ChartDirector
Ver 1.5

C++ Programmer's Manual

Advanced Software Engineering Limited
www.advsofteng.com

ASE doc code: cdcpp150004

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 this 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.

Public Version

The public 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 public version of the ChartDirector software without charge.

- You may redistribute the unmodified public version of the ChartDirector software, provided you do not charge for it.

- You may embed the unmodified public 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 Limited's web site at 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 public 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 Limited’s web site at www.advsofteng.com.

Support Contact

Please refer to Advanced Software Engineering Limited’s web site at www.advsofteng.com.

Table Of Contents

1

Introduction

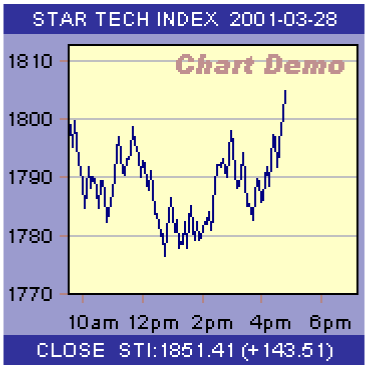
Welcome to ChartDirector

ChartDirector is a powerful software graphics library for creating professionally looking charts in PNG, JPEG, WBMP or alternative GIF[1] format.

If you are reading this document using the Adobe Acrobat Reader on a Windows based computer, to see the graphics charts clearly, the best magnification to use is 133.3% to avoid disortion due to alias.

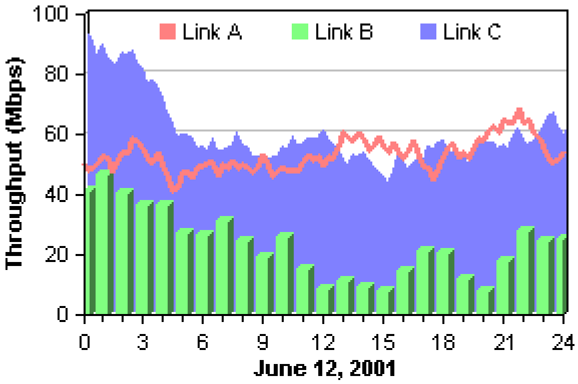
- ChartDirector employs a powerful and elegant object model. The charts are modeled as objects hierarchies, containing title objects, legend objects, axis objects, data set objects, etc. You may control the properties of those objects, providing you tremendous flexibility to design the charts you like.

Width : 180 pixels
Height : 180 pixels
Size : 1414 bytes



- ChartDirector models charts as layers. You may combine layers of various chart types on the same chart. ChartDirector supports alpha transparency coloring. You may draw some layers using semi-transparent colors to allow underlying layers to be seen.

Width : 300 pixels
Height : 200 pixels
Size : 2217 bytes

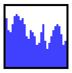


---

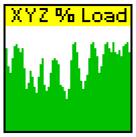
[1] Alternative GIF is not GIF. It does not use the GIF compression algorithm to avoid patent and legal problems. In practice, almost all browers and image programs can read alternative GIF, so it is an acceptable replacement for GIF. Alternative GIF produces images with larger file sizes than GIF. Because of the problems with GIF, the recommended image format to use is PNG. In most cases, PNG produces smaller images than GIF and alternative GIF, and it supports true color images (GIF only supports 256 colors). Like GIF, PNG is supported by almost all browsers and image programs.

- Ability to generate very small charts for thumbnails.

32 x 32 pixels, 166 bytes

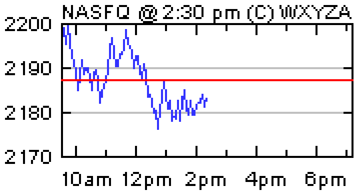


64 x 64 pixels, 314 bytes



- Very small file size, suitable for Internet applications. As shown in the demo charts in this chapter, a small thumbnail chart can be less than 200 bytes, while a complex multi-layer chart of 300 x 200 pixels consumes is just 2217 bytes. You can put a lot charts in your web page without worrying about download time.

Width: 200 pixels
Height: 100 pixels
Size : 877 bytes



Despite the flexibility, ChartDirector is very easy to use. Even relatively complex charts require just a few lines of code.

Programming Languages

The ChartDirector core is written in C++ for high performance. The ChartDirector API is available:

- C++
- Perl
- Python
- PHP
- COM / ASP / Visual Basic / VBScript / JScript

Supported Platforms

As of the writing of this document, ChartDirector is available on Windows 98/NT/ME/2000. Please visit Advanced Software Engineering’s web site at www.advsofteng.com for updated information.

2

Installation

Copying to the Hard Disk

ChartDirector is designed so that it can be installed “cleanly” without “polluting” your computer. It does not modify your registry settings, and it does not install files into your system32 directory.

To install ChartDirector:

- Create an empty subdirectory where ChartDirector will be installed in.
- Unzip the ChartDirector distribution zip file to the subdirectory.

Sample Projects

ChartDirector comes with a number of sample C++ projects under the “cppdemo” directory, together with project files for Microsoft VC++.

If you are using Microsoft VC++, you can double click on the “cppdemo.dsw” file under “cppdemo” directory. You may then try to compile the projects to verify that ChartDirector is installed correctly.

If you are using other compilers, please refer to the following section on how to set up the development environment to use ChartDirector.

Setting Up the Development Environment

When you develop a project using ChartDirector, the development environment should be configured appropriately so that it can find the proper header files and library files.

- The development environment must be configured so that it can access the headers files in the “include” subdirectory under the directory where ChartDirector is installed. That means the search path for header files should contain the ChartDirector “include” subdirectory.

  If you are using Microsoft VC++, the steps to do this are:

  - Go to the menu Project/Settings.
  - Click on the C/C++ tab.
  - In the “Category” list box, select “Preprocessor”

    - In the “Additional include directories” field, enter path of the ChartDirector “include” subdirectory.

- After compilation, the object file needs to link with “chartdir.lib”. (It should be in the “lib” subdirectory under the directory where ChartDirector is installed.)

  If you are using Microsoft VC++, the steps to do this are:

    - Go to the menu Project/Settings.
    - Click on the Link tab.
    - In the “Object/library modules” field, add the path name for the “chartdir.lib”.

- When the executable runs, it needs to be able to find the “chartdir.dll”. That means you need to copy the chartdir.dll to the same directory where the executable runs, or you need to copy it to a directory that is under the operating system search path. One common location where most DLLs are installed is the system32 directory.

Installing the ChartDirector License

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

To install the license code, simply create a one line ASCII file using Notepad or other text editor, and put the license code in that line. The whole file should contain only the license code. Name that file “chartdir.lic” and put the file in the same directory where you put the “chartdir.dll”.

After installation, the “chartdir.dll” should be in the “lib” subdirectory under the directory where you unzip ChartDirector. However, you may copy “chartdir.dll” to a different place (e.g. the system32 directory). In that case, you need to copy the “chartdir.lic” to the same place as well.

3

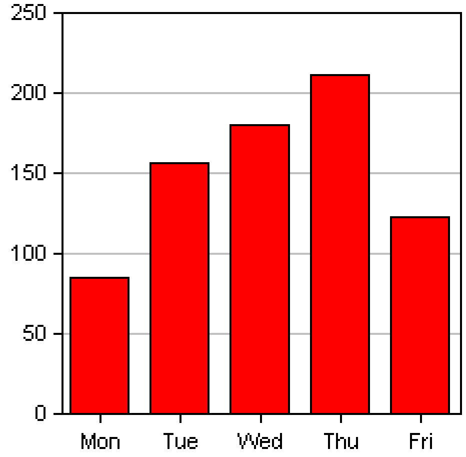
Getting Started

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 public 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 project is available in the “cppdemo\simplebar” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the bar chart
   double data[] = {85, 156, 179.5, 211, 123};

   //The labels for the bar chart
   const char *labels[] = {"Mon", "Tue", "Wed", "Thu", "Fri"};

   //First, create a XYChart of size 250 pixels x 250 pixels
   XYChart *c = XYChart::create(250, 250);

   //Set the plotarea rectange to start at (30, 30) and of
   //200 pixels in width and 200 in height
   c->setPlotArea(30, 30, 200, 200);

   //Set the x axis labels using the supplied labels
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add a bar chart layer using the supplied data
   c->addBarLayer(sizeof(data) / sizeof(data[0]), data);

   //output the chart as a PNG file
   c->makeChart("simplebar.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

The code is explained below :

- `#include "chartdir.h"`

  To use the ChartDirector API, you just need to include the “chartdir.h” header file. This header file will in turn include other necessary header files.

- `XYChart *c = XYChart::create(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->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);`

  The above line of code sets the labels on the x axis. The xAxis method retrieves the x axis object inside the XYChart object. The setLabels method binds the text labels to the x axis object. The first argument to the setLabels method is the number of text labels, while the second argument is an array of text string (a text string is char * in C++).

- `c->addBarLayer(sizeof(data) / sizeof(data[0]), data);`

  The above line adds a bar chart layer to the XYChart. In ChartDirector, any chart type that has an x axis and y axis is represented as a layer in the XYChart. An XYChart can contain a lot of layers.

You may 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 first argument is the number of data points, while the second argument is an array of double precision floating point numbers representing the values of the data points.

- `c->makeChart(filename);`

  Up to step 4, the chart is completed. We need to output it somehow. In our simple project, we just output the chart as a PNG formatted file.

