer3](#) methods of the XYChart object.

Method	Description
<a href="#"><u>setBarGap</u></a>	Set the gap between the bars in a bar chart layer.
<b>Methods inherited from Layer</b>	
<a href="#"><u>set3D</u></a>	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
<a href="#"><u>setLineWidth</u></a>	Set the default line width of data lines when drawing data sets on the layer.
<a href="#"><u>setBorderColor</u></a>	Set the default border color and border effect for drawing all the data sets on the layer.
<a href="#"><u>setDataCombineMethod</u></a>	Set the method used to combine multiple data sets in a layer.
<a href="#"><u>setLegend</u></a>	Set the order of the data names when entered into the legend box.
<a href="#"><u>addDataSet</u></a>	Add a data set to the chart layer.
<a href="#"><u>getDataSet</u></a>	Get the requested DataSet object.
<a href="#"><u>setXData</u></a>	Set the x values of the data points in the data sets.
<a href="#"><u>setXData2</u></a>	Set the x values of the data points in the data sets as evenly distributed in a range.
<a href="#"><u>getXCoor</u></a>	Get the x coordinate of a point given the x value.
<a href="#"><u>getYCoor</u></a>	Get the y coordinate of a point given the y value.
<a href="#"><u>setDataLabelFormat</u></a>	Set the data label format of the all data labels for all data sets in the layer.
<a href="#"><u>setDataLabelStyle</u></a>	Set the style used to draw data labels for all data sets in the layer.
<a href="#"><u>addCustomDataLabel</u></a>	Add a custom data label to a data point.
<a href="#"><u>setAggregateLabelFormat</u></a>	Set the data label format of the aggregate data labels.
<a href="#"><u>setAggregateLabelStyle</u></a>	Enable and set the style used to draw aggregate data labels in the layer.
<a href="#"><u>addCustomAggregateLabel</u></a>	Add a custom aggregate data label to an aggregated data point.
<a href="#"><u>getImageCoor</u></a>	Get the image map coordinates of a data point as drawn on the layer as HTML image map attributes.
<a href="#"><u>getImageCoor2</u></a>	Get the image map coordinates of all data points at the same x-axis position as drawn on the layer as HTML image map attributes.
<a href="#"><u>getHTMLImageMap</u></a>	Generate an HTML image map for all data points as drawn on the layer.

## setBarGap

PHP/Perl/Python Usage

setBarGap(barGap [, subBarGap])

C++ Prototype

virtual void setBarGap(double barGap, double subBarGap = 0.2) = 0;

Description

Set the gap between the bars in a bar chart layer.

Arguments

Argument	Default Value	Description
barGap	(Mandatory)	The gap between the bars as the portion of the space between the midpoints of the bars. The barGap typically is in the range 0 – 1. A value of 0 means the bars are tightly packed together with no gap in between. If a negative bar gap is used, the bars may overlap each others.  Note that for multi-bar charts, the barGap means the distance between the bar groups, not the distance between individual bars.
subBarGap	0.2	This argument only applies to multi-bar charts. This is the gap between bars within a bar group, represented as the portion of the space between the midpoints of the bars. The subBarGap typically is in the range 0 – 1. A value of 0 means the bars within a bar group are tightly packed together with no gap in between. If a negative subBarGap is used, the bars within a bar group may overlap each others.

Return Value

None

## LineLayer

The LineLayer class, as its name implies, represents line chart layers. The LineLayer is a subclass of the [Layer](#) class. The LineLayer is created by using the [addLineLayer](#) and [addLineLayer2](#) methods of the XYChart object.

The LineLayer has no additional method other than implementing methods inherited from the Layer class.

Method	Description
--------	-------------

<u>setGapColor</u>	Set the color and style of the line used for jumping across NoValue data points.
<u>setImageMapWidth</u>	Set the effective width of the line used in creating the image map for the line segments that joined the data points.
<b>Methods inherited from Layer</b>	
<u>set3D</u>	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
<u>setLineWidth</u>	Set the default line width of data lines when drawing data sets on the layer.
<u>setBorderColor</u>	Set the default border color and border effect for drawing all the data sets on the layer.
<u>setDataCombineMethod</u>	Set the method used to combine multiple data sets in a layer.
<u>setLegend</u>	Set the order of the data names when entered into the legend box.
<u>addDataSet</u>	Add a data set to the chart layer.
<u>getDataSet</u>	Get the requested DataSet object.
<u>setXData</u>	Set the x values of the data points in the data sets.
<u>setXData2</u>	Set the x values of the data points in the data sets as evenly distributed in a range.
<u>getXCoor</u>	Get the x coordinate of a point given the x value.
<u>getYCoor</u>	Get the y coordinate of a point given the y value.
<u>setDataLabelFormat</u>	Set the data label format of the all data labels for all data sets in the layer.
<u>setDataLabelStyle</u>	Set the style used to draw data labels for all data sets in the layer.
<u>addCustomDataLabel</u>	Add a custom data label to a data point.
<u>setAggregateLabelFormat</u>	Set the data label format of the aggregate data labels.
<u>setAggregateLabelStyle</u>	Enable and set the style used to draw aggregate data labels in the layer.
<u>addCustomAggregateLabel</u>	Add a custom aggregate data label to an aggregated data point.
<u>getImageCoor</u>	Get the image map coordinates of a data point as drawn on the layer as HTML image map attributes.
<u>getImageCoor2</u>	Get the image map coordinates of all data points at the same x-axis position as drawn on the layer as HTML image map attributes.
<u>getHTMLImageMap</u>	Generate an HTML image map for all data points as drawn on the layer.

## setGapColor

PHP/Perl/Python Usage

setGapColor(lineColor [, lineWidth ])

C++ Prototype

virtual void setGapColor(int lineColor, int lineWidth = -1) = 0;

Description

Set the color and style of the line used for jumping across [NoValue](#) data points.

By default, the lineColor is Transparent, which means NoValue data points will result in a discontinuous line.

Arguments

Argument	Default Value	Description
lineColor	(Mandatory)	The line color of the line used for jumping across NoValue data points
lineWidth	-1	The line width of the line used for jumping across NoValue data points. A value of -1 means the width will be the same as the line width of the line joining the data points.

Return Value

None

## setImageMapWidth

PHP/Perl/Python Usage

setImageMapWidth(width)

C++ Prototype

virtual void setImageMapWidth(int width) = 0;

Description

Set the effective width of the line used for image maps.

For thin lines, it is hard to click on the lines. So in generate an image map for a line chart, ChartDirector can act as if the line is a very thick line. The default is 10 pixels.

Arguments

Argument	Default Value	Description
width	(Mandatory)	The effective width of the line used for generating image maps.



Return Value  
None

## ScatterLayer

The ScatterLayer class, as its name implies, represents scatter chart layers. The ScatterLayer is a subclass of the [Layer](#) class. The ScatterLayer is created by using the [addScatterLayer](#) method of the XYChart object.

A scatter chart can be considered as a special kind of line chart, in which both x and y values are used to plot the data points, and in which the data symbols are enabled and the line width is set to zero. Therefore only the data symbols are visible.

ChartDirector supports setting the line width to non-zero using the [setLineWidth](#) method of the Layer object. In this case, there will be lines joining the data points.

Method	Description
<a href="#">setGapColor</a>	Set the color and style of the line used for jumping across NoValue data points.
<a href="#">setImageMapWidth</a>	Set the effective width of the line used in creating the image map for the line segments that joined the data points.
<b>Methods inherited from Layer</b>	
<a href="#">set3D</a>	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
<a href="#">setLineWidth</a>	Set the default line width of data lines when drawing data sets on the layer.
<a href="#">setBorderColor</a>	Set the default border color and border effect for drawing all the data sets on the layer.
<a href="#">setDataCombineMethod</a>	Set the method used to combine multiple data sets in a layer.
<a href="#">setLegend</a>	Set the order of the data names when entered into the legend box.
<a href="#">addDataSet</a>	Add a data set to the chart layer.
<a href="#">getDataSet</a>	Get the requested DataSet object.
<a href="#">setXData</a>	Set the x values of the data points in the data sets.
<a href="#">setXData2</a>	Set the x values of the data points in the data sets as evenly distributed in a range.
<a href="#">getXCoor</a>	Get the x coordinate of a point given the x value.
<a href="#">getYCoor</a>	Get the y coordinate of a point given the y value.
<a href="#">setDataLabelFormat</a>	Set the data label format of the all data labels for all data sets in the layer.

<a href="#"><u>setDataLabelStyle</u></a>	Set the style used to draw data labels for all data sets in the layer.
<a href="#"><u>addCustomDataLabel</u></a>	Add a custom data label to a data point.
<a href="#"><u>setAggregateLabelFormat</u></a>	Set the data label format of the aggregate data labels.
<a href="#"><u>setAggregateLabelStyle</u></a>	Enable and set the style used to draw aggregate data labels in the layer.
<a href="#"><u>addCustomAggregateLabel</u></a>	Add a custom aggregate data label to an aggregated data point.
<a href="#"><u>getImageCoor</u></a>	Get the image map coordinates of a data point as drawn on the layer as HTML image map attributes.
<a href="#"><u>getImageCoor2</u></a>	Get the image map coordinates of all data points at the same x-axis position as drawn on the layer as HTML image map attributes.
<a href="#"><u>getHTMLImageMap</u></a>	Generate an HTML image map for all data points as drawn on the layer.

## setGapColor

PHP/Perl/Python Usage

setGapColor(lineColor [, lineWidth ])

C++ Prototype

virtual void setGapColor(int lineColor, int lineWidth = -1) = 0;

Description

Set the color and style of the line used for jumping across [NoValue](#) data points.

By default, the lineColor is Transparent, which means NoValue data points will result in a discontinuous line.

Arguments

Argument	Default Value	Description
lineColor	(Mandatory)	The line color of the line used for jumping across NoValue data points
lineWidth	-1	The line width of the line used for jumping across NoValue data points. A value of -1 means the width will be the same as the line width of the line joining the data points.

Return Value

None

## setImageMapWidth

PHP/Perl/Python Usage

setImageMapWidth(width)

C++ Prototype

```
virtual void setImageMapWidth(int width) = 0;
```

Description

Set the effective width of the line used in creating the image map for the line segments that joined the data points.

By default, in a scatter chart layer, there is no line segment joining the data points. So the image map will only contain the data symbols.

However, if the scatter chart layer is configured to include lines for joining the data points (using the [setLineWidth](#) method of the Layer object), the image map can also contain the lines.

For thin lines, it is hard to click on the lines. So in generate an image map for a scatter chart, ChartDirector can act as if the line is a very thick line. The default is 0 pixel in a scatter chart, which means no image map will be created for the lines.

Arguments

Argument	Default Value	Description
width	(Mandatory)	The effective width of the line used for generating image maps.

Return Value

None

## TrendLayer

The TrendLayer class, as its name implies, represents trend chart layers. The TrendLayer is a subclass of the [Layer](#) class. The TrendLayer is created by using the [addTrendLayer](#) or [addTrendLayer2](#) method of the XYChart object.

The trend chart layer will draw a best fit straight line for all the data in the data sets contained within the trend layer. The best fit straight line will be calculated using the least square method.