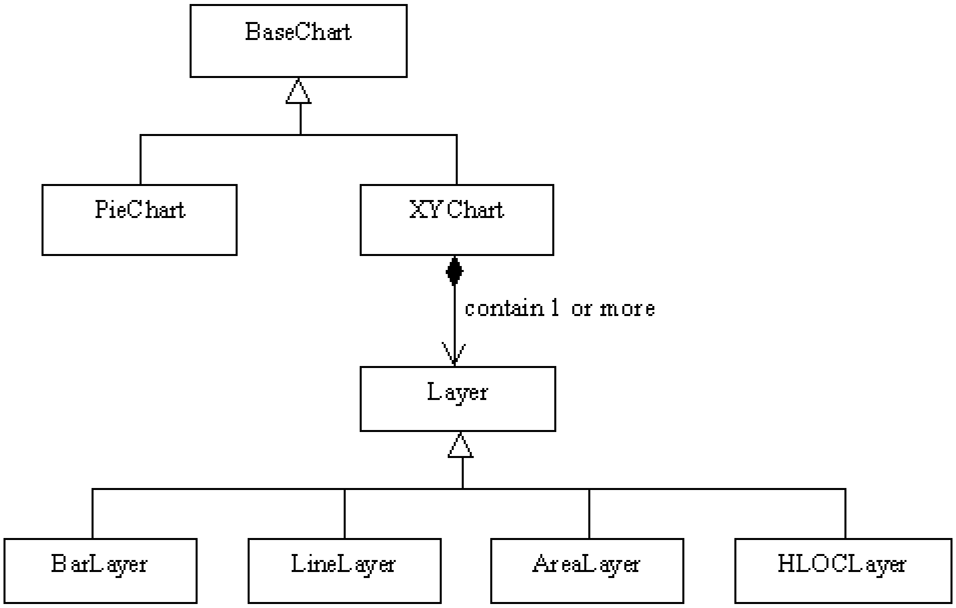
  ChartDirector supports PNG, JPEG, WBMP and alternative GIF image formats. ChartDirector can also output the image directly to memory instead of to a file. Outputting to memory is most useful for web applications where the image can then be sent directly to the browser without having to save it to an intermediate file.

- `c->destroy();`

  The above line simple destroys the XYChart object. You need to call the destroy method for every chart object you created to free up memory. You should not use the chart object any more after calling its destroy method.

Chart Object Model Overview

The ChartDirector employs a powerful and elegant object model. An overview of the top most level is shown in the diagram below.



The ChartDirector classifies charts into two main classes – PieChart and XYChart.

The PieChart class, as it name implies, represents pie charts.

The XYChart class represents all chart types that have an x axis and a y axis. The actual chart types supported by the XYChart are implemented as layers. The BarLayer, LineLayer, AreaLayer and

HLOCLayer represent bar charts, line charts, area charts and high-low-open-close charts respectively. An XYChart can contain many layers, so combination charts (e.g. combining line chart and bar chart) can be created easily.

The PieChart and XYChart are derived from the superclass BaseChart. The BaseChart represents features common to all ChartDirector charts, such as background wallpaper, chart title, legend box, etc.

Similarly, the BarLayer, LineLayer, AreaLayer and HLOCLayer are derived from the superclass Layer. The Layer represents features common to all layers.

Details of the PieChart, XYChart and various Layers will be discussed in subsequent chapters.

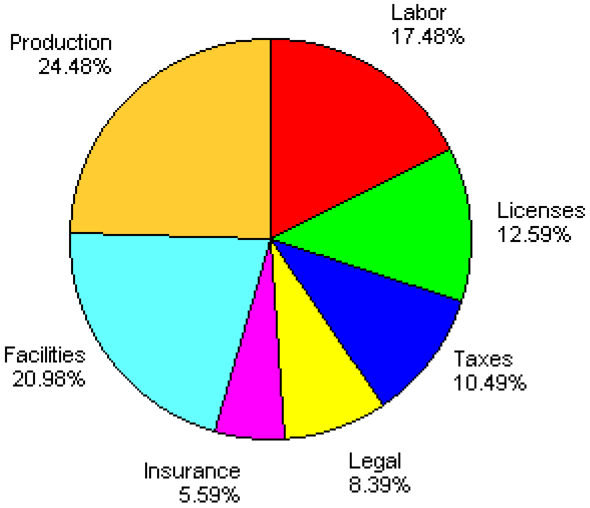
4

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 the PieChart::create method.
- 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 makeChart method.
- Destroy the chart object to free up memory using the destroy method.

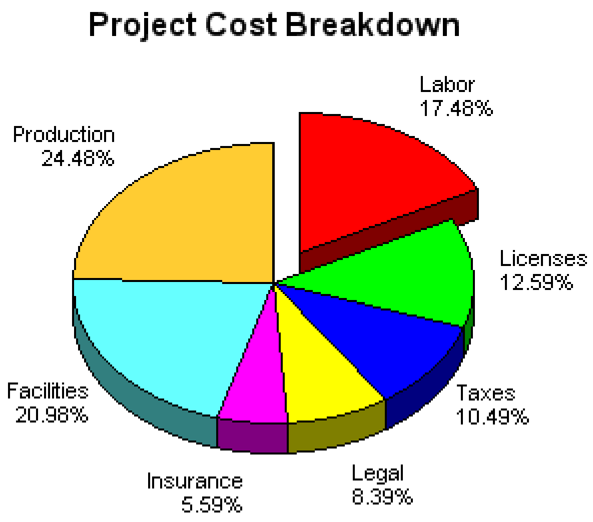
(The following project is available in the "cppdemo\simplepie" subdirectory.)

```
int main(int argc, char *argv[])
{
   //The data for the pie chart
   double data[] = { 25, 18, 15, 12, 8, 30, 35 };

   //The labels for the pie chart
   const char *labels[] = { "Labor", "Licenses", "Taxes", "Legal",
      "Insurance", "Facilities", "Production" };

   //First, create a PieChart of size 360 pixels x 300 pixels
   PieChart *c = PieChart::create(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(sizeof(data)/sizeof(data[0]), data, labels);

   //output the chart as a PNG file
   c->makeChart("simplepie.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

3D Pie Chart with Title

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



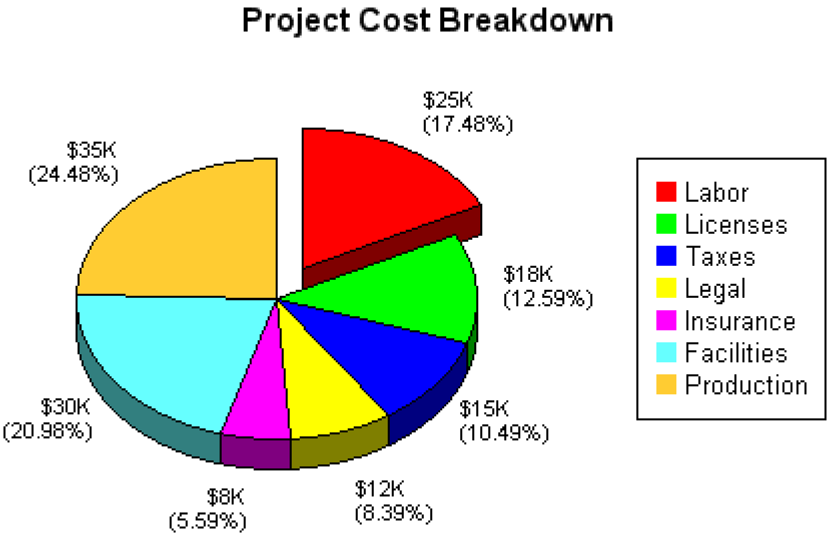
(The following project is available in the “cppdemo\threedpie” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the pie chart
   double data[] = { 25, 18, 15, 12, 8, 30, 35 };

   //The labels for the pie chart
   const char *labels[] = { "Labor", "Licenses", "Taxes", "Legal",
      "Insurance", "Facilities", "Production" };

   //First, create a PieChart of size 360 pixels x 300 pixels
   PieChart *c = PieChart::create(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(sizeof(data)/sizeof(data[0]), data, labels);

   //Explode the 1st sector
   c->sector(0)->setExplode();

   //output the chart as a PNG file
   c->makeChart("threedpie.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

Pie Chart with Legend

This example extends the previous 3D chart example to demonstrates two more ChartDirector features:

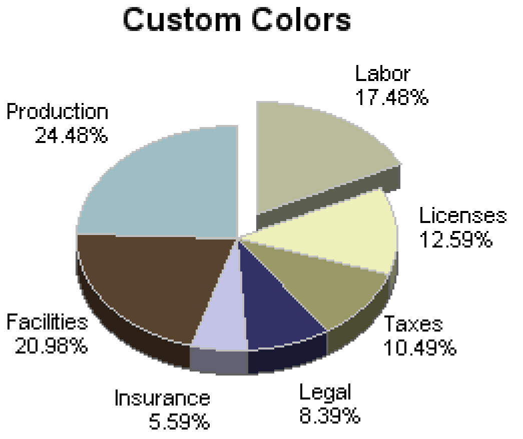
- Add a legend box using the addLegend method
- Change the label format of the sectors using the setLabelFormat method (Note that the sector labels in this example are different from the previous example.)

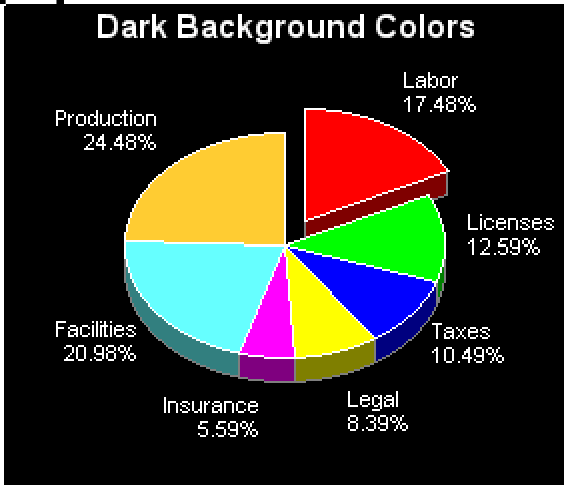


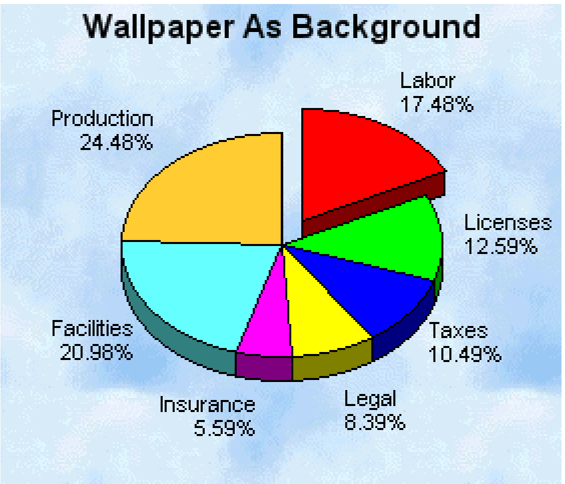
(The following project is available in the “cppdemo\legendpie” subdirectory.)

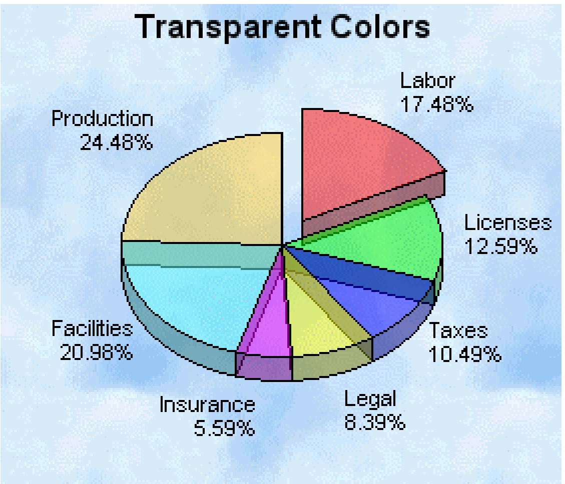
```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the pie chart
   double data[] = { 25, 18, 15, 12, 8, 30, 35 };

   //The labels for the pie chart
   const char *labels[] = { "Labor", "Licenses", "Taxes", "Legal",
      "Insurance", "Facilities", "Production" };

   //First, create a PieChart of size 450 pixels x 300 pixels
   PieChart *c = PieChart::create(450, 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
   c->addTitle("Project Cost Breakdown");

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

   //add a legend box where the top left corner is at (330, 80)
   c->addLegend(330, 80);

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

   //Set the pie data and the pie labels
   c->setData(sizeof(data)/sizeof(data[0]), data, labels);

   //Explode the 1st sector
   c->sector(0)->setExplode();

   //output the chart as a PNG file
   c->makeChart("legendpie.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

Coloring Scheme and Wallpaper







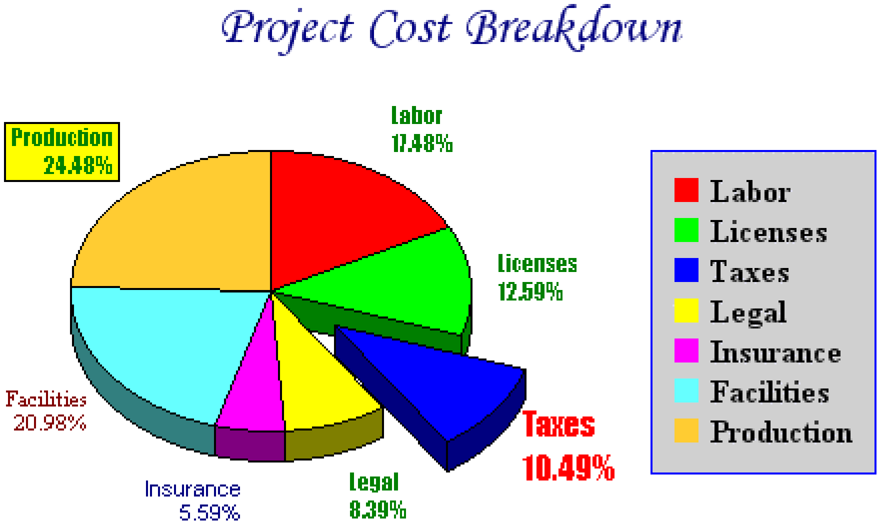


This example demonstrates how to modify the coloring scheme using the setColor, setColors and setColors(2) methods, and to apply a background image to the chart using the setWallpaper method.

(The following project is available in the “cppdemo\colorpie” subdirectory.)

```
#include "chartdir.h"

//
// colorpie
// ========
// function to draw a pie
//
// parameters:
// - colorScheme: 0 = custom coloring
//                1 = dark background coloring
//                2 = wallpaper as background
//                3 = wallpaper with semi-transparent pie
//
// - title: the title shown on the chart
// - filename: the file to save the chart
//
void colorpie(int colorScheme, const char *title, const char *filename)
{
   //The data for the pie chart
   double data[] = { 25, 18, 15, 12, 8, 30, 35 };

   //The labels for the pie chart
   const char *labels[] = { "Labor", "Licenses", "Taxes", "Legal",
      "Insurance", "Facilities", "Production" };

   //Colors of the sectors if custom coloring is used
   int colors[] = { 0xb8bc9c, 0xecf0b9, 0x999966, 0x333366, 0xc3c3e6,
      0x594330, 0xa0bdc4, -1 };

   //First, create a PieChart of size 280 pixels x 240 pixels
   PieChart *c = PieChart::create(280, 240);

   //Set the center of the pie at (140, 120) and the radius to 80 pixels
   c->setPieSize(140, 120, 80);
   //Add a title to the pie chart
   c->addTitle(title);

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

   //Set the coloring schema
   switch (colorScheme)
   {
   case 0 :
      //custom coloring, set the LineColor to light gray
      c->setColor(LineColor, 0xc0c0c0);
      //use the custom color array for the data colors (sector colors)
      c->setColors(DataColor, colors);
      break;
   case 1 :
      //dark background scheme, use the standard white on black palette
      c->setColors(whiteOnBlackPalette);
      break;
   case 2 :
      //wallpaper as background
      c->setWallpaper("bg.png");
      break;
   default :
      //transparent pie - wallpaper as background
      c->setWallpaper("bg.png");
      //use the standard semi-transparent color palette
      c->setColors(transparentPalette);
      break;
   }

   //Set the pie data and the pie labels
   c->setData(sizeof(data)/sizeof(data[0]), data, labels);

   //Explode the 1st sector
   c->sector(0)->setExplode();

   //output the chart as a PNG file
   c->makeChart(filename);

   //destroy the chart to free up resources
   c->destroy();
}


int main(int argc, char *argv[])
{
   colorpie(0, "Custom Colors", "customcolorpie.png");
   colorpie(1, "Dark Background Colors", "darkbgpie.png");
   colorpie(2, "Wallpaper As Background", "wallpaperpie.png");
   colorpie(3, "Transparent Colors", "transparentpie.png");
   return 0;
}
```

Text Style and Text Colors



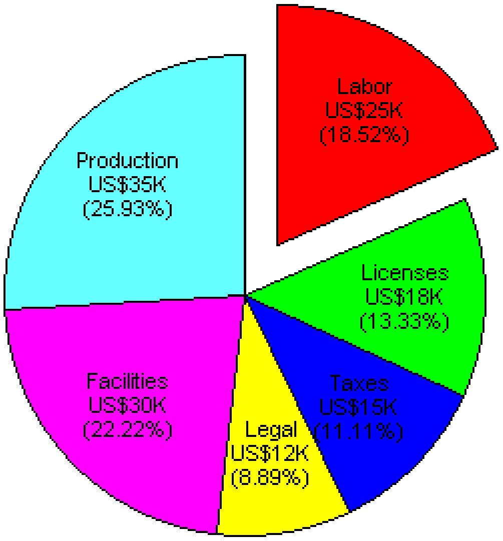
This example demonstrates how to control the text styles and text colors. Note that in the chart above, the title, the sector labels, and the legends all have different fonts and colors. One of the sector labels even has a box to highlight itself.

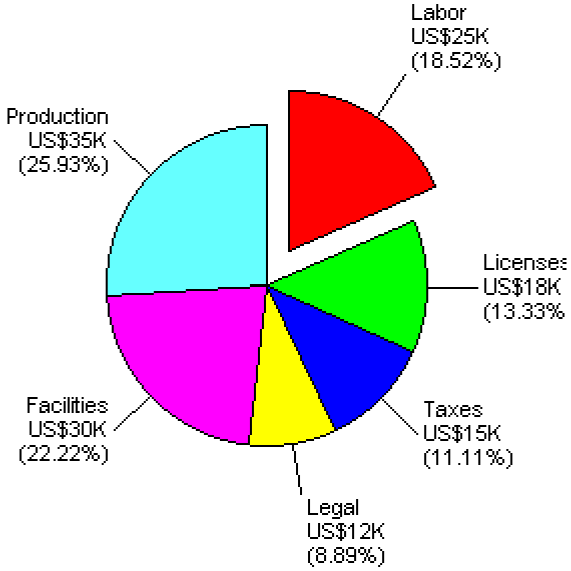
- The title text and font are specified using the addTitle method.
- The legend box font is specified using the addLegend method. The legend box background and border colors are specified using the setBackground method of the LegendBox object. (The LegendBox object is returned by the addLegend method.)
- 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 highlighting box enclosing the sector label is specified using the setBackground method of the TextBox object returned by the setLabelStyle method.

(The following project is available in the "cppdemo\fontpie" subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the pie chart
   double data[] = { 25, 18, 15, 12, 8, 30, 35 };

   //The labels for the pie chart
   const char *labels[] = { "Labor", "Licenses", "Taxes", "Legal",
      "Insurance", "Facilities", "Production" };
   //First, create a PieChart of size 480 pixels x 300 pixels
   PieChart *c = PieChart::create(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 the font Monotype Corsiva ("mtcorsva")
   //and font size of 20 points. text color is deep blue (0x000080)
   c->addTitle("Project Cost Breakdown", "mtcorsva.ttf", 20, 0x000080);

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

   //add a legend box using the font Times New Romans Bold ("timesbd.ttf") and
   //set the font size to 12 points. Set the background color of the legend box
   //to light gray (0xd0d0d0), and the border to blue (0x0000ff).
   c->addLegend(340, 80, true, "timesbd.ttf", 12)->setBackground(0xd0d0d0,
       0x0000ff);

   //set the default font for all sector labels to Impact ("impact.ttf") and font
   //size to 8 points; font color is set to drak green (0x008000).
   c->setLabelStyle("impact.ttf", 8, 0x008000);

   //Set the pie data and the pie labels
   c->setData(sizeof(data)/sizeof(data[0]), data, labels);

   //Explode the 3rd sector
   c->sector(2)->setExplode(40);

   //Use Impact/12 points/Red as font for the 3rd sector
   c->sector(2)->setLabelStyle("impact.ttf", 12, 0xff0000);

   //Use default font (Arial)/8 points/Deep Blue as the font for the 5th sector
   c->sector(4)->setLabelStyle(0, 8, 0x000080);

   //Use Times New Romans/8 points/Deep Red as the font for the 6th sector
   c->sector(5)->setLabelStyle("times.ttf", 8, 0x800000);

   //Use Impact/8 points/Deep Green as the font for the 7th sector. In addition,
   //add a background box to the label filled with yellow color (0xffff00) with
   //a black (0x000000) border.
   c->sector(6)->setLabelStyle("impact.ttf", 8, 0x008000)->setBackground(0xffff00,
      0x000000);

   //output the chart as a PNG file
   c->makeChart("fontpie.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

Label Positioning, Formatting and Join Line





This example demonstrates how to control the positions of the sector labels, and to include join lines to connect the labels to the sector perimeter using the setLabelPos method. It also demonstrates how to specify the format of the label using the setLabelFormat method.