Note that the trend chart layer will not draw the data points itself. It will only draw the trend line. To draw both the data points and the trend line, use two separate layers. For example, you can use a line chart layer (or scatter chart layer) to draw the data points, and a trend chart layer to draw the trend line.

Each trend layer only draws one single trend line that fits all the data sets it contains. If you have multiple data sets, and you want to draw a separate trend line for each data set, use multiple trend chart layers, with each layer containing one data set.

Method	Description
<a href="#">setImageMapWidth</a>	Set the effective width of the line used in creating the image maps.
Methods inherited from Layer	

<a href="#"><u>set3D</u></a>	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
<a href="#"><u>setLineWidth</u></a>	Set the default line width of data lines when drawing data sets on the layer.
<a href="#"><u>setBorderColor</u></a>	Set the default border color and border effect for drawing all the data sets on the layer.
<a href="#"><u>setDataCombineMethod</u></a>	Set the method used to combine multiple data sets in a layer.
<a href="#"><u>setLegend</u></a>	Set the order of the data names when entered into the legend box.
<a href="#"><u>addDataSet</u></a>	Add a data set to the chart layer.
<a href="#"><u>getDataSet</u></a>	Get the requested DataSet object.
<a href="#"><u>setXData</u></a>	Set the x values of the data points in the data sets.
<a href="#"><u>setXData2</u></a>	Set the x values of the data points in the data sets as evenly distributed in a range.
<a href="#"><u>getXCoor</u></a>	Get the x coordinate of a point given the x value.
<a href="#"><u>getYCoor</u></a>	Get the y coordinate of a point given the y value.
<a href="#"><u>setDataLabelFormat</u></a>	Set the data label format of the all data labels for all data sets in the layer.
<a href="#"><u>setDataLabelStyle</u></a>	Set the style used to draw data labels for all data sets in the layer.
<a href="#"><u>addCustomDataLabel</u></a>	Add a custom data label to a data point.
<a href="#"><u>setAggregateLabelFormat</u></a>	Set the data label format of the aggregate data labels.
<a href="#"><u>setAggregateLabelStyle</u></a>	Enable and set the style used to draw aggregate data labels in the layer.
<a href="#"><u>addCustomAggregateLabel</u></a>	Add a custom aggregate data label to an aggregated data point.
<a href="#"><u>getImageCoor</u></a>	Get the image map coordinates of a data point as drawn on the layer as HTML image map attributes.
<a href="#"><u>getImageCoor2</u></a>	Get the image map coordinates of all data points at the same x-axis position as drawn on the layer as HTML image map attributes.
<a href="#"><u>getHTMLImageMap</u></a>	Generate an HTML image map for all data points as drawn on the layer.

## setImageMapWidth

PHP/Perl/Python Usage  
setImageMapWidth(width)

C++ Prototype

virtual void setImageMapWidth(int width) = 0;

#### Description

Set the effective width of the line used in creating the image maps.

For thin lines, it is hard to click on the lines. So in generate an image map for a trend line, ChartDirector can act as if the line is a very thick line. The default is 10 pixel in a trend chart layer.

#### Arguments

Argument	Default Value	Description
width	(Mandatory)	The effective width of the line used for generating image maps.

#### Return Value

None

## AreaLayer

The AreaLayer class, as its name implies, represents area chart layers. The AreaLayer is a subclass of the [Layer](#) class. The AreaLayer is created by using the [addAreaLayer](#) and [addAreaLayer2](#) methods of the XYChart object.

The AreaLayer has no additional method other than implementing methods inherited from the Layer class.

Method	Description
<b>Methods inherited from Layer</b>	
<a href="#">set3D</a>	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
<a href="#">setLineWidth</a>	Set the default line width of data lines when drawing data sets on the layer.
<a href="#">setBorderColor</a>	Set the default border color and border effect for drawing all the data sets on the layer.
<a href="#">setDataCombineMethod</a>	Set the method used to combine multiple data sets in a layer.
<a href="#">setLegend</a>	Set the order of the data names when entered into the legend box.
<a href="#">addDataSet</a>	Add a data set to the chart layer.
<a href="#">getDataSet</a>	Get the requested DataSet object.
<a href="#">setXData</a>	Set the x values of the data points in the data sets.
<a href="#">setXData2</a>	Set the x values of the data points in the data sets as evenly distributed in a range.
<a href="#">getXCoord</a>	Get the x coordinate of a point given the x value.
<a href="#">getYCoord</a>	Get the y coordinate of a point given the y value.

<a href="#"><u>setDataLabelFormat</u></a>	Set the data label format of the all data labels for all data sets in the layer.
<a href="#"><u>setDataLabelStyle</u></a>	Set the style used to draw data labels for all data sets in the layer.
<a href="#"><u>addCustomDataLabel</u></a>	Add a custom data label to a data point.
<a href="#"><u>setAggregateLabelFormat</u></a>	Set the data label format of the aggregate data labels.
<a href="#"><u>setAggregateLabelStyle</u></a>	Enable and set the style used to draw aggregate data labels in the layer.
<a href="#"><u>addCustomAggregateLabel</u></a>	Add a custom aggregate data label to an aggregated data point.
<a href="#"><u>getImageCoor</u></a>	Get the image map coordinates of a data point as drawn on the layer as HTML image map attributes.
<a href="#"><u>getImageCoor2</u></a>	Get the image map coordinates of all data points at the same x-axis position as drawn on the layer as HTML image map attributes.
<a href="#"><u>getHTMLImageMap</u></a>	Generate an HTML image map for all data points as drawn on the layer.

## HLOCLayer

The HLOCLayer class represents high-low-open-close chart layers. The HLOCLayer is a subclass of the [Layer](#) class. The HLOCLayer is created by using the [addHLOCLayer](#) and [addHLOCLayer2](#) methods of the XYChart object.

The HLOCLayer has no additional method other than implementing methods inherited from the Layer class.

Method	Description
<a href="#"><u>setDataGap</u></a>	Set the gap between two adjacent HLOC elements.
<b>Methods inherited from Layer</b>	
<a href="#"><u>set3D</u></a>	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
<a href="#"><u>setLineWidth</u></a>	Set the default line width of data lines when drawing data sets on the layer.
<a href="#"><u>setBorderColor</u></a>	Set the default border color and border effect for drawing all the data sets on the layer.
<a href="#"><u>setDataCombineMethod</u></a>	Set the method used to combine multiple data sets in a layer.
<a href="#"><u>setLegend</u></a>	Set the order of the data names when entered into the legend box.
<a href="#"><u>addDataSet</u></a>	Add a data set to the chart layer.
<a href="#"><u>getDataSet</u></a>	Get the requested DataSet object.
<a href="#"><u>setXData</u></a>	Set the x values of the data points in the data sets.

<a href="#"><u>setXData2</u></a>	Set the x values of the data points in the data sets as evenly distributed in a range.
<a href="#"><u>getXCoord</u></a>	Get the x coordinate of a point given the x value.
<a href="#"><u>getYCoord</u></a>	Get the y coordinate of a point given the y value.
<a href="#"><u>setDataLabelFormat</u></a>	Set the data label format of the all data labels for all data sets in the layer.
<a href="#"><u>setDataLabelStyle</u></a>	Set the style used to draw data labels for all data sets in the layer.
<a href="#"><u>addCustomDataLabel</u></a>	Add a custom data label to a data point.
<a href="#"><u>setAggregateLabelFormat</u></a>	Set the data label format of the aggregate data labels.
<a href="#"><u>setAggregateLabelStyle</u></a>	Enable and set the style used to draw aggregate data labels in the layer.
<a href="#"><u>addCustomAggregateLabel</u></a>	Add a custom aggregate data label to an aggregated data point.
<a href="#"><u>getImageCoord</u></a>	Get the image map coordinates of a data point as drawn on the layer as HTML image map attributes.
<a href="#"><u>getImageCoord2</u></a>	Get the image map coordinates of all data points at the same x-axis position as drawn on the layer as HTML image map attributes.
<a href="#"><u>getHTMLImageMap</u></a>	Generate an HTML image map for all data points as drawn on the layer.

## setDataGap

PHP/Perl/Python Usage  
setDataGap(gap)

C++ Prototype  
virtual void setDataGap(double gap) = 0;

Description  
Set the gap between two adjacent HLOC elements.

Arguments

Argument	Default Value	Description
gap	(Mandatory)	The gap between two adjacent HLOC elements as the portion of the space between the midpoints of the elements. The gap must be in the range 0 – 1. A value of 0 (the default) means there is no gap between two adjacent HLOC elements.

Return Value  
None

# CandleStickLayer

The CandleStickLayer class represents candlestick chart layers. The CandleStickLayer is a subclass of the [Layer](#) class. The CandleStickLayer is created by using the [addCandleStickLayer](#) method of the XYChart object.

Method	Description
<a href="#">setDataGap</a>	Set the gap between two adjacent candles.
<b>Methods inherited from Layer</b>	
<a href="#">set3D</a>	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
<a href="#">setLineWidth</a>	Set the default line width of data lines when drawing data sets on the layer.
<a href="#">setBorderColor</a>	Set the default border color and border effect for drawing all the data sets on the layer.
<a href="#">setDataCombineMethod</a>	Set the method used to combine multiple data sets in a layer.
<a href="#">setLegend</a>	Set the order of the data names when entered into the legend box.
<a href="#">addDataSet</a>	Add a data set to the chart layer.
<a href="#">getDataSet</a>	Get the requested DataSet object.
<a href="#">setXData</a>	Set the x values of the data points in the data sets.
<a href="#">setXData2</a>	Set the x values of the data points in the data sets as evenly distributed in a range.
<a href="#">getXCoord</a>	Get the x coordinate of a point given the x value.
<a href="#">getYCoord</a>	Get the y coordinate of a point given the y value.
<a href="#">setDataLabelFormat</a>	Set the data label format of the all data labels for all data sets in the layer.
<a href="#">setDataLabelStyle</a>	Set the style used to draw data labels for all data sets in the layer.
<a href="#">addCustomDataLabel</a>	Add a custom data label to a data point.
<a href="#">setAggregateLabelFormat</a>	Set the data label format of the aggregate data labels.
<a href="#">setAggregateLabelStyle</a>	Enable and set the style used to draw aggregate data labels in the layer.
<a href="#">addCustomAggregateLabel</a>	Add a custom aggregate data label to an aggregated data point.
<a href="#">getImageCoord</a>	Get the image map coordinates of a data point as drawn on the layer as HTML image map attributes.
<a href="#">getImageCoord2</a>	Get the image map coordinates of all data points at the same x-axis position as drawn on the layer as HTML image map attributes.



<a href="#"><u>getHTMLImageMap</u></a>	Generate an HTML image map for all data points as drawn on the layer.
--	---

## setDataGap

PHP/Perl/Python Usage

setDataGap(gap)

C++ Prototype

virtual void setDataGap(double gap) = 0;

Description

Set the gap between two adjacent candles.

Arguments

Argument	Default Value	Description
gap	(Mandatory)	The gap between two adjacent candles as the portion of the space between the midpoints of the candles. The gap must be in the range 0 – 1. A value of 0 means there is no gap between two adjacent candles. The default value is 0.2.

Return Value

None

## DataSet

The DataSet class represents data sets. It is created by using the [addDataSet](#) method of the Layer class.