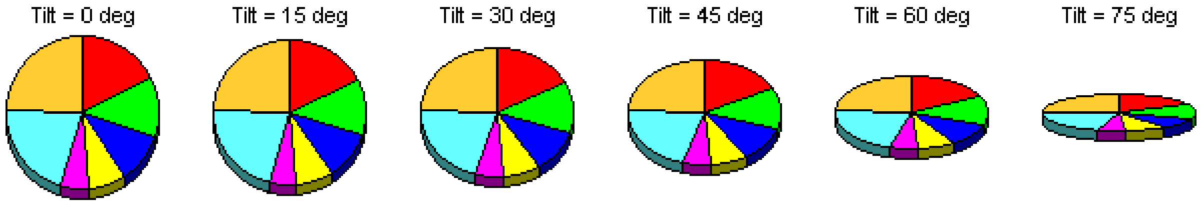
(The following project is available in the "cppdemo\labelpie" subdirectory.)

```
#include "chartdir.h"

/////////////////////////////////////////////////////////////////////////////
// Draw a pie chart where the label is on top of the pie
/////////////////////////////////////////////////////////////////////////////
void innerlabelpie()
{
   //The data for the pie chart
   double data[] = { 25, 18, 15, 12, 30, 35 };

   //The labels for the pie chart
   const char *labels[] = { "Labor", "Licenses", "Taxes", "Legal",
      "Facilities", "Production" };

   //First, create a PieChart of size 300 pixels x 300 pixels
   PieChart *c = PieChart::create(300, 300);

   //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.
   //A negative number means the label is inside the pie.
   c->setLabelPos(-40);

   //Modify the label format to contain three lines showing the sector name,
   //sector value, and the sector percentage. The sector value 99 is shown
   //as US$99K.
   c->setLabelFormat("&label&\nUS$&value&K\n(&percent&%)");

   //Set the pie data and the pie labels
   c->setData(sizeof(data)/sizeof(data[0]), data, labels);

   //Explode the 1st sector
   c->sector(0)->setExplode();

   //output the chart as a PNG file
   c->makeChart("innerlabelpie.png");

   //destroy the chart to free up resources
   c->destroy();
}

/////////////////////////////////////////////////////////////////////////////
// Draw a pie chart where the label is outside the pie
/////////////////////////////////////////////////////////////////////////////
void outerlabelpie()
{
   //The data for the pie chart
   double data[] = { 25, 18, 15, 12, 30, 35 };

   //The labels for the pie chart
   const char *labels[] = { "Labor", "Licenses", "Taxes", "Legal",
      "Facilities", "Production" };

   //First, create a PieChart of size 300 pixels x 300 pixels
   PieChart *c = PieChart::create(300, 300);

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

   //Set the label position to be 25 pixels from the pie. A positive value
   //indicates the label is outside the pie. Furthermore, include a join line
   //to link the sector label with the pie.
   c->setLabelPos(25, LineColor);

   //Modify the label format to contain three lines showing the sector name,
   //sector value, and the sector percentage. The sector value 99 is shown
   //as US$99K.
   c->setLabelFormat("&label&\nUS$&value&K\n(&percent&%)");

   //Set the pie data and the pie labels
   c->setData(sizeof(data)/sizeof(data[0]), data, labels);

   //Explode the 1st sector
   c->sector(0)->setExplode();

   //output the chart as a PNG file
   c->makeChart("outerlabelpie.png");

   //destroy the chart to free up resources
   c->destroy();
}
```

```
int main(int argc, char *argv[])
{
   innerlabelpie();
   outerlabelpie();
   return 0;
}
```

Varying 3D Depth and Angles

This example illustrates how to change the depth and tilt angle of a 3D pie from 0 to 90 degrees using the set3D method. It also demonstrates how to disable the sector labels using the setLabelStyle method.



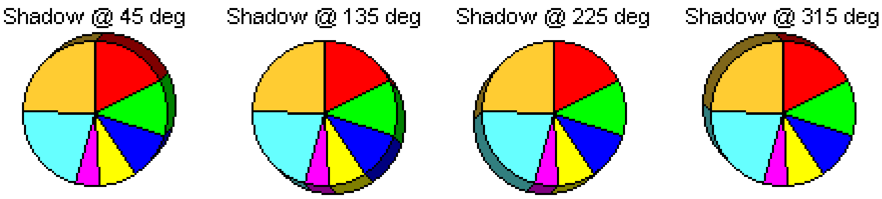
(The following project is available in the “cppdemo\3danglepie” subdirectory.)

```
#include <stdio.h>
#include "chartdir.h"

void threedanglepie(int angle, const char *filename)
{
   //The data for the pie chart
   double data[] = { 25, 18, 15, 12, 8, 30, 35 };

   //First, create a PieChart of size 100 pixels x 110 pixels
   PieChart *c = PieChart::create(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
   char buffer[256];
   sprintf(buffer, "Tilt = %d deg", angle);
   c->addTitle(buffer, "arial.ttf", 8);

   //Set the pie data and the pie labels
   c->setData(sizeof(data)/sizeof(data[0]), data);

   //Disable the sector labels by setting the color to transparent
   c->setLabelStyle(0, 8, Transparent);

   //output the chart
   c->makeChart(filename);

   //destroy the chart to free up resources
   c->destroy();
}

int main(int argc, char *argv[])
{
   threedanglepie(0, "tilt0pie.png");
   threedanglepie(15, "tilt15pie.png");
   threedanglepie(30, "tilt30pie.png");
   threedanglepie(45, "tilt45pie.png");
   threedanglepie(60, "tilt60pie.png");
   threedanglepie(75, "tilt75pie.png");
   return 0;
}
```

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 that draws the pie as an ellipse 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 to use using the set3D method.

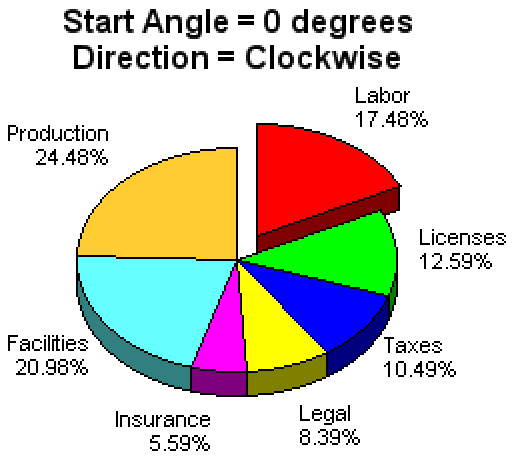
(The following project is available in the “cppdemo\shadowpie” subdirectory.)

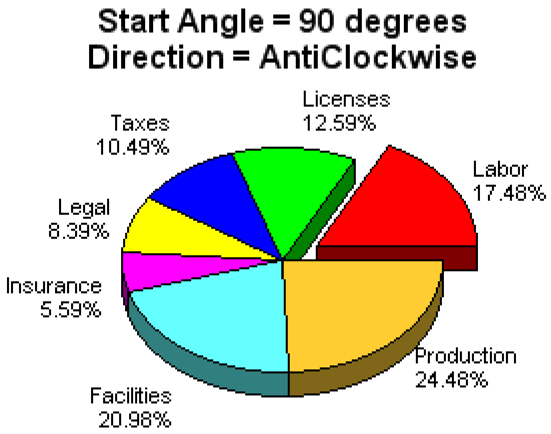
```
#include <stdio.h>
#include "chartdir.h"

void shadowpie(int angle, const char *filename)
{
   //The data for the pie chart
   double data[] = { 25, 18, 15, 12, 8, 30, 35 };

   //First, create a PieChart of size 100 pixels x 110 pixels
   PieChart *c = PieChart::create(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 tilt angle
   char buffer[256];
   sprintf(buffer, "Shadow @ %d deg", angle);
   c->addTitle(buffer, "arial.ttf", 8);

   //Set the pie data and the pie labels
   c->setData(sizeof(data)/sizeof(data[0]), data);

   //Disable the sector labels by setting the color to transparent
   c->setLabelStyle(0, 8, Transparent);

   //output the chart
   c->makeChart(filename);

   //destroy the chart to free up resources
   c->destroy();
}

int main(int argc, char *argv[])
{
   shadowpie(45, "shadow45pie.png");
   shadowpie(135, "shadow135pie.png");
   shadowpie(225, "shadow225pie.png");
   shadowpie(315, "shadow315pie.png");
   return 0;
}
```

Start Angle and Layout Direction

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

Both the start angle and the layout direction can be changed using the setStartAngle method.





(The following project is available in the “cppdemo\anglepie” subdirectory.)

```
#include <stdio.h>
#include "chartdir.h"

void anglepie(int angle, bool clockwise, const char *filename)
{
   //The data for the pie chart
   double data[] = { 25, 18, 15, 12, 8, 30, 35 };

   //The labels for the pie chart
   const char *labels[] = { "Labor", "Licenses", "Taxes", "Legal",
      "Insurance", "Facilities", "Production" };

   //First, create a PieChart of size 280 pixels x 240 pixels
   PieChart *c = PieChart::create(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
   char buffer[256];
   sprintf(buffer, "Start Angle = %d degrees\nDirection = %s",
      angle, clockwise ? "Clockwise" : "AntiClockwise");
   c->addTitle(buffer);

   //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(sizeof(data)/sizeof(data[0]), data, labels);

   //Explode the 1st sector
   c->sector(0)->setExplode();

   //output the chart
   c->makeChart(filename);

   //destroy the chart to free up resources
   c->destroy();
}

int main(int argc, char *argv[])
{
   anglepie(0, true, "angle0pie.png");
   anglepie(90, false, "angle_90pie.png");
   return 0;
}
```

5

XY Chart

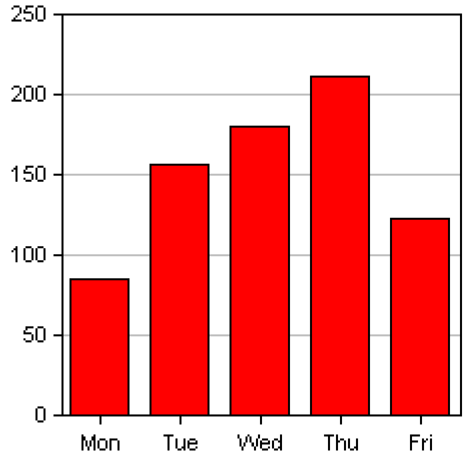
In ChartDirector, XYChart refers to any chart that has an x axis and a y axis. These include bar charts, line charts, area charts and high-low-open-close charts.

ChartDirector employs a layering architecture for XY charts. Each chart type (bar, line, area, high-low-open-close) is representing 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 the XYChart::create 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.. (The exception is if you are creating thumbnails that do not have axis labels.)
- 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 makeChart method.
- Destroy the chart object to free up memory using the destroy method.

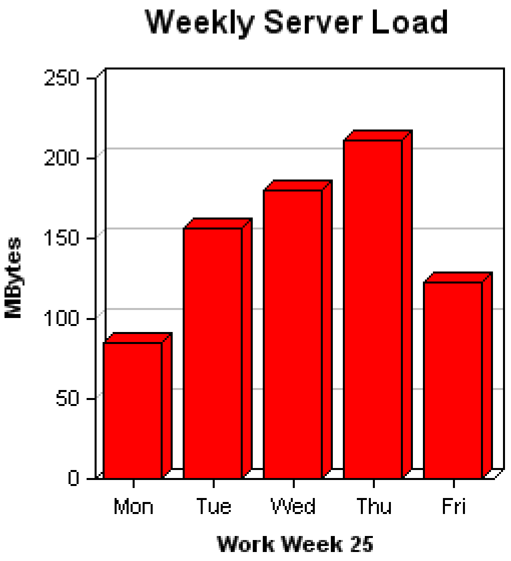
(The following project is available in the "cppdemo\simplebar" subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the bar chart
   double data[] = {85, 156, 179.5, 211, 123};

   //The labels for the bar chart
   const char *labels[] = {"Mon", "Tue", "Wed", "Thu", "Fri"};

   //First, create a XYChart of size 250 pixels x 250 pixels
   XYChart *c = XYChart::create(250, 250);

   //Set the plotarea rectanlge to start at (30, 20) and of
   //200 pixels in width and 200 in height
   c->setPlotArea(30, 20, 200, 200);

   //Set the x axis labels using the supplied labels
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add a bar chart layer using the supplied data
   c->addBarLayer(sizeof(data) / sizeof(data[0]), data);

   //output the chart as a PNG file
   c->makeChart("simplebar.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

3D Bar Chart with Titles

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

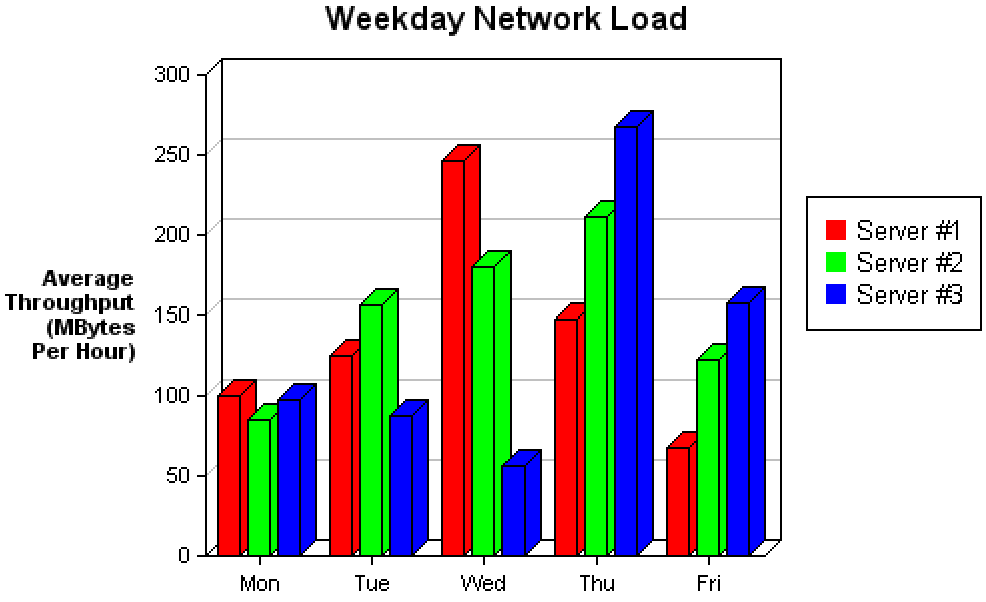
- Draw the bars in 3D using the set3D method
- Add a title to the chart using the addTitle method
- Add a title to the x axis using the setTitle method of the x axis object
- Add a title to the y axis using the setTitle method of the y axis object

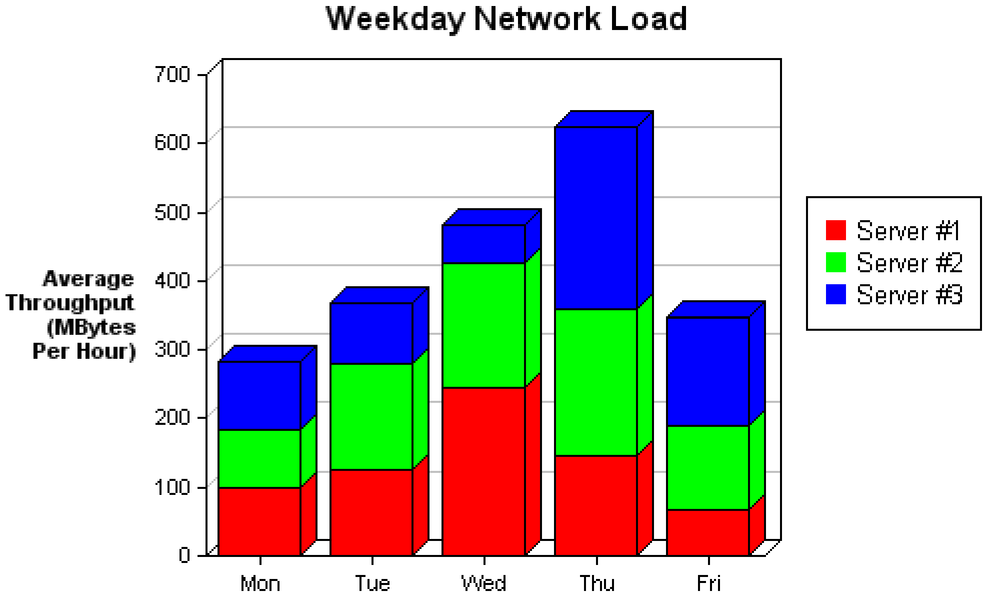


(The following project is available in the “cppdemo\threedbar” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the bar chart
   double data[] = {85, 156, 179.5, 211, 123};

   //The labels for the bar chart
   const char *labels[] = {"Mon", "Tue", "Wed", "Thu", "Fri"};

   //First, create a XYChart of size 300 pixels x 300 pixels
   XYChart *c = XYChart::create(300, 300);

   //Set the plotarea rectangle to start at (50, 40) and of
   //200 pixels in width and 200 in height
   c->setPlotArea(50, 40, 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 using the supplied data
   c->addBarLayer(sizeof(data) / sizeof(data[0]), data)->set3D();

   //Set the x axis labels using the supplied labels
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //output the chart as a PNG file
   c->makeChart("threedbar.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

Multi-Bar Chart and Stacked Bar Chart





This example introduces the concept of multiple data sets on the same layer. You may include multiple data sets on each layer of the XY Chart. The method for representing multiple data sets is different for each chart types. For bar charts, you may represent the data sets by drawing the bars side by side or by stacking the bars up.

This example also demonstrate the following features of ChartDirector:

- Add a bar layer using the addBarLayer method, then add multiple data sets to the bar layer using the addDataSet method

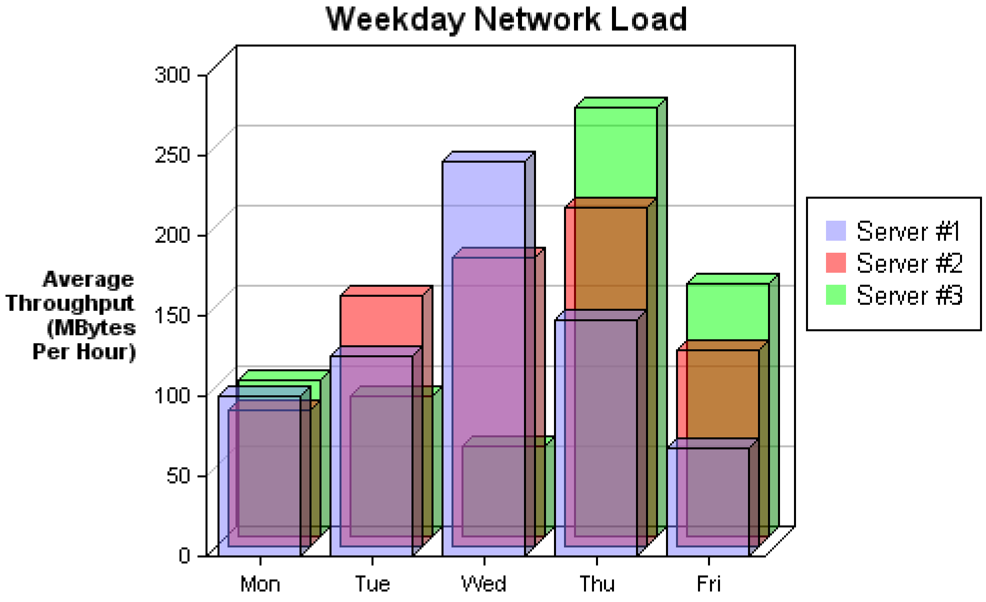
- Add a legend to the chart using the addLegend method

- 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 “\n” in the title.