Method	Description
<a href="#"><u>setDataName</u></a>	Set the name of the data set.
<a href="#"><u>setDataColor</u></a>	Set the colors used to draw the data set.
<a href="#"><u>setUseYAxis2</u></a>	Determine whether the primary y-axis or secondary y-axis to use when drawing the data set on the chart.
<a href="#"><u>setLineWidth</u></a>	Set the width of data lines when drawing the data set on the layer.
<a href="#"><u>setDataSymbol</u></a>	Use one of the built-in symbols as the graphics symbol to plot the data point on the charts.
<a href="#"><u>setDataSymbol2</u></a>	Load an image from a file and use it as the graphics symbol to plot the data point on the charts.
<a href="#"><u>setDataSymbol3</u></a>	Use a DrawArea object as the graphics symbol to plot the data point on the charts.

<a href="#"><u>setDataLabelFormat</u></a>	Set the data label format of the data labels of the data set.
<a href="#"><u>setDataLabelStyle</u></a>	Set the style used to draw data labels for the data set.

## setDataName

PHP/Perl/Python Usage

setDataName(name)

C++ Prototype

```
virtual void setDataName(const char *name) = 0;
```

Description

Set the name of the data set. The name will be used in the legend box, if one is available for the chart. If the name is not set, there will be no legend entry for the data set, even if a legend box is available on the chart.

Arguments

Argument	Default Value	Description
name	(Mandatory)	The name of the data set. The name will be used in the legend box, if one is available.

Return Value

None.

## setDataColor

PHP/Perl/Python Usage

setDataColor(dataColor [, edgeColor [, shadowColor [, shadowEdgeColor]]])

C++ Prototype

```
virtual void setDataColor(int dataColor, int edgeColor = -1, int shadowColor = -1, int shadowEdgeColor = -1) = 0;
```

Description

Set the colors used to draw the data set.

Arguments

Argument	Default Value	Description
dataColor	(Mandatory)	The main color used to draw the data set. For a bar layer, this would be the color of the bar. For a line layer, this would be the color of the line. For an area layer, this would be the color of the area. For a HLOC layer, this would be the color of the HLOC line.

edgeColor	-1	<p>The color used to draw the edges for the data set, if the layer type has edges. For a bar layer, the edges mean the edges of the bar. For an area layer, the edges mean the edges of the area. Line layers and HLOC layers do not have edges for the data points, so this parameter is not applicable.</p> <p>The default value of -1 means that the edges are drawn using the default border color of the layer (defined using the <a href="#">setBorderColor</a> method of the Layer object).</p>
shadowColor	-1	<p>The color to use to draw shadows in 3D. This parameter is only applicable for 3D layers. The default value of -1 means the shadow color will be a “darker” version of the data color. The ChartDirector computes the “darker” color by reducing the RGB components of the data color in half.</p>
shadowEdgeColor	-1	<p>The color to use to draw edges of the shadows in 3D. This parameter is only applicable for 3D layers. The default value of -1 means the shadow color will be a “darker” version of the edge color. The ChartDirector computes the “darker” color by reducing the RGB components of the edge color in half.</p>

Return Value  
None.

## setUseYAxis2

PHP/Perl/Python Usage  
setUseYAxis2([b])

C++ Prototype  
virtual void setUseYAxis2(bool b = true) = 0;

Description  
Determine whether the primary y-axis or secondary y-axis to use when drawing the data set on the chart. If this method is never called, the primary y-axis will be used.

Arguments

Argument	Default Value	Description
b	1	A “true” (non-zero) value means the secondary y-axis will be used. A “false” value means the primary y-axis will be used.

Return Value  
None.

## setLineWidth

PHP/Perl/Python Usage

setLineWidth(w)

C++ Prototype

```
virtual void setLineWidth(int w) = 0;
```

Description

Set the width of data lines when drawing the data set on the layer. This only applies to layers that employ lines to represent data.

If this method is not called, the line width will be the default line width for the layer that contains the data set. The default line width of a layer is set using the [setLineWidth](#) method of the layer object.

Arguments

Argument	Default Value	Description
w	(Mandatory)	The width of the line in pixels.

Return Value

None.

## setDataSymbol

PHP/Perl/Python Usage

setDataSymbol(symbol [, size [, fillColor [, edgeColor]]])

C++ Prototype

```
virtual void setDataSymbol(SymbolType symbol, int size = 5, int fillColor = -1, int edgeColor = -1) = 0;
```





Description















Use one of the built-in symbols as the graphics symbol to plot the data points on the charts.

In the current version of ChartDirector, this symbol is only used to for drawing line chart layers.

If you want to load a symbol image from a file instead of using the built-in symbols, use the [setDataSymbol2](#) method. If you want to use a symbol image that you create “on the fly” using the [DrawArea](#) object, use the [setDataSymbol3](#) method.

The built-in symbols are specified by using the following constants as the “symbol” argument.

Constant	Symbol at 15 pixels	Symbol at 5 pixels (default size)
SquareSymbol		
DiamondSymbol		

TriangleSymbol		
RightTriangleSymbol		
LeftTriangleSymbol		
InvertedTriangleSymbol		
CircleSymbol		
CrossSymbol		
Cross2Symbol		

#### Arguments

Argument	Default Value	Description
symbol	(Mandatory)	One of the pre-defined symbol constants (see description above) to specify the symbol to use.
size	5	The width and height of the symbol in pixels. The default symbol size is 5 pixels. It is recommended to use an odd number for the symbol size, as many symbols will draw better if the center point has integer coordinates.
fillColor	-1	The color used to fill the symbol. The default value of -1 means the color of the data set will be used (set using the <a href="#">setDataColor</a> method).
edgeColor	-1	The edge color used to draw the edge of the symbol. The default value of -1 means the edge color of the data set will be used (set using the <a href="#">setDataColor</a> method).

Return Value

None

## setDataSymbol2

PHP/Perl/Python Usage

setDataSymbol2(filename)

C++ Prototype

```
virtual void setDataSymbol(const char *image) = 0;
```

Description

Load an image from a file and use it as the graphics symbol to plot the data point on the charts.

In the current version of ChartDirector, this symbol is only used to for drawing line chart layers.

If you want to use the ChartDirector built-in symbols, use the [setDataSymbol](#) method. If you want to use a symbol image that you create “on the fly” using the DrawArea object, use the [setDataSymbol3](#) method.

#### Arguments

Argument	Default Value	Description
image	(Mandatory)	The filename of the image file. The extensions png, jpg/jpeg, gif and wbmp/wmp (case insensitive) represent PNG, JPEG, GIF and WAP bitmap respectively.

#### Return Value

None.

## setDataSymbol3

PHP/Perl/Python Usage

setDataSymbol3(obj)

#### C++ Prototype

```
virtual void setDataSymbol(const DrawArea *image) = 0;
```

#### Description

Use a [DrawArea](#) object as the graphics symbol to plot the data point on the charts.

In the current version of ChartDirector, this symbol is only used to for drawing line chart layers.

If you want to use the ChartDirector built-in symbols, use the [setDataSymbol](#) method. If you want to load a symbol image from a file instead of using the built-in symbols, use the [setDataSymbol2](#) method.

#### Arguments

Argument	Default Value	Description
obj	(Mandatory)	The DrawArea object to be used as the symbol.

#### Return Value

None.

## setDataLabelFormat

PHP/Perl/Python Usage

setDataLabelFormat(formatString)

#### C++ Prototype

```
virtual void setDataLabelFormat(const char *formatString) = 0;
```

#### Description

Set the data label format of the data labels of the data set. If you want to set the label format for all data sets in a particular chart layer, use the [setDataLabelFormat](#) method of the Layer object.

Data labels are labels that appear besides the data points in the chart. For the exact positions where data labels will be drawn, please refer to the description of the [setDataLabelStyle](#) method of the Layer object.

By default, the data label format is “{value}”, which means the label will contain the data value. You can include more information and control the formatting of the data label by supplying an appropriate format string. Please refer to the section on [Data and Text Formatting](#) for details.

#### Arguments

Argument	Default Value	Description
formatString	(Mandatory)	The format string. See above for description.

#### Return Value

None

## setDataLabelStyle

PHP/Perl/Python Usage

```
setDataLabelStyle([font [, fontHeight [, fontColor [, fontAngle]]]])
```

#### C++ Prototype

```
virtual TextStyle *setDataLabelStyle(const char *font = 0, double fontHeight = 8, int fontColor = TextColor, double fontAngle = 0) = 0;
```

#### Description

Set the style used to draw data labels for the data set. If you want to set the style for all data sets within a particular chart layer, use the [setDataLabelStyle](#) method of the Layer object.

Data labels are labels that appear besides the data points in the chart. For the exact positions where data labels will be drawn, please refer to the description of the [setDataLabelStyle](#) method of the Layer object.

When using data labels with y-axis auto-scaling, it is recommended that the [setAutoScale](#) or [setTopMargin](#) method of the YAxis object be used to reserve some margin at the top of the plot area. This ensures the data labels are always drawn within the plot area.

For details about how to specify font style, please refer to the section on [Font Specification](#).

#### Arguments

Argument	Default Value	Description
font	""	The font used to draw the labels. A "" (empty string) means using the default font (Arial).
fontSize	8	The font size used to draw the labels.
fontColor	<a href="#">TextColor</a>	The color used to draw the labels.
fontAngle	0	Set the rotation of the font.

Return Value

A [TextBox](#) object that represents the Usage of the data labels. You may use the methods of the TextBox object to fine-tune the appearance of the data labels.

## DrawArea

The DrawArea class is the graphics toolkit that ChartDirector employs to draw the charts. Each [BaseChart](#) class contains a DrawArea object. This DrawArea object is accessible via the [getDrawArea](#) method, so that you could draw custom lines, shapes or texts things on the charts.

You could also use the DrawArea class in standalone mode to create images.

Method	Description
<a href="#">DrawArea Constructor</a>	Create a DrawArea object.
<a href="#">destroy</a>	Destroy the DrawArea object. (Used only in ChartDirector for C++.)
<a href="#">setSize</a>	Set the size and background color of the image.
<a href="#">getWidth</a>	Get the width of the image.
<a href="#">getHeight</a>	Get the height of the image.
<a href="#">setBgColor</a>	Set the background color of the image.
<a href="#">pixel</a>	Apply the specified color to a pixel.
<a href="#">getPixel</a>	Get the color of a pixel.
<a href="#">line</a>	Draw a straight line.
<a href="#">hline</a>	Draw a horizontal line.
<a href="#">vline</a>	Draw a vertical line.
<a href="#">arc</a>	Draw an arc.
<a href="#">rect</a>	Draw a rectangle.
<a href="#">polygon</a>	Draw a polygon.
<a href="#">surface</a>	Draw a 3D surface.
<a href="#">sector</a>	Draw a sector.
<a href="#">cylinder</a>	Draw a cylinder surface.
<a href="#">circle</a>	Draw a circle or an ellipse.
<a href="#">fill</a>	Fill an area using the specified color, where the area is bounded by a given border color.
<a href="#">fill2</a>	Fill an area using the specified color, where the area is defined as a continuous region having the same color.
<a href="#">text</a>	Draw text on the image. This method is exactly the same as the text2 method except that it is simplified to contain less arguments.