(The following project is available in the “cppdemo\multibar” subdirectory.)

```
#include "chartdir.h"

void multibar(Layer::DataCombineMethod dataCombineMethod, const char *filename)
{
   //The data for the bar chart
   double data0[] = {100, 125, 245.78, 147, 67};
   double data1[]  = {85, 156, 179.5, 211, 123};
   double data2[]  = {97, 87, 56, 267, 157};

   //The labels for the bar chart
   const char *labels[] = {"Mon", "Tue", "Wed", "Thu", "Fri"};

   //First, create a XYChart of size 250 pixels x 250 pixels
   XYChart *c = XYChart::create(500, 320);

   //Set the plotarea rectangle to start at (100, 40) and of
   //280 pixels in width and 240 in height
   c->setPlotArea(100, 40, 280, 240);

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

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

   //Add a multiline title to the y axis. draw the title upright by setting the
   //font angle 0 (the default is to draw the title sideways for y axis)
   c->yAxis()->setTitle("Average\nThroughput\n(MBytes\nPer Hour)")
      ->setFontAngle(0);

   //Set the labels on the x axis
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add a bar layer and set the layer 3D depth to 8 pixels
   BarLayer *layer = c->addBarLayer(dataCombineMethod, 8);

   //Add the three data sets to the bar layer
   layer->addDataSet(sizeof(data0) / sizeof(data0[0]), data0, -1, "Server #1");
   layer->addDataSet(sizeof(data1) / sizeof(data1[0]), data1, -1, "Server #2");
   layer->addDataSet(sizeof(data2) / sizeof(data2[0]), data2, -1, "Server #3");

   //output the chart
   c->makeChart(filename);
   //destroy the chart to free up resources
   c->destroy();
}

int main(int argc, char *argv[])
{
   multibar(Layer::Stack, "stackbar.png");
   multibar(Layer::Side, "sidebar.png");
   return 0;
}
```

Depth Bar Chart



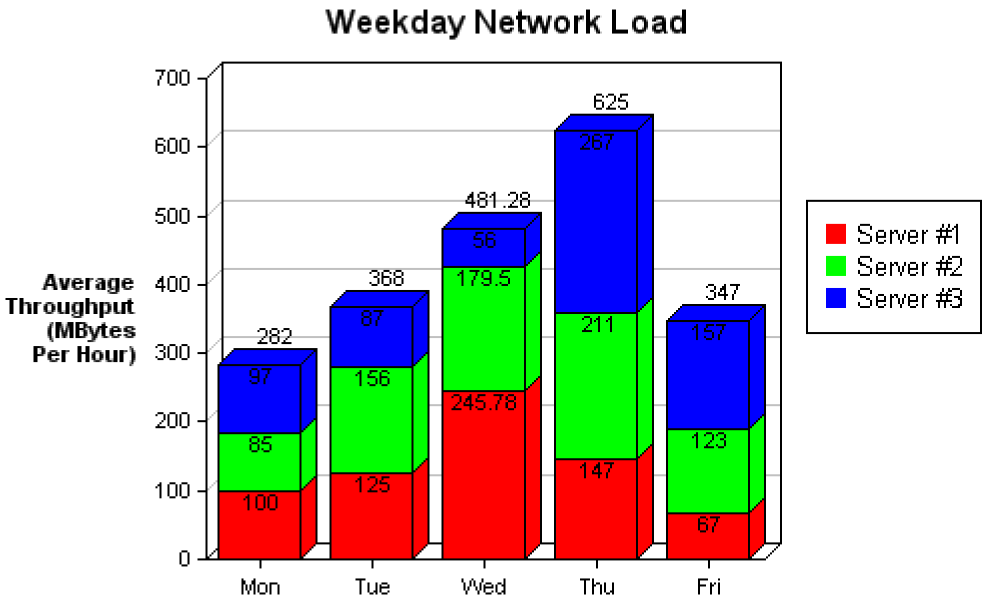
This example illustrates how you could display multiple data sets by using layers. ChartDirector allows you to any arbitrary XY chart types for each layer. In this particular example, all layers are of bar chart type.

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

(The following project is available in the “cppdemo\depthbar” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the bar chart
   double data0[] = {100, 125, 245.78, 147, 67};
   double data1[]  = {85, 156, 179.5, 211, 123};
   double data2[]  = {97, 87, 56, 267, 157};
   //The labels for the bar chart
   const char *labels[] = {"Mon", "Tue", "Wed", "Thu", "Fri"};

   //First, create a XYChart of size 500 pixels x 320 pixels
   XYChart *c = XYChart::create(500, 320);

   //Set the plotarea rectangle to start at (100, 40) and of
   //280 pixels in width and 240 in height
   c->setPlotArea(100, 40, 280, 240);

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

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

   //Add a multiline title to the y axis. draw the title upright by setting the
   //font angle 0 (the default is to draw the title sideways for y axis)
   c->yAxis()->setTitle("Average\nThroughput\n(MBytes\nPer Hour)")
      ->setFontAngle(0);

   //Set the labels on the x axis
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add three bar layers, each represent one data set
   c->addBarLayer(sizeof(data0) / sizeof(data0[0]), data0, 0x808080ff,
      "Server #1", 5);
   c->addBarLayer(sizeof(data1) / sizeof(data1[0]), data1, 0x80ff0000,
      "Server #2", 5);
   c->addBarLayer(sizeof(data2) / sizeof(data2[0]), data2, 0x8000ff00,
      "Server #3", 5);

   //output the chart as a PNG file
   c->makeChart("depthbar.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

Bar Label



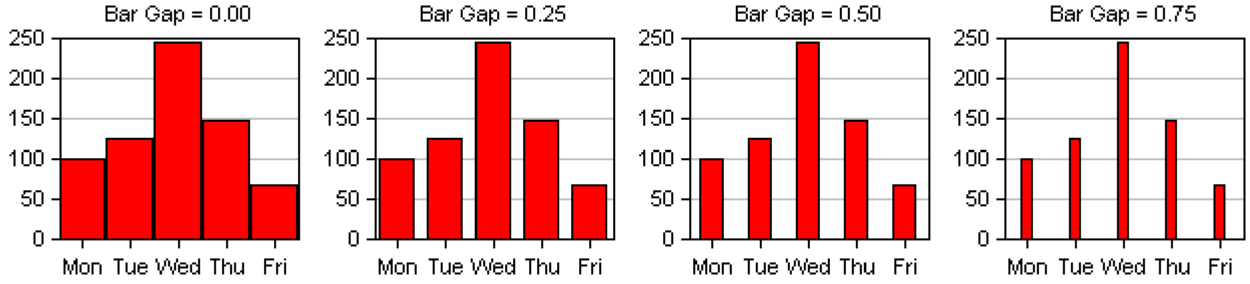
This example illustrates how you could add labels to the whole bar using the setAggregateLabelStyle method, as well as add labels to bar segment using the setDataLabelStyle method.

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.

(The following project is available in the "cppdemo\labelbar" subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the bar chart
   double data0[] = {100, 125, 245.78, 147, 67};
   double data1[]  = {85, 156, 179.5, 211, 123};
   double data2[]  = {97, 87, 56, 267, 157};

   //The labels for the bar chart
   const char *labels[] = {"Mon", "Tue", "Wed", "Thu", "Fri"};

   //First, create a XYChart of size 500 pixels x 320 pixels
   XYChart *c = XYChart::create(500, 320);

   //Set the plotarea rectangle to start at (100, 40) and of
   //280 pixels in width and 240 in height
   c->setPlotArea(100, 40, 280, 240);

   //Add a legend box at (400, 100)
   c->addLegend(400, 100);
   //Add a title to the chart
   c->addTitle("Weekday Network Load");

   //Add a multiline title to the y axis. draw the title upright by setting the
   //font angle 0 (the default is to draw the title sideways for y axis)
   c->yAxis()->setTitle("Average\nThroughput\n(MBytes\nPer Hour)")
      ->setFontAngle(0);

   //Set the labels on the x axis
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add a bar layer and set the layer 3D depth to 8 pixels
   BarLayer *layer = c->addBarLayer(Layer::Stack, 8);

   //Add the three data sets to the bar layer
   layer->addDataSet(sizeof(data0) / sizeof(data0[0]), data0, -1, "Server #1");
   layer->addDataSet(sizeof(data1) / sizeof(data1[0]), data1, -1, "Server #2");
   layer->addDataSet(sizeof(data2) / sizeof(data2[0]), 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();

   //output the chart
   c->makeChart("labelbar.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

Bar Gap



This example illustrates the effects of manipulating the bar gap using the setBarGap method.

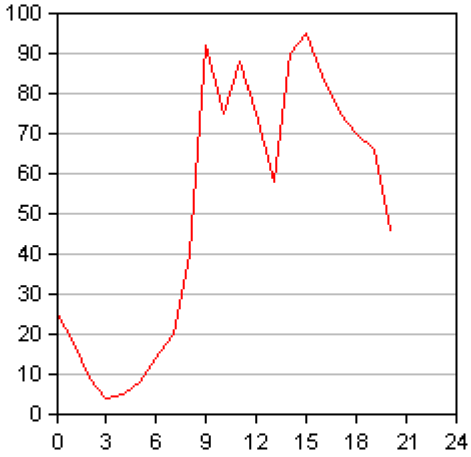
(The following project is available in the “cppdemo\gapbar” subdirectory.)

```
#include <stdio.h>
#include "chartdir.h"
```

```
void gapbar(double bargap, const char *filename)
{
   //The data for the bar chart
   double data[] = {100, 125, 245.78, 147, 67};

   //The labels for the bar chart
   const char *labels[] = {"Mon", "Tue", "Wed", "Thu", "Fri"};

   //First, create a XYChart of size 150 pixels x 150 pixels
   XYChart *c = XYChart::create(150, 150);

   //Set the plotarea at (25, 20) and dimension 120(w) x 100(h)
   c->setPlotArea(25, 20, 120, 100);

   //add a title to display to bar gap using 8 point arial font
   char buffer[256];
   sprintf(buffer, "      Bar Gap = %0.2f", bargap);
   c->addTitle(buffer, "arial.ttf", 8);

   //Set the x axis labels using the supplied labels
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add a bar chart layer using the supplied data and set the bar gap
   c->addBarLayer(sizeof(data) / sizeof(data[0]), data)->setBarGap(bargap);;

   //output the chart
   c->makeChart(filename);

   //destroy the chart to free up resources
   c->destroy();
}

int main(int argc, char *argv[])
{
   gapbar(0, "gapbar00.png");
   gapbar(0.25, "gapbar25.png");
   gapbar(0.5, "gapbar50.png");
   gapbar(0.75, "gapbar75.png");
   return 0;
}
```

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. The only difference is that instead of using the addBarLayer method to create a bar chart layer, the addLineLayer method is used to create a line chart layer.



Note that the line stops in the middle before reaching the right boundary of the plot area. ChartDirector supports using a special constant "NoValue" to denote that a point has no value. The chart on the right is drawn with the last few data points specified as "NoValue". You can also use "NoValue" in the middle of a line to make a broken line.

Although this example is based on line chart, the "NoValue" special constant is also applicable to other XY charts.

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

- Create a XYChart object using the XYChart::create 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.. (The exception is if you are creating thumbnails that do not have axis labels.)
- 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 makeChart method.
- Destroy the chart object to free up memory using the destroy method.

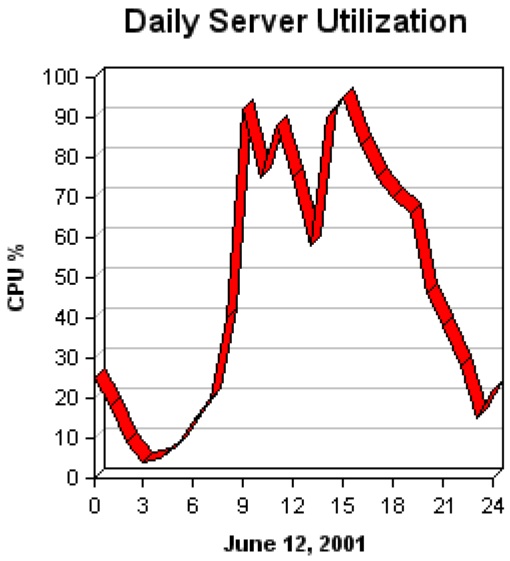
(The following project is available in the "cppdemo\simpleline" subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the line chart
   double data[] = {25, 18, 9, 4, 5, 8, 14, 20, 40, 92, 75, 88, 75,
      58, 90, 95, 83, 75, 70, 66, 46, NoValue, NoValue, NoValue, NoValue};

   //The labels for the line chart
   const char *labels[] = {"0", "", "", "3", "", "", "6", "", "", "9", "", "",
      "12", "", "", "15", "", "", "18", "", "", "21", "", "", "24"};
   //First, create a XYChart of size 250 pixels x 250 pixels
   XYChart *c = XYChart::create(250, 250);

   //Set the plotarea at (30, 20) and of 200 pixel (w) x 200 pixels (h)
   c->setPlotArea(30, 20, 200, 200);

   //Add a line chart layer using the supplied data
   c->addLineLayer(sizeof(data) / sizeof(data[0]), data);

   //Set the x axis labels using the supplied labels
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //output the chart as a PNG file
   c->makeChart("simpleline.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

3D Line Chart

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

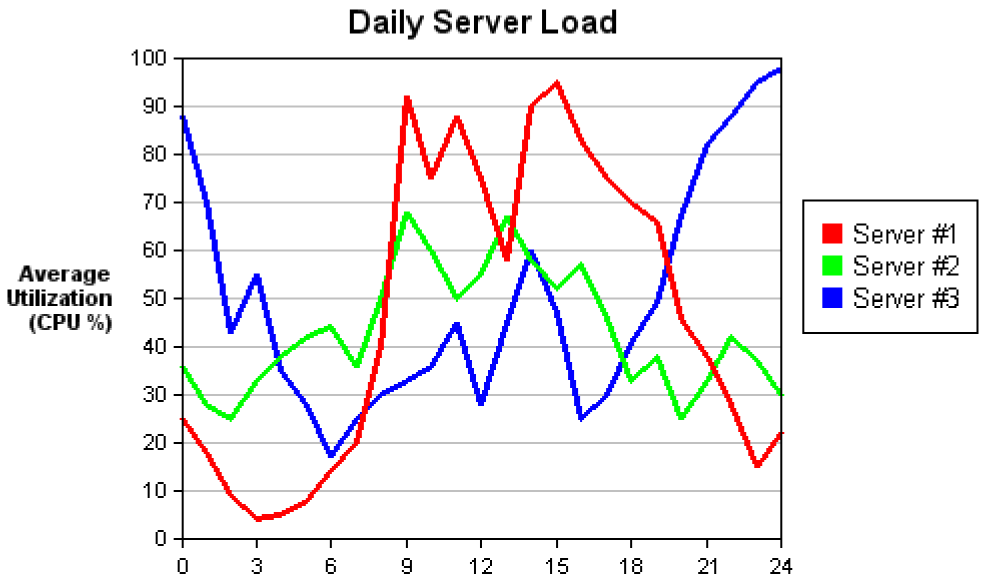
- Draw the line in 3D using the set3D method
- Add a title to the chart using the addTitle method
- Add a title to the x axis using the setTitle method of the x axis object
- Add a title to the y axis using the setTitle method of the y axis object



(The following project is available in the “cppdemo\threedline” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the line chart
   double data[] = {25, 18, 9, 4, 5, 8, 14, 20, 40, 92, 75, 88, 75,
      58, 90, 95, 83, 75, 70, 66, 46, 38, 28, 15, 22};
   //The labels for the line chart
   const char *labels[] = {"0", "", "", "3", "", "", "6", "", "", "9", "", "",
      "12", "", "", "15", "", "", "18", "", "", "21", "", "", "24"};

   //First, create a XYChart of size 300 pixels x 300 pixels
   XYChart *c = XYChart::create(300, 300);

   //Set the plotarea rectangle to start at (50, 40) and of
   //200 pixels in width and 200 in height
   c->setPlotArea(50, 40, 200, 200);

   //Add a title to the chart
   c->addTitle("Daily Server Utilization");

   //Add a title to the y axis
   c->yAxis()->setTitle("CPU %");

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

   //Add a line chart layer using the supplied data
   c->addLineLayer(sizeof(data) / sizeof(data[0]), data)->set3D();

   //Set the x axis labels using the supplied labels
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //output the chart as a PNG file
   c->makeChart("threedline.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

Multi-Line Chart



This example demonstrates how you could plot multiple data sets on the same line chart layer. It also demonstrates how you can set the line width in the line chart. Note that the lines in the above chart are much thicker than those in previous examples.

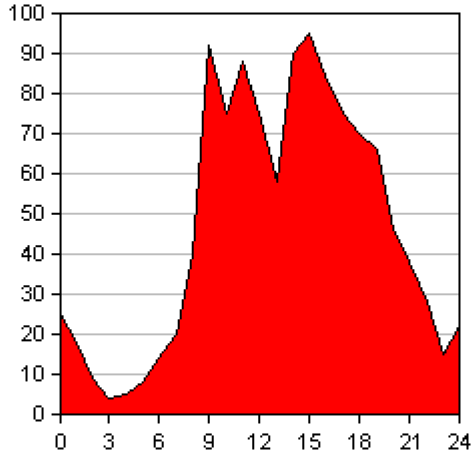
- Add a line layer using the addLineLayer method, then add multiple data sets to the line layer using the addDataSet method
- Set the line width using the setLineWidth method
- Add a legend to the chart using the addLegend method
- Add a title to the y axis using the addTitle method, and draw the title 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 “\n” in the title.

(The following project is available in the “cppdemo\multiline” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the line chart
   double data0[] = {25, 18, 9, 4, 5, 8, 14, 20, 40, 92, 75, 88, 75,
      58, 90, 95, 83, 75, 70, 66, 46, 38, 28, 15, 22};
   double data1[] = {36, 28, 25, 33, 38, 42, 44, 36, 50, 68, 60, 50, 55,
      67, 58, 52, 57, 46, 33, 38, 25, 33, 42, 37, 30};
   double data2[] = {88, 70, 43, 55, 35, 28, 17, 25, 30, 33, 36, 45, 28,
      45, 60, 47, 25, 30, 41, 49, 67, 82, 88, 95, 98};

   //The labels for the line chart
   const char *labels[] = {"0", "", "", "3", "", "", "6", "", "", "9", "", "",
      "12", "", "", "15", "", "", "18", "", "", "21", "", "", "24"};

   //First, create a XYChart of size 500 pixels x 300 pixels
   XYChart *c = XYChart::create(500, 300);

   //Set the plotarea rectangle to start at (100, 30) and of
   //300 pixels in width and 240 in height
   c->setPlotArea(100, 30, 300, 240);

   //Add a legend box at (410, 100)
   c->addLegend(410, 100);

   //Add a title to the chart
   c->addTitle("Daily Server Load");

   //Add a multiline title to the y axis. draw the title upright by setting the
   //font angle 0 (the default is to draw the title sideways for y axis)
   c->yAxis()->setTitle("Average\nUtilization\n(CPU %)")->setFontAngle(0);

   //Set the labels on the x axis
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add a line layer
   LineLayer *layer = c->addLineLayer();

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

   //Add the three data sets to the line layer
   layer->addDataSet(sizeof(data0) / sizeof(data0[0]), data0, -1, "Server #1");
   layer->addDataSet(sizeof(data1) / sizeof(data1[0]), data1, -1, "Server #2");
   layer->addDataSet(sizeof(data2) / sizeof(data2[0]), data2, -1, "Server #3");

   //output the chart as a PNG file
   c->makeChart("multiline.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

Simple Area Chart

This project 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 only 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.



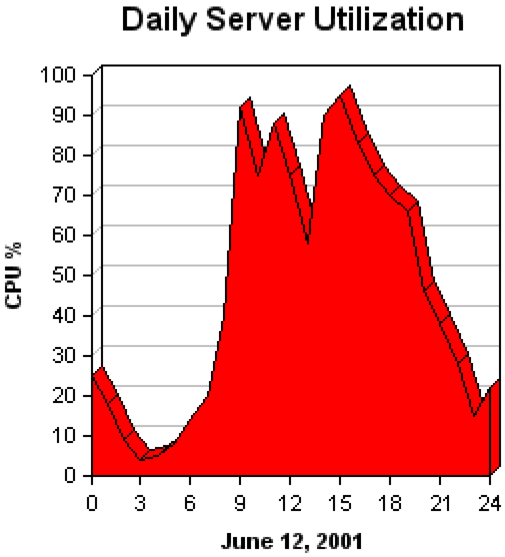
The followings are the basic steps in creating an area chart:

- Create a XYChart object using the XYChart::create 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.. (The exception is if you are creating thumbnails that do not have axis labels.)
- 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 makeChart method.
- Destroy the chart object to free up memory using the destroy method.