<u>text2</u>	Draw text on the image.
<u>text3</u>	Create a <u>TTFTText</u> object that represents the text to be drawn. This method is exactly the same as the <u>text4</u> method except that it is simplified to contain less arguments.
<u>text4</u>	Create a <u>TTFTText</u> object that represents the text to be drawn.
<u>close</u>	Destroy the TTFTText object created using <u>text3</u> or <u>text4</u> . (Used only in ChartDirector for C++.)
<u>merge</u>	Apply another DrawArea image on top of the current DrawArea image.
<u>tile</u>	Apply another DrawArea image on top of the current DrawArea image as a wallpaper.
<u>load</u>	Load the specified image into the current DrawArea. This method will determine the image type by using the extension of the filename.
<u>loadGIF</u>	Load the specified GIF image into the current DrawArea.
<u>loadPNG</u>	Load the specified PNG image into the current DrawArea.
<u>loadJPG</u>	Load the specified JPEG image into the current DrawArea.
<u>loadWMP</u>	Load the specified WAP bitmap image into the current DrawArea.
<u>out</u>	Write the current DrawArea image to an image file. This method will determine the image type by using the extension of the filename.
<u>outGIF</u>	Write the current DrawArea image to an alternative GIF image file.
<u>outGIF2</u>	Write the current DrawArea image as an alternative GIF image in memory.
<u>outPNG</u>	Write the current DrawArea image to a PNG image file.
<u>outPNG2</u>	Write the current DrawArea image as a PNG image in memory.
<u>outJPG</u>	Write the current DrawArea image to a JPEG image file.
<u>outJPG2</u>	Write the current DrawArea image as a JPG image in memory.
<u>outWMP</u>	Write the current DrawArea image to a WAP bitmap image file.
<u>outWMP2</u>	Write the current DrawArea image as a WAP bitmap image in memory.
<u>setPaletteMode</u>	Set the palette mode when writing the image to a PNG file.
<u>setDitherMethod</u>	Set the dithering method if dithering is required.
<u>setTransparentColor</u>	Set the transparent color for the image when writing the image to an image file.
<u>setAntiAliasText</u>	Set whether anti-alias methods are when drawing text.
<u>setInterlace</u>	Set the interlace mode when writing the image to an image file.
<u>setColorTable</u>	Change the colors of the palette color table.

<a href="#"><u>getARGBColor</u></a>	Change the given color to ARGB format if the given color is a palette table color.
<a href="#"><u>dashLineColor</u></a>	Create a dash line color that is suitable for drawing dash lines.
<a href="#"><u>patternColor</u></a>	Create a pattern color that can be used to fill areas with repeating patterns. The pattern is defined using an array of colors as the bitmap.
<a href="#"><u>patternColor2</u></a>	Create a pattern color that can be used to fill areas with repeating patterns. The pattern is defined by loading from an image file.
<a href="#"><u>gradientColor</u></a>	Create a gradient color that can be used to fill areas with color gradients. The gradient color is defined using a gradient line segment with a starting color and an ending color.
<a href="#"><u>gradientColor2</u></a>	Create a gradient color that can be used to fill areas with color gradients. The gradient color is defined using a gradient line segment with multiple color points.

## DrawArea Constructor

PHP/Perl Usage

`new DrawArea()`

Python Usage

`DrawArea()`

C++ Prototype

`static DrawArea* create();`

Description

Create a DrawArea object. This method is only needed if you use DrawArea in standalone mode. If you use DrawArea in ChartDirector charts, the DrawArea object is automatically created by the [BaseChart](#), and is accessible via the [getDrawArea](#) method.

Arguments

None.

Return Value

The DrawArea object created.

## destroy

C++ Prototype

`virtual void destroy() = 0;`

Description

This method is for the C++ version of ChartDirector only. It is not required in other versions of ChartDirector.

Destroy the DrawArea object. This method is only needed if you explicitly creates a DrawArea object using the DrawArea constructor. If you use DrawArea in ChartDirector charts, the DrawArea object is automatically created and destroyed by the [BaseChart](#).

Arguments

None.

Return Value

None.

## setSize

PHP/Perl/Python Usage

setSize(width, height [, bgColor])

C++ Prototype

virtual void setSize(int width, int height, int bgColor = 0xffffffff) = 0;

Description

Set the size and background color of the image.

Arguments

Argument	Default Value	Description
width	(Mandatory)	The width of the image in pixels.
height	(Mandatory)	The height of the image in pixels.
bgColor	0xffffffff	The background color of the image.

Return Value

None.

## getWidth

PHP/Perl/Python Usage

getWidth()

C++ Prototype

virtual int getWidth() const = 0;

Description

Get the width of the image.

Arguments

None.

Return Value

The width of the image in pixels.

## getHeight

PHP/Perl/Python Usage

getHeight()

C++ Prototype

```
virtual int getHeight() const = 0;
```

Description

Get the height of the image.

Arguments

None.

Return Value

The height of the image in pixels.

## setBgColor

PHP/Perl/Python Usage

setBgColor(c)

C++ Prototype

```
virtual void setBgColor(int c) = 0;
```

Description

Set the background color of the image.

Arguments

Argument	Default Value	Description
c	(Mandatory)	The background color of the image.

Return Value

None.

## pixel

PHP/Perl/Python Usage

pixel(x, y, c)

C++ Prototype

```
virtual void pixel(int x, int y, int c) = 0;
```

Description

Apply the specified color to a pixel.

Arguments

Argument	Default Value	Description
----------	---------------	-------------

x	(Mandatory)	The x coordinate of the pixel.
y	(Mandatory)	The y coordinate of the pixel.
c	(Mandatory)	The color to apply to the pixel.

Return Value  
None.

## getPixel

PHP/Perl/Python Usage

getPixel(x, y)

C++ Prototype

virtual int getPixel(int x, int y) const = 0;

Description

Get the color of a pixel.

Arguments

Argument	Default Value	Description
x	(Mandatory)	The x coordinate of the pixel.
y	(Mandatory)	The y coordinate of the pixel.

Return Value

The color of the pixel.

## line

PHP/Perl/Python Usage

line(x1, y1, x2, y2, c [, lineWidth])

C++ Prototype

virtual void line(int x1, int y1, int x2, int y2, int c, int lineWidth = 1) = 0;

Description

Draw a straight line.

Arguments

Argument	Default Value	Description
x1	(Mandatory)	The x coordinate of the first end-point of the line.
y1	(Mandatory)	The y coordinate of the first end-point of the line.
x2	(Mandatory)	The x coordinate of the second end-point of the line.
y2	(Mandatory)	The y coordinate of the second end-point of the line.

c	(Mandatory)	The color of the line.
lineWidth	1	The line width (thickness) of the line.

Return Value  
None.

## hline

PHP/Perl/Python Usage

hline(x1, x2, y, c)

C++ Prototype

virtual void hline(int x1, int x2, int y, int c) = 0;

Description

Draw a horizontal line. Although the [line](#) method can also be used to draw horizontal line, the hline method is more efficient.

Arguments

Argument	Default Value	Description
x1	(Mandatory)	The x coordinate of the first end-point of the line.
x2	(Mandatory)	The x coordinate of the second end-point of the line.
y	(Mandatory)	The y coordinate of the line.
c	(Mandatory)	The color of the line.

Return Value  
None.

## vline

PHP/Perl/Python Usage

vline(y1, y2, x, c)

C++ Prototype

virtual void vline(int y1, int y2, int x, int c) = 0;

Description

Draw a vertical line. Although the [line](#) method can also be used to draw vertical line, the vline method is more efficient.

Arguments

Argument	Default Value	Description
y1	(Mandatory)	The y coordinate of the first end-point of the line.
y2	(Mandatory)	The y coordinate of the second end-point of the line.

x	(Mandatory)	The x coordinate of the line.
c	(Mandatory)	The color of the line.

Return Value  
None.

## arc

PHP/Perl/Python Usage  
arc(cx, cy, rx, ry, a1, a2, c)

C++ Prototype  
virtual void arc(int cx, int cy, int rx, int ry, double a1, double a2, int c) = 0;

Description  
Draw an arc. This method supports independent vertical radius and horizontal radius, so both circular and elliptical arcs can be drawn.

### Arguments

Argument	Default Value	Description
cx	(Mandatory)	The x coordinate of the center of the circle or ellipse that contains the arc.
cy	(Mandatory)	The y coordinate of the center of the circle or ellipse that contains the arc.
rx	(Mandatory)	The horizontal radius of the circle or ellipse that contains the arc.
ry	(Mandatory)	The vertical radius of the circle or ellipse that contains the arc.
a1	(Mandatory)	The start angle of the arc in degrees. The angle is measured clockwise, with the y-axis as the 0 degree.
a2	(Mandatory)	The end angle of the arc in degrees. The angle is measured clockwise, with the y-axis as the 0 degree.
c	(Mandatory)	The color of the arc.

Return Value  
None.

## rect

PHP/Perl/Python Usage  
rect(x1, y1, x2, y2, edgeColor, fillColor [, raisedEffect])

C++ Prototype

```
virtual void rect(int x1, int y1, int x2, int y2, int edgeColor, int fillColor, int raisedEffect = 0) = 0;
```

Description

Draw a rectangle.

Arguments

Argument	Default Value	Description
x1	(Mandatory)	The x coordinate of one of the corner of the rectangle.
y1	(Mandatory)	The y coordinate of one of the corner of the rectangle.
x2	(Mandatory)	The x coordinate of the corner of the rectangle that is opposite to the corner as specified in (x1, y1).
y2	(Mandatory)	The y coordinate of the corner of the rectangle that is opposite to the corner as specified in (x1, y1).
edgeColor	(Mandatory)	The border color of the rectangle. If you do not want to draw a border for the rectangle, set the edgeColor the same as the fillColor.
fillColor	(Mandatory)	The color used to fill the rectangle. If you do not want to fill the rectangle, set the fillColor to Transparent.
raisedEffect	0	The width of the 3D border of the box. For positive values, the box will appear raised. For negative values, the box will appear depressed. A zero value means no 3D border should be drawn, that is, the box will appear as flat.

Return Value

None.

## **polygon**

PHP/Perl/Python Usage

`polygon(points, edgeColor, fillColor)`

C++ Prototype

```
virtual void polygon(IntArray x, IntArray y, int edgeColor, int fillColor) = 0;
```

Description

Draw a polygon.

Arguments

Argument	Default Value	Description
----------	---------------	-------------



(PHP/Perl/Python) points	(Mandatory)	A list of coordinates representing the vertices of a polygon. Each coordinate is a tuple (or list or array) containing two numbers for the x and y coordinates. <b><u>The argument is required for PHP/Perl/Python only.</u></b>
(C++ Only) x	(Mandatory)	An array containing the x coordinates of the vertices of the polygon. <b><u>The argument is required for C++ only.</u></b>
(C++ Only) y	(Mandatory)	An array containing the y coordinates of the vertices of the polygon. <b><u>The argument is required for C++ only.</u></b>
edgeColor	(Mandatory)	The border color of the polygon. If you do not want to draw a border for the polygon, set the edgeColor the same as the fillColor.
fillColor	(Mandatory)	The color used to fill the polygon. If you do not want to fill the polygon, set the fillColor to Transparent.

Return Value

None.

## surface

PHP/Perl/Python Usage

surface(x1, y1, x2, y2, depthX, depthY, edgeColor, fillColor)

C++ Prototype

virtual void surface(int x1, int y1, int x2, int y2, int depthX, int depthY, int edgeColor, int fillColor) = 0;

Description

Draw a 3D surface. A 3D surface can be imagined as a surface that is vertical to the drawing surface. The intersection between the two surfaces is a straight line. If the angle between the two surfaces is 90 degrees, the 3D surface will look like the straight line when projected onto the 3D surface. If the angle is something else, the 3D surface will look like a parallelogram when projected into a 2D image.

Since this method essentially draws a parallelogram to represent 3D surfaces on the image, it can also be used to draw parallelograms.

Arguments

Argument	Default Value	Description
x1	(Mandatory)	The x coordinate of the first end-point of the intersection line between the 3D surface and the drawing surface. (The line can be considered as one of the edge of the parallelogram.)

y1	(Mandatory)	The y coordinate of the first end-point of the intersection line between the 3D surface and the drawing surface. (The line can be considered as one of the edge of the parallelogram.)
x2	(Mandatory)	The x coordinate of the first end-point of the intersection line between the 3D surface and the drawing surface. (The line can be considered as one of the edge of the parallelogram.)
y2	(Mandatory)	The y coordinate of the first end-point of the intersection line between the 3D surface and the drawing surface. (The line can be considered as one of the edge of the parallelogram.)
depthX	(Mandatory)	The x component of the depth of the 3D surface when projected into the drawing surface. (This can be considered as the x-displacement of the opposite parallege edge relative to the edge (x1, y1) to (x2, y2).)
depthY	(Mandatory)	The y component of the depth of the 3D surface when projected into the drawing surface. (This can be considered as the y-displacement of the opposite parallege edge relative to the edge (x1, y1) to (x2, y2).)
edgeColor	(Mandatory)	The border color of the 3D surface. If you do not want to draw a border for the 3D surface, set the edgeColor the same as the fillColor.
fillColor	(Mandatory)	The color used to fill the 3D surface. If you do not want to fill the 3D surface, set the fillColor to Transparent.

Return Value  
None.

## sector

PHP/Perl/Python Usage

sector(cx, cy, rx, ry, a1, a2, edgeColor, fillColor)

C++ Prototype

virtual void sector(int cx, int cy, int rx, int ry, double a1, double a2, int edgeColor, int fillColor) = 0;

Description

Draw a sector. This method supports independent vertical radius and horizontal radius, so both circular and elliptical sectors can be drawn.

Arguments

Argument	Default Value	Description
----------	---------------	-------------

cx	(Mandatory)	The x coordinate of the center of the circle or ellipse that contains the sector.
cy	(Mandatory)	The y coordinate of the center of the circle or ellipse that contains the sector.
rx	(Mandatory)	The horizontal radius of the circle or ellipse that contains the sector.
ry	(Mandatory)	The vertical radius of the circle or ellipse that contains the sector.
a1	(Mandatory)	The start angle of the sector in degrees. The angle is measured clockwise, with the y-axis as the 0 degree.
a2	(Mandatory)	The end angle of the sector in degrees. The angle is measured clockwise, with the y-axis as the 0 degree.
edgeColor	(Mandatory)	The border color of the sector. If you do not want to draw a border for the sector, set the edgeColor the same as the fillColor.
fillColor	(Mandatory)	The color used to fill the sector. If you do not want to fill the sector, set the fillColor to Transparent.

Return Value

None.

## cylinder

PHP/Perl/Python Usage

cylinder(cx, cy, rx, ry, a1, a2, depthX, depthY, edgeColor, fillColor)

C++ Prototype

```
virtual void cylinder(int cx, int cy, int rx, int ry, double a1, double a2, int depthX, int depthY, int edgeColor, int fillColor) = 0;
```

Description

Draw a cylinder surface. A cylinder surface can be considered as the area spanned by moving an arc in 3D space.

Arguments

Argument	Default Value	Description
cx	(Mandatory)	The x coordinate of the center of the circle or ellipse that contains the arc that spans the cylinder surface.
cy	(Mandatory)	The y coordinate of the center of the circle or ellipse that contains the arc that spans the cylinder surface.
rx	(Mandatory)	The horizontal radius of the circle or ellipse that contains the arc that spans the cylinder surface.

ry	(Mandatory)	The vertical radius of the circle or ellipse that contains the arc that spans the cylinder surface.
a1	(Mandatory)	The start angle of the arc that spans the cylinder surface in degrees. The angle is measured clockwise, with the y-axis as the 0 degree.
a2	(Mandatory)	The end angle of the arc that spans the cylinder surface in degrees. The angle is measured clockwise, with the y-axis as the 0 degree.
depthX	(Mandatory)	The x-displacement of a vector that represents the motion of the arc to span the cylinder.
depthY	(Mandatory)	The y-displacement of a vector that represents the motion of the arc to span the cylinder.
edgeColor	(Mandatory)	The border color of the cylinder surface. If you do not want to draw a border for the cylinder surface, set the edgeColor the same as the fillColor.
fillColor	(Mandatory)	The color used to fill the cylinder surface. If you do not want to fill the cylinder surface, set the fillColor to Transparent.

Return Value  
None.

## circle

PHP/Perl/Python Usage

circle(cx, cy, rx, ry, edgeColor, fillColor)

C++ Prototype

virtual void circle(int cx, int cy, int rx, int ry, int edgeColor, int fillColor) = 0;

Description

Draw a circle or an ellipse.

Arguments

Argument	Default Value	Description
cx	(Mandatory)	The x coordinate of the center of the circle or ellipse.
cy	(Mandatory)	The y coordinate of the center of the circle or ellipse.
rx	(Mandatory)	The horizontal radius of the circle or ellipse.
ry	(Mandatory)	The vertical radius of the circle or ellipse.

edgeColor	(Mandatory)	The border color of the circle or ellipse. If you do not want to draw a border for the circle or ellipse, set the edgeColor the same as the fillColor.
fillColor	(Mandatory)	The color used to fill the circle or ellipse. If you do not want to fill the circle or ellipse, set the fillColor to Transparent.

Return Value  
None

## fill

PHP/Perl/Python Usage  
fill(x, y, color, borderColor)

C++ Prototype  
virtual void fill(int x, int y, int color, int borderColor) = 0;

Description  
Fill an area using the specified color, where the area is bounded by a given border color.

Arguments

Argument	Default Value	Description
x	(Mandatory)	The x coordinate one of the pixels inside the area to be filled.
y	(Mandatory)	The y coordinate one of the pixels inside the area to be filled.
color	(Mandatory)	The color used to fill the area.
borderColor	(Mandatory)	The color of the border that bounds the area.

Return Value  
None

## fill2

PHP/Perl/Python Usage  
fill2(x, y, color)

C++ Prototype  
virtual void fill(int x, int y, int color) = 0;

Description  
Fill an area using the specified color, where the area is defined as a continuous region having the same color.

#### Arguments

Argument	Default Value	Description
x	(Mandatory)	The x coordinate one of the pixels inside the area to be filled.
y	(Mandatory)	The y coordinate one of the pixels inside the area to be filled.
color	(Mandatory)	The color used to fill the area.

#### Return Value

None

### text

PHP/Perl/Python Usage

text(str, font, fontSize, x, y, color)

#### C++ Prototype

virtual void text(const char \*str, const char \*font, double fontSize, int x, int y, int color) = 0;

#### Description

Draw text on the image. This method is exactly the same as the [text2](#) method except that it is simplified to contain less arguments.

#### Arguments

Argument	Default Value	Description
str	(Mandatory)	A string representing the text to be drawn.
font	(Mandatory)	The font used to draw the text. See <a href="#">Font Specification</a> on how fonts are specified. If a font file contains multiple fonts, the first font is used.
fontSize	(Mandatory)	The size of the font in points.
x	(Mandatory)	The x coordinate of the reference point of the text. The location of the reference point in the text is determined by the alignment argument (see below). By default, the reference point is the top-left corner of the text.
y	(Mandatory)	The y coordinate of the reference point of the text. The location of the reference point in the text is determined by the alignment argument (see below). By default, the reference point is the top-left corner of the text.
color	(Mandatory)	The color of the text.

#### Return Value

None

## text2

PHP/Perl/Python Usage

text2(str, font, fontIndex, fontHeight, fontWidth, angle, vertical, x, y, color [, alignment])

C++ Prototype

virtual void text(const char \*str, const char \*font, int fontIndex, double fontHeight, double fontWidth, double angle, bool vertical, int x, int y, int color, Alignment alignment = TopLeft) = 0;

Description

Draw text on the image.

Arguments

Argument	Default Value	Description
str	(Mandatory)	A string representing the text to be drawn.
font	(Mandatory)	The font used to draw the text. See <a href="#">Font Specification</a> on how fonts are specified.
fontIndex	(Mandatory)	The font index of the font file in case the font file contains more than one font. See <a href="#">Font Specification</a> on how fonts are specified.
fontHeight	(Mandatory)	The height of the font in points.
fontWidth	(Mandatory)	The width of the font in points.
angle	(Mandatory)	The rotation angle of the text.
vertical	(Mandatory)	A “true” (non-zero) value indicates the text should be layout vertically, while a “false” value indicates the text should be layout horizontally. Vertical layout is mostly used in Oriental languages such as Chinese, Japanese and Korean.
x	(Mandatory)	The x coordinate of the reference point of the text. The location of the reference point in the text is determined by the alignment argument (see below). By default, the reference point is the top-left corner of the text.
y	(Mandatory)	The y coordinate of the reference point of the text. The location of the reference point in the text is determined by the alignment argument (see below). By default, the reference point is the top-left corner of the text.
color	(Mandatory)	The color of the text.
alignment	TopLeft	The location of the reference point of the text. See <a href="#">Alignment Specification</a> for possible alignment positions.

Return Value  
None

## text3

PHP/Perl/Python Usage

text3(str, font, fontSize)

C++ Prototype

```
virtual TTFTText* text(const char *str, const char *font, double fontSize) = 0;
```

Description

Create a [TTFTText](#) object that represents the text to be drawn. You may later call the [draw](#) method of the TTFTText object to draw the text.

This method is exactly the same as the [text4](#) method except that it is simplified to contain less arguments.

This method is usually used when you need to know the size of the text in order to decide where the draw the text. In this case, you can use this method to obtain a TTFTText object, and use the methods of this object to determine the text sizes first before drawing the text.

Arguments

Argument	Default Value	Description
str	(Mandatory)	A string representing the text to be drawn.
font	(Mandatory)	The font used to draw the text. See <a href="#">Font Specification</a> on how fonts are specified. If a font file contains multiple fonts, the first font is used.
fontSize	(Mandatory)	The size of the font in points.

Return Value

The [TTFTText](#) object created.

## text4

PHP/Perl/Python Usage

text4(text, font, fontIndex, fontHeight, fontWidth, angle, vertical)

C++ Prototype

```
virtual TTFTText* text(const char *text, const char *font, int fontIndex, double fontHeight, double fontWidth, double angle, bool vertical) = 0;
```

Description

Create a [TTFTText](#) object that represents the text to be drawn. You may later call the [draw](#) method of the TTFTText object to draw the text.