(The following project is available in the “cppdemo\simplearea” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the area chart
   double data[] = {25, 18, 9, 4, 5, 8, 14, 20, 40, 92, 75, 88, 75,
      58, 90, 95, 83, 75, 70, 66, 46, 38, 28, 15, 22};

   //The labels for the area chart
   const char *labels[] = {"0", "", "", "3", "", "", "6", "", "", "9", "", "",
      "12", "", "", "15", "", "", "18", "", "", "21", "", "", "24"};

   //First, create a XYChart of size 250 pixels x 250 pixels
   XYChart *c = XYChart::create(250, 250);

   //Set the plotarea at (30, 20) and of 200 pixel (w) x 200 pixels (h)
   c->setPlotArea(30, 20, 200, 200);

   //Add a area chart layer using the supplied data
   c->addAreaLayer(sizeof(data) / sizeof(data[0]), data);
   //Set the x axis labels using the supplied labels
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //output the chart as a PNG file
   c->makeChart("simplearea.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

3D Area Chart



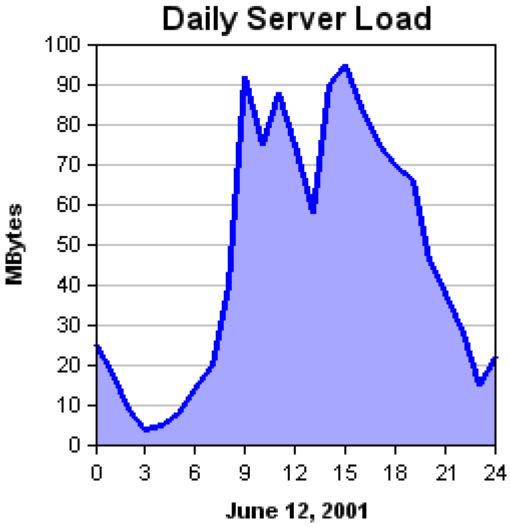
This example extends the previous simple area chart example by introducing the following features of ChartDirector:

- 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 project is available in the “cppdemo\threedarea” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the area chart
   double data[] = {25, 18, 9, 4, 5, 8, 14, 20, 40, 92, 75, 88, 75,
      58, 90, 95, 83, 75, 70, 66, 46, 38, 28, 15, 22};

   //The labels for the area chart
   const char *labels[] = {"0", "", "", "3", "", "", "6", "", "", "9", "", "",
      "12", "", "", "15", "", "", "18", "", "", "21", "", "", "24"};

   //First, create a XYChart of size 300 pixels x 300 pixels
   XYChart *c = XYChart::create(300, 300);

   //Set the plotarea rectangle to start at (50, 40) and of
   //200 pixels in width and 200 in height
   c->setPlotArea(50, 40, 200, 200);
   //Add a title to the chart
   c->addTitle("Daily Server Utilization");

   //Add a title to the y axis
   c->yAxis()->setTitle("CPU %");

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

   //Add an area chart layer using the supplied data
   c->addAreaLayer(sizeof(data) / sizeof(data[0]), data)->set3D();

   //Set the x axis labels using the supplied labels
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //output the chart as a PNG file
   c->makeChart("threedarea.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

Line Area Chart



This example illustrates how to create a special style of area chart in which the boundary line of the area chart is highlighted.

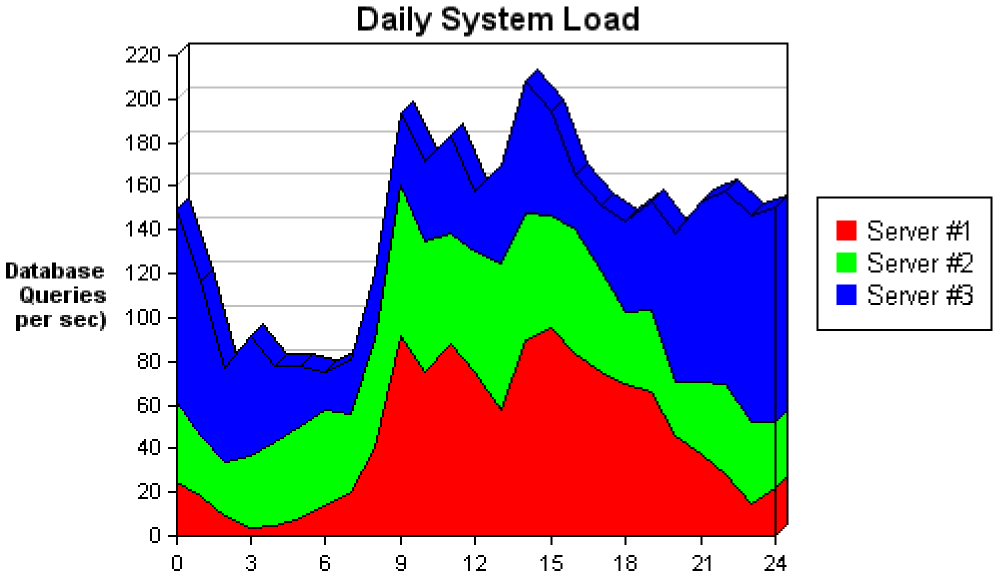
To create this style of area chart:

- Use the setDataColor method of the DataSet object to specify the area and line colors. This overrides the default line color, which is the LineColor in the color palette for an area chart (see previous Simple Area Chart example).
- Use the setLineWidth method of the layer object to increase the line width to highlight it.

(The following project is available in the “cppdemo\linearea” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the area chart
   double data[] = {25, 18, 9, 4, 5, 8, 14, 20, 40, 92, 75, 88, 75,
      58, 90, 95, 83, 75, 70, 66, 46, 38, 28, 15, 22};

   //The labels for the area chart
   const char *labels[] = {"0", "", "", "3", "", "", "6", "", "", "9", "", "",
      "12", "", "", "15", "", "", "18", "", "", "21", "", "", "24"};

   //First, create a XYChart of size 300 pixels x 280pixels
   XYChart *c = XYChart::create(300, 280);

   //Set the plotarea at (50, 25) and of 200 pixel (w) x 200 pixels (h)
   c->setPlotArea(50, 25, 200, 200);

   //Add a title to the chart
   c->addTitle("Daily Server Load");

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

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

   //Set the x axis labels using the supplied labels
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add an area layer to the chart
   AreaLayer *layer = c->addAreaLayer();

   //Add a data set to the area layer, and set the data color to
   //light blue (0xa8a8ff) and the line color to blue (0x0000ff)
   layer->addDataSet(sizeof(data) / sizeof(data[0]), data)
      ->setDataColor(0xa8a8ff, 0x0000ff);

   //Set the line width to 3 pixels to highlight the line
   layer->setLineWidth(3);

   //output the chart as a PNG file
   c->makeChart("linearea.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

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.

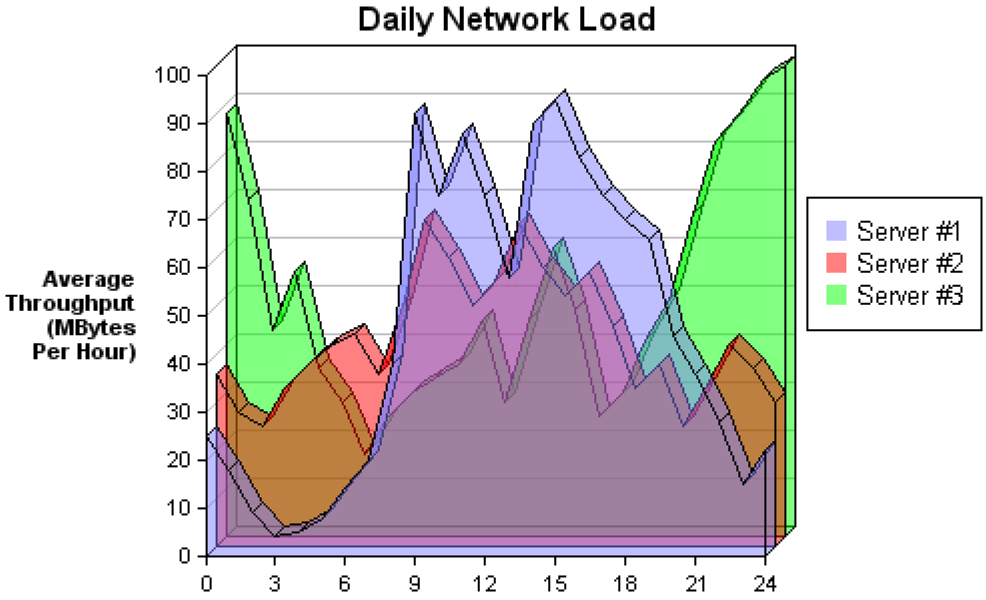
This example also demonstrate the following features of the ChartDirector:

- 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 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 “\n” in the title.

(The following project is available in the “cppdemo\stackarea” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the area chart
   double data0[] = {25, 18, 9, 4, 5, 8, 14, 20, 40, 92, 75, 88, 75,
      58, 90, 95, 83, 75, 70, 66, 46, 38, 28, 15, 22};
   double data1[] = {36, 28, 25, 33, 38, 42, 44, 36, 50, 68, 60, 50, 55,
      67, 58, 52, 57, 46, 33, 38, 25, 33, 42, 37, 30};
   double data2[] = {88, 70, 43, 55, 35, 28, 17, 25, 30, 33, 36, 45, 28,
      45, 60, 47, 25, 30, 41, 49, 67, 82, 88, 95, 98};

   //The labels for the area chart
   const char *labels[] = {"0", "", "", "3", "", "", "6", "", "", "9", "", "",
       "12", "", "", "15", "", "", "18", "", "", "21", "", "", "24"};

   //First, create a XYChart of size 500 pixels x 300 pixels
   XYChart *c = XYChart::create(500, 300);

   //Set the plotarea rectangle to start at (90, 30) and of
   //300 pixels in width and 240 in height
   c->setPlotArea(90, 30, 300, 240);

   //Add a legend box at (410, 100)
   c->addLegend(410, 100);

   //Add a title to the chart
   c->addTitle("Daily System Load");

   //Add a multiline title to the y axis. draw the title upright by setting the
   //font angle 0 (the default is to draw the title sideways for y axis)
   c->yAxis()->setTitle("Database\nQueries\nper sec)")->setFontAngle(0);

   //Set the labels on the x axis
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add an area layer
   AreaLayer *layer = c->addAreaLayer();

   //Draw the area layer in 3D
   layer->set3D();

   //Add the three data sets to the area layer
   layer->addDataSet(sizeof(data0) / sizeof(data0[0]), data0, -1, "Server #1");
   layer->addDataSet(sizeof(data1) / sizeof(data1[0]), data1, -1, "Server #2");
   layer->addDataSet(sizeof(data2) / sizeof(data2[0]), data2, -1, "Server