This method is usually used when you need to know the size of the text in order to decide where the draw the text. In this case, you can use this method to obtain a `TTFTText` object, and use the methods of this object to determine the text sizes first before drawing the text.

#### Arguments

Argument	Default Value	Description
text	(Mandatory)	A string representing the text to be drawn.
font	(Mandatory)	The font used to draw the text. See <a href="#">Font Specification</a> on how fonts are specified.
fontIndex	(Mandatory)	The font index of the font file in case the font file contains more than one font. See <a href="#">Font Specification</a> on how fonts are specified.
fontHeight	(Mandatory)	The height of the font in points.
fontWidth	(Mandatory)	The width of the font in points.
angle	(Mandatory)	The rotation angle of the text.
vertical	(Mandatory)	A “true” (non-zero) value indicates the text should be layout vertically, while a “false” value indicates the text should be layout horizontally. Vertical layout is mostly used in Oriental languages such as Chinese, Japanese and Korean.

#### Return Value

The [TTFTText](#) object created.

## close

C++ Prototype

```
virtual void close(TTFTText *text) = 0;
```

#### Description

This method is for the C++ version of `ChartDirector` only. It is not required in other versions of `ChartDirector`.

Destroy the `TTFTText` object created using [text3](#) or [text4](#).

#### Arguments

Argument	Default Value	Description
text	(Mandatory)	The <code>TTFTText</code> object to destroy.

#### Return Value

None.

## merge

PHP/Perl/Python Usage

`merge(d, x, y, align, transparency)`

C++ Prototype

`virtual void merge(const DrawArea *d, int x, int y, Alignment align, int transparency) = 0;`

Description

Apply another DrawArea image on top of the current DrawArea image.

Arguments

Argument	Default Value	Description
d	(Mandatory)	A DrawArea object representing another DrawArea image.
x	(Mandatory)	The x coordinate of a reference point within the current DrawArea image. The exactly location of the other DrawArea image relatively to the reference point will depend on the align argument (see below).
y	(Mandatory)	The y coordinate of a reference point within the current DrawArea image. The exactly location of the other DrawArea image relatively to the reference point will depend on the align argument (see below).
align	(Mandatory)	The alignment of the other DrawArea image relative to the reference point. See <a href="#">Alignment Specification</a> for possible alignment positions.
transparency	(Mandatory)	Specify the transparency level of the other DrawArea image. A value of 0 means non-transparent, while a value of 255 means totally transparent.

Return Value

None.

## tile

PHP/Perl/Python Usage

`tile(d, transparency)`

C++ Prototype

`virtual void tile(const DrawArea *d, int transparency) = 0;`

Description

Apply another DrawArea image on top of the current DrawArea image as a wallpaper. If the current DrawArea image is larger than the wallpaper image, the wallpaper image will be repeated applied on top of the current DrawArea image until the whole image is covered.

#### Arguments

Argument	Default Value	Description
d	(Mandatory)	A DrawArea object representing another DrawArea image.
transparency	(Mandatory)	Specify the transparency level of the other DrawArea image. A value of 0 means non-transparent, while a value of 255 means totally transparent.

#### Return Value

None

## load

PHP/Perl/Python Usage

load(filename)

C++ Prototype

```
virtual bool load(const char *filename) = 0;
```

#### Description

Load the specified image into the current DrawArea. The existing DrawArea image will be overwritten by the loaded image, including the size and background colors. This method will determine the image type by using the extension of the filename. The extensions png, jpg/jpeg, gif and wbmp/wmp (case insensitive) represent PNG, JPEG, GIF and WAP bitmap respectively.

#### Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the image to be loaded.

#### Return Value

A 1 (true) value indicates the load operation is successful, otherwise a 0 (false) value is returned.

## loadGIF

PHP/Perl/Python Usage

loadGIF(filename)

C++ Prototype

```
virtual bool loadGIF(const char *filename) = 0;
```

#### Description

Load the specified GIF image into the current DrawArea. The existing DrawArea image will be overwritten by the loaded image, including the size and background colors.

#### Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the image to be loaded.

#### Return Value

A 1 (true) value indicates the load operation is successful, otherwise a 0 (false) value is returned.

## loadPNG

PHP/Perl/Python Usage

loadPNG(filename)

#### C++ Prototype

```
virtual bool loadPNG(const char *filename) = 0;
```

#### Description

Load the specified PNG image into the current DrawArea. The existing DrawArea image will be overwritten by the loaded image, including the size and background colors.

#### Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the image to be loaded.

#### Return Value

A 1 (true) value indicates the load operation is successful, otherwise a 0 (false) value is returned.

## loadJPG

PHP/Perl/Python Usage

loadJPG(filename)

#### C++ Prototype

```
virtual bool loadJPG(const char *filename) = 0;
```

#### Description

Load the specified JPEG image into the current DrawArea. The existing DrawArea image will be overwritten by the loaded image, including the size and background colors.

#### Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the image to be loaded.

#### Return Value

A 1 (true) value indicates the load operation is successful, otherwise a 0 (false) value is returned.

## loadWMP

PHP/Perl/Python Usage

loadWMP(filename)

C++ Prototype

```
virtual bool loadWMP(const char *filename) = 0;
```

Description

Load the specified WAP bitmap image into the current DrawArea. The existing DrawArea image will be overwritten by the loaded image, including the size and background colors.

Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the image to be loaded.

Return Value

A 1 (true) value indicates the load operation is successful, otherwise a 0 (false) value is returned.

## out

PHP/Perl/Python Usage

out(filename)

C++ Prototype

```
virtual bool out(const char *filename) = 0;
```

Description

Write the current DrawArea image to an image file. This method will determine the image type by using the extension of the filename. The extensions png, jpg/jpeg, gif and wbmp/wmp (case insensitive) represent PNG, JPEG, GIF and WAP bitmap respectively.

Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the output image file.

Return Value

A “true” (1) value indicates the operation is successful, otherwise return “false” (0).

## outGIF

PHP/Perl/Python Usage

outGIF(filename)

C++ Prototype

```
virtual bool outGIF(const char *filename) = 0;
```

#### Description

Write the current DrawArea image to an alternative GIF image file. If you want to have the alternative GIF image in memory instead of writing to a file, use the [outGIF2](#) method instead.

#### Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the output image file.

#### Return Value

A “true” (1) value indicates the operation is successful, otherwise return “false” (0).

## outGIF2

PHP/Perl/Python Usage

outGIF2()

C++ Prototype

virtual [MemBlock](#) outGIF() = 0;

#### Description

Write the current DrawArea image as an alternative GIF image to memory. This method is usually used for web applications where the output image is directly transferred to the network. If you want to output the image to a file, use the [outGIF](#) method instead.

#### Arguments

None

#### Return Value

Returns a block of memory (as a string in PHP/Perl/Python, and as a MemBlock in C++) that contains the binary image of the chart in the requested format.

## outPNG

PHP/Perl/Python Usage

outPNG(filename)

C++ Prototype

virtual bool outPNG(const char \*filename) = 0;

#### Description

Write the current DrawArea image to a PNG image file. If you want to have the PNG image in memory instead of writing to a file, use the [outPNG2](#) method instead.

#### Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the output image file.

Return Value

A “true” (1) value indicates the operation is successful, otherwise return “false” (0).

## outPNG2

PHP/Perl/Python Usage

outPNG2()

C++ Prototype

virtual [MemBlock](#) outPNG() = 0;

Description

Write the current DrawArea image as a PNG image to memory. This method is usually used for web applications where the output image is directly transferred to the network. If you want to output the image to a file, use the [outPNG](#) method instead.

Arguments

None

Return Value

Returns a block of memory (as a string in PHP/Perl/Python, and as a MemBlock in C++) that contains the binary image of the chart in the requested format.

## outJPG

PHP/Perl/Python Usage

outJPG(filename [, quality])

C++ Prototype

virtual bool outJPG(const char \*filename, int quality = 80) = 0;

Description

Write the current DrawArea image to a JPEG image file. If you want to have the JPEG image in memory instead of writing to a file, use the [outJPG2](#) method instead.

Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the output image file.
quality	80	The quality of the image. The JPEG algorithm allows you to sacrifice image quality for compression ratio. A quality value of 95 gives a very good quality image but has low compression ratio (large image size). A low quality value (e.g. 30) gives a poorer quality image but high compression ratio.

Return Value

A “true” (1) value indicates the operation is successful, otherwise return “false” (0).

## outJPG2

PHP/Perl/Python Usage

outJPG2([quality])

C++ Prototype

```
virtual MemBlock outJPG(int quality = 80) = 0;
```

Description

Write the current DrawArea image as a JPG image to memory. This method is usually used for web applications where the output image is directly transferred to the network. If you want to output the image to a file, use the [outJPG](#) method instead.

Arguments

Argument	Default Value	Description
quality	80	The quality of the image. The JPEG algorithm allows you to sacrifice image quality for compression ratio. A quality value of 95 gives a very good quality image but has low compression ratio (large image size). A low quality value (e.g. 30) gives a poorer quality image but high compression ratio.

Return Value

Returns a block of memory (as a string in PHP/Perl/Python, and as a MemBlock in C++) that contains the binary image of the chart in the requested format.

## outWMP

PHP/Perl/Python Usage

outWMP(filename)

C++ Prototype

```
virtual bool outWMP(const char *filename) = 0;
```

Description

Write the current DrawArea image to a WAP bitmap image file. If you want to have the WAP bitmap image in memory instead of writing to a file, use the [outWMP2](#) method instead.

Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the output image file.

Return Value

A “true” (1) value indicates the operation is successful, otherwise return “false” (0).



## outWMP2

PHP/Perl/Python Usage

outWMP2()

C++ Prototype

```
virtual MemBlock outWMP() = 0;
```

Description

Write the current DrawArea image as a WAP bitmap image to memory. This method is usually used for web applications where the output image is directly transferred to the network. If you want to output the image to a file, use the [outWMP](#) method instead.

Arguments

None

Return Value

Returns a block of memory (as a string in PHP/Perl/Python, and as a MemBlock in C++) that contains the binary image of the chart in the requested format.

## setPaletteMode

PHP/Perl/Python Usage

setPaletteMode(p)

C++ Prototype

```
virtual void setPaletteMode(PaletteMode p) = 0;
```

Description

Set the palette mode when writing the image to a PNG file.

The PNG format supports both palette based images and true color images. Palette based images can only support 256 colors, but can be smaller in size.

The current supported palette modes are as follows:

Palette Mode	Description
TryPalette	Use palette mode if the image contains less than 256 colors. Use true color mode if the image contains more than 256 colors. This is the default setting.
ForcePalette	Use palette mode even if the image contains more than 256 colors. In this case, dithering operation will be applied to reduce the image to 256 colors.
NoPalette	Use true color mode regardless of the actual number of colors in the image.

#### Arguments

Argument	Default Value	Description
p	(Mandatory)	The palette mode for PNG images. See above for description.

#### Return Value

None.

## setDitherMethod

PHP/Perl/Python Usage

setDitherMethod(m)

#### C++ Prototype

```
virtual void setDitherMethod(DitherMethod m) = 0;
```

#### Description

Set the dithering method if dithering is required.

Dithering is an operating to reduce the colors of an image. It is required if an image has more colors than can be supported by the image format. For example, a GIF image can only have 256 colors. If the actual image contains more than 256 colors, dithering is used to reduce the image to less than 256 colors.

In ChartDirector, if dithering is needed, the image will be reduced to the standard 216 colors web-safe color palette.

The ChartDirector supports several common dithering algorithms as follows: (The explanation of the dithering algorithms is outside the scope of this documentation. Please refer to Computer Graphics text book for explanations.)

Dithering Mode	Description
Quantize	For any pixel, adjust it to the nearest color in the standard 216 colors web-safe color palette.
OrderedDither	Use ordered dithering algorithm with a 4 x 4 matrix.
ErrorDiffusion	Use the Floyd and Steinberg error diffusion algorithm. This is the default setting.

#### Arguments

Argument	Default Value	Description
m	(Mandatory)	The dithering algorithm to use, in case it is necessary to do dithering. See above for description.

Return Value  
None.

## setTransparentColor

PHP/Perl/Python Usage  
setTransparentColor(c)

C++ Prototype  
virtual void setTransparentColor(int c) = 0;

### Description

Set the transparent color for the image when writing the image to an image file. This only applies if the image file format supports transparent color (such as GIF and PNG).

### Arguments

Argument	Default Value	Description
c	(Mandatory)	The color that is designated as the transparent color.

Return Value  
None

## setAntiAliasText

PHP/Perl/Python Usage  
setAntiAliasText(a)

C++ Prototype  
virtual void setAntiAliasText(AntiAliasMode a) = 0;

### Description

Set whether anti-alias methods are used when drawing text. Currently, three anti-alias modes are supported in ChartDirector as follows:

Dithering Mode	Description
NoAntiAlias	Do not use anti-alias method to draw text.
AntiAlias	Always use anti-alias method to draw text.
AutoAntiAlias	Automatically determine if the font is suitable for using anti-alias. Currently, the algorithm will apply anti-alias “large” and/or “bold” fonts. It is because anti-aliasing small fonts could cause the font to become less readable. This is the default setting.

### Arguments

Argument	Default Value	Description
a	(Mandatory)	The text anti-alias mode. See above for description.

Return Value  
None

## setInterlace

PHP/Perl/Python Usage  
setInterlace(i)

C++ Prototype  
virtual void setInterlace(bool i) = 0;

### Description

Set the interlace mode when writing the image to an image file. This only applies to image file format that supports interlacing (GIF and PNG).

### Arguments

Argument	Default Value	Description
i	(Mandatory)	A “true” (non-zero) value means the image is interlaced. A “false” value means the image is non-interlaced. Note that sometimes an interlaced image is less compressible, and therefore may have a large image size. The default is non-interlace.

Return Value  
None

## setColorTable

PHP/Perl/Python Usage  
setColorTable(colors, offset)

C++ Prototype  
virtual void setColorTable([IntArray](#) colors, int offset) = 0;

### Description

Change the colors of the palette color table starting with the specified offset position in the palette color table. See [Color Specification](#) for how palette color tables are used in ChartDirector.

Note that this color table is different from the palette table that is saved with a palette image. All palette images in ChartDirector area always saved using the web-safe 216 colors palette table.

### Arguments

Argument	Default Value	Description
colors	(Mandatory)	A list of colors to replace the colors in the palette color table.

offset	(Mandatory)	The offset position that marks start position within the palette color table where the colors will be replaced.
--------	-------------	---

Return Value

None

## getARGBColor

PHP/Perl/Python Usage

getARGBColor(c)

C++ Prototype

```
virtual int getARGBColor(int c) = 0;
```

Description

Change the given color to ARGB format if the given color is a palette table color. If the given color is already in ARGB format, the same value is returned.

Arguments

Argument	Default Value	Description
c	(Mandatory)	The color to be changed to ARGB format.

Return Value

The ARGB color converted from the given color.

## dashLineColor

PHP/Perl/Python Usage

dashLineColor(color, patternCode)

C++ Prototype

```
virtual int dashLineColor(int color, int dashPattern) = 0;
```

Description

A dash line color is a “dynamic” color that switches on and off periodically. When it is used to draw a line, the line will appear as a dash line.

The dashLineColor method is used to create the dash line color. It accepts the dash line specification as input, and returns a 32-bit integer acting as the “handle” of the dash line color. You can use the handle in any ChartDirector API that expects a color as an argument.

The style of the dash line is defined by a “pattern code”. The “pattern code” is a 4-byte (that is, 32-bit) integer, where each byte alternatively represents the number of pixels that should be switched on or off.

For the example, the pattern code 0x01020304 means the first 1 pixel should be switched on, followed by 2 pixels switched off, followed by 3 pixels switched on, followed by 4 pixels switched off. After that, the pattern will repeat itself from the first pixel.

ChartDirector comes from several predefined constants for common dash line patterns. This is illustrated in the following table.

Predefined Constants	Value	Dash Line Style
DashLine	0x00000505	- - - - -
DotLine	0x00000202	.....
DotDashLine	0x05050205	- . - . - . - .
AltDashLine	0x0A050505	- - - - -

#### Arguments

Argument	Default Value	Description
color	(Mandatory)	The color to draw the dash line.
patternCode	(Mandatory)	A 32-bit integer representing the style of the dash line.

#### Return Value

Return a 32 bit integer acting as a “handle” to the dash line color.

## patternColor

PHP/Perl/Python Usage

patternColor(colorArray, height [, startX, startY])

#### C++ Prototype

virtual int patternColor([IntArray](#) c, int h, int startX = 0, int startY = 0) = 0;

#### Description

A pattern color is a “dynamic” color that changes according to a 2D periodic pattern. When it is used to fill an area, the area will look like being tiled with a wallpaper pattern.

The patternColor method accepts an array of colors as the bitmap pattern. If you would like to use a pattern from an image file, use the [patternColor2](#) method instead.

In the patternColor method, the bitmap specification consists an array of integers representing the color of the bitmap pixels, and the height of the bitmap. ChartDirector will automatically compute the width of the bitmap as the size of the array divided by the height (since size of array = width x height).

The patternColor method returns a 32-bit integer acting as the “handle” of the pattern color. You can use the handle in any ChartDirector API that expects a color as an argument.

By default, the top-left corner of the pattern is aligned with the image coordinate (0, 0). In some cases, it may be required that the pattern be aligned with some other points. The optional startX and startY parameters may be used to provide an alternative reference point.

#### Arguments

Argument	Default Value	Description
colorArray	(Mandatory)	An array of 32-bit integers representing the colors of the pixels of the bitmap pattern. The color of the pixel at (x, y) should correspond to index (x + y * width - 1) of the array.
height	(Mandatory)	The height of the bitmap pattern in pixels.
startX	0	The x coordinate of a reference point to align the pattern.
startY	0	The y coordinate of a reference point to align the pattern.

#### Return Value

Return a 32 bit integer acting as a “handle” to the pattern color.

## patternColor2

PHP/Perl/Python Usage

patternColor(filename [, startX, startY])

#### C++ Prototype

```
virtual int patternColor(const char *filename, int startX = 0, int startY = 0) = 0;
```

#### Description

A pattern color is a “dynamic” color that changes according to a 2D periodic pattern. When it is used to fill an area, the area will look like being tiled with a wallpaper pattern.

The patternColor2 method is used to create the pattern color using an image file as the pattern. If you would like to use a pattern from memory, use the [patternColor](#) method instead.

The patternColor2 method will auto-detect the image file format using the file name extension, which must either be png, jpg, jpeg, gif, wbmp or wmp (case insensitive).

The patternColor2 method returns a 32-bit integer acting as the “handle” of the pattern color. You can use the handle in any ChartDirector API that expects a color as an argument.

By default, the top-left corner of the pattern is aligned with the image coordinate (0, 0). In some cases, it may be required that the pattern be aligned with some other points. The optional startX and startY parameters may be used to provide an alternative reference point.

#### Arguments

Argument	Default Value	Description
filename	(Mandatory)	An image file providing the pattern.
startX	0	The x coordinate of a reference point to align the pattern.
startY	0	The y coordinate of a reference point to align the pattern.

Return Value

Return a 32 bit integer acting as a “handle” to the pattern color.

## gradientColor

PHP/Perl/Python Usage

gradientColor(startX, startY, endX, endY, startColor, endColor)

C++ Prototype

```
virtual int gradientColor(int startX, int startY, int endX, int endY, int startColor, int endColor) = 0;
```

Description

A gradient color is a “dynamic” color that changes progressively from one color to other colors along a direction. A simple gradient color contains two colors (a starting color and an ending color) along the direction, while a complex gradient color may contain many intermediate colors along the direction.

The gradientColor method creates a simple gradient color. To create a complex gradient color, please refer to the [gradientColor2](#) method.

In the gradientColor method, the gradient color is defined using a starting color, an ending color, and a gradient line segment. The gradient line segment is a reference line segment to determine the direction and span of the gradient color. If a pixel is located outside the span of the gradient line, the gradient line will be repeated like tiling with a wallpaper pattern.

The gradientColor method returns a 32-bit integer acting as the “handle” of the pattern color. You can use the handle in any ChartDirector API that expects a color as an argument.

Arguments

Argument	Default Value	Description
startX	(Mandatory)	The x coordinate of the starting point of the gradient line segment.
startY	(Mandatory)	The y coordinate of the starting point of the gradient line segment.
endX	(Mandatory)	The x coordinate of the ending point of the gradient line segment.
endY	(Mandatory)	The y coordinate of the ending point of the gradient line segment.
startColor	(Mandatory)	The starting color.
endColor	(Mandatory)	The ending color.

Return Value

Return a 32 bit integer acting as a “handle” to the gradient color.



## gradientColor2

PHP/Perl/Python Usage

`gradientColor2(colorArray [, angle [, scale [, startX, startY]]])`

C++ Prototype

`virtual int gradientColor(const int * colorArray, double angle = 90, double scale = 1, int startX = 0, int startY = 0) = 0;`

Description

A gradient color is a “dynamic” color that changes progressively from one color to other colors along a direction. A simple gradient color contains two colors (a starting color and an ending color) along the direction, while a complex gradient color may contain many intermediate colors along the direction.

The `gradientColor2` method creates a complex gradient color. To create a simple gradient color, please refer to the [gradientColor](#) method.

In the `gradientColor2` method, the gradient color is defined using an array of positions and their colors along a “gradient line segment”. During definition, the gradient line segment is always assumed as horizontal and 256 pixels in length. When actually drawing with the gradient color, the actual direction and span of the gradient line can be controlled using the “angle” and “scale” parameters, while the starting point of the gradient line can be controlled using the “startX” and “startY” parameters.

The array of positions and their colors along the gradient line is of the following format:

```
position0, color0, position1, color1, ..... positionN, colorN
```

where “positionX” is the pixel position along the gradient line, and “colorX” is the color at that position. During definition, the gradient line it is always assumed to be 256 pixels in length, so “position0” is always 0, and the last position is always 256 (or 0x100).

For example, the array:

```
0, 0xFF0000, 0x80, 0xFFFF00, 0x100, 0x00FF00
```

means pixel 0 will be of red color 0xFF0000, pixel 128 (0x80) will be of yellow color (0xFFFF00) and pixel 256 will be of the green color (0x00FF00). Between pixel 0 and pixel 128, the color will change from red to yellow progressively. Between pixel 128 and pixel 256, the color will change from yellow to green progressively.

If a pixel is located outside the span of the gradient line, the gradient line will be repeated like tiling with a wallpaper pattern.

The `gradientColor2` method returns a 32-bit integer acting as the “handle” of the gradient color. You can use the handle in any `ChartDirector` API that expects a color as an argument.

One common use of `gradientColor2` is to define gradient colors that have a metallic look and feel. `ChartDirector` comes from several predefined gradient arrays for common metallic colors – gold, silver, red metal, green metal and blue metal.

Predefined Gradients	Value
goldGradient	0, 0xFFE743, 0x60, 0xFFFFFE0, 0xB0, 0xFFFF0B0, 0x100, 0xFFE743
silverGradient	0, 0xC8C8C8, 0x60, 0xF8F8F8, 0xB0, 0xE0E0E0, 0x100, 0xC8C8C8
redMetalGradient	0, 0xE09898, 0x60, 0xFFF0F0, 0xB0, 0xF0D8D8, 0x100, 0xE09898
greenMetalGradient	0, 0x98E098, 0x60, 0xF0FFF0, 0xB0, 0xD8F0D8, 0x100, 0x98E098
blueMetalGradient	0, 0x9898E0, 0x60, 0xF0F0FF, 0xB0, 0xD8D8F0, 0x100, 0x9898E0

**Note for PHP developers:** In ChartDirector for PHP, the above pre-defined gradients are implemented as global variables. When using them inside a PHP function, a line such as “global \$goldGradient;” is required to make the variable accessible inside the function. Please refer to your PHP documentation for details on variable scopes.

## TTFText

The TTFText object represents text strings. It is created by [text3](#) or [text4](#) method of the DrawArea object.

Method	Description
<a href="#">getWidth</a>	Get the width of the bounding box of the text.
<a href="#">getHeight</a>	Get the height of the bounding box of the text.
<a href="#">getLineHeight</a>	Get the height of the bounding box of one line of text in pixels.
<a href="#">getLineDistance</a>	Get the line distance of the text in pixels.
<a href="#">draw</a>	Draw the text.

## getWidth

PHP/Perl/Python Usage

getWidth()

C++ Prototype

virtual int getWidth() const = 0;

Description

Get the width of the bounding box of the text.

Arguments

None.

Return Value

The width of the bounding box of the text in pixels.

## **getHeight**

PHP/Perl/Python Usage

getHeight()

C++ Prototype

virtual int getHeight() const = 0;

Description

Get the height of the bounding box of the text.

Arguments

None.

Return Value

The height of the bounding box of the text in pixels.

## **getLineHeight**

PHP/Perl/Python Usage

getLineHeight()

C++ Prototype

virtual int getLineHeight() const = 0;

Description

Get the height of the bounding box of one line of text in pixels. The line height may be different from the height of the text, because a text may contain multiple lines (by embedding the newline character '\n' in the text).

Arguments

None.

Return Value

The line height of the bounding box of one line of text in pixels.

## **getLineDistance**

PHP/Perl/Python Usage

getLineDistance()

C++ Prototype

virtual int getLineDistance() const = 0;

Description

Get the line distance of the text in pixels. The line distance is the distance between two lines in pixels.

Arguments

None.

Return Value

The line distance of the text in pixels.

## draw

PHP/Perl/Python Usage

draw(x, y, color, alignment)

C++ Prototype

```
virtual void draw(int x, int y, int color, Alignment alignment) const = 0;
```

Description

Draw the text.

Arguments

Argument	Default Value	Description
x	(Mandatory)	The x coordinate of the reference point of the text. The location of the reference point in the text is determined by the alignment argument (see below). By default, the reference point is the top-left corner of the text.
y	(Mandatory)	The y coordinate of the reference point of the text. The location of the reference point in the text is determined by the alignment argument (see below). By default, the reference point is the top-left corner of the text.
color	(Mandatory)	The color of the text.
alignment	TopLeft	The location of the reference point of the text. See <a href="#">Alignment Specification</a> for possible alignment positions.

Return Value

None.

---

# ChartDirector API Index

---

---

## A

addAreaLayer · 226, 227  
addBarLayer · 220  
addBarLayer3 · 221  
addCandleStickLayer · 229  
addCustomAggregateLabel · 265  
addCustomDataLabel · 262  
addDataSet · 256  
addDrawObj · 180  
addHLOCLayer · 228  
addKey · 192  
addLabel · 243  
addLegend · 178  
addLine · 181  
addLineLayer · 222, 223  
addMark · 237  
addScatterLayer · 224  
addText · 180  
addTitle · 177  
addTrendLayer · 225  
addZone · 238  
Alignment Specification · 160  
AltDashLine · 318  
AntiAlias · 315  
arc · 295  
AreaLayer · 277  
ARGB Color · 155  
AutoAntiAlias · 315

---

## B

BackgroundColor · 157, 158  
BarLayer · 268  
BaseAxis · 232  
BaseChart · 172  
blueMetalGradient · 322  
Bottom · 160  
BottomCenter · 160  
BottomLeft · 160

BottomRight · 160  
Box · 164

---

## C

CandleStickLayer · 280  
Center · 160  
circle · 300  
CircleLayout · 209  
CircleSymbol · 285  
close · 305  
Color Specification · 155  
Cross2Symbol · 285  
CrossSymbol · 285  
cylinder · 299

---

## D

Dash Line Colors · 156  
DashLine · 318  
dashLineColor · 183, 317  
Data Formatting · 161  
DataColor · 158  
DataSet · 281  
destroy · 173, 290  
DiamondSymbol · 284  
DotDashLine · 318  
DotLine · 318  
draw · 324  
DrawArea · 290  
DrawArea · 288

---

## E

ErrorDiffusion · 314

---

## F

fill · 301  
Font Angle · 159  
Font Color · 159

Font File · 158  
Font Height · 159  
Font Index · 159  
Font Size · 159  
Font Specification · 158  
Font Width · 159  
FONTPATH · 159  
ForcePalette · 313  
Formatting · 161

---

## ***G***

getARGBColor · 317  
getColor · 183  
getDataSet · 257  
getDrawArea · 179  
getHeight · 292, 323  
getHTMLImageMap · 189, 195, 267  
getImageCoor · 165, 211, 266  
getImageCoor2 · 194, 267  
getLabelCoor · 211  
getLineDistance · 323  
getLineHeight · 323  
getPixel · 293  
getWidth · 291, 322  
getXCoor · 259  
getYCoor · 260  
GIF · 188  
goldGradient · 322  
Gradient Color · 156  
gradientColor · 185, 186, 320  
gradientColor2 · 321  
greenMetalGradient · 322

---

## ***H***

hline · 294  
HLOCLayer · 278

---

## ***I***

International Character Sets · 154  
InvertedTriangleSymbol · 285  
ISO-8859-1 · 154

---

---

## ***J***

JPG · 188

---

## ***L***

Label Formatting · 161  
Latin1 · 154  
Layer · 252  
layout · 187  
Left · 160  
LeftTriangleSymbol · 285  
LegendBox · 191  
line · 293  
Line · 170  
LineColor · 157, 158  
LineLayer · 270  
load · 307  
loadGIF · 307  
loadJPG · 308  
loadPNG · 308  
loadWMP · 309

---

## ***M***

makeChart · 187, 188, 189  
Mark · 250  
merge · 306

---

## ***N***

No Value Specification · 160  
NoAntiAlias · 315  
NoPalette · 313  
Numeric Formatting · 163

---

## ***O***

OrderedDither · 314  
out · 309  
outGIF · 309, 310  
outJPG · 311, 312  
outPNG · 310, 311  
outWMP · 312, 313  
Overlay · 255

---

---

## ***P***

Palette · 157, 158  
Palette Colors · 157  
Parameter Substitution · 161  
Pattern Color · 156  
patternColor · 184, 318  
patternColor2 · 319  
PieChart · 197, 199  
pixel · 292  
PlotArea · 230  
PNG · 188  
polygon · 296

---

## ***Q***

Quantize · 314

---

## ***R***

rect · 295  
redMetalGradient · 322  
Right · 160  
RightTriangleSymbol · 285

---

## ***S***

ScatterLayer · 273  
sector · 206, 298  
Sector · 207  
set3D · 200, 253  
setAggregateLabelFormat · 263  
setAggregateLabelStyle · 264  
setAlignment · 170  
setAntiAliasText · 315  
setAutoScale · 247  
setBackground · 165, 174, 230, 231  
setBarGap · 270  
setBgColor · 292  
setBgImage · 176  
setBorder · 175  
setBorderColor · 254  
setColor · 171, 182, 210  
setColors · 182, 235  
setColorTable · 316  
setData · 206

setDataColor · 282  
setDataCombineMethod · 255, 256  
setDataGap · 279, 281  
setDataLabelFormat · 261, 286  
setDataLabelStyle · 261, 287  
setDataName · 282  
setDataSymbol · 284  
setDataSymbol2 · 285  
setDataSymbol3 · 286  
setDitherMethod · 314  
setDrawOnTop · 252  
setExplode · 207  
setFontAngle · 168  
setFontColor · 169  
setFontSize · 167  
setFontStyle · 167  
setGapColor · 272, 274  
setGridColor · 231  
setGridWidth · 232  
setImageMapWidth · 272, 274, 276  
setIndent · 243  
setInterlace · 316  
setKeySize · 193  
setLabelFormat · 201, 208, 250  
setLabelGap · 233  
setLabelLayout · 202, 209  
setLabelPos · 205, 209  
setLabels · 240  
setLabelStyle · 202, 208, 233  
setLegend · 256  
setLinearScale · 241, 242, 245  
setLineColor · 204  
setLineWidth · 252, 254, 284  
setLogScale · 246  
setMargin · 169  
setMarkColor · 251  
setPaletteMode · 313  
setPieSize · 199  
setPlotArea · 219  
setPos · 164, 171  
setSize · 164, 174, 291  
setStartAngle · 200  
setText · 167, 193  
setTickDensity · 249  
setTickLength · 236, 237  
setTitle · 234

setTitlePos · 234  
setTopMargin · 249  
setTransparentColor · 176, 315  
setUseYAxis2 · 283  
setValue · 251  
setWallpaper · 175  
setWidth · 171, 236  
setXData · 258, 259  
setYAxisOnRight · 218  
Side · 255  
SideLayout · 209  
silverGradient · 322  
SquareSymbol · 284  
Stack · 255  
surface · 297  
swapXY · 218  
syncYAxis · 217

---

## ***T***

text · 302, 303, 304  
Text Formatting · 161  
TextBox · 166  
TextColor · 157  
tile · 306  
Top · 160  
TopCenter · 160  
TopLeft · 160  
TopRight · 160  
Transparent · 156  
TrendLayer · 275  
TriangleSymbol · 285

TryPalette · 313  
TTFTText · 322

---

## ***U***

Unicode · 154  
UTF8 · 154

---

## ***V***

Vertical Layout · 159  
vline · 294

---

## ***W***

Western European Character Set · 154  
WMP · 188

---

## ***X***

xAxis · 215  
XAxis · 239  
xAxis2 · 215  
XYChart · 212, 214

---

## ***Y***

yAxis · 216  
YAxis · 244  
yAxis2 · 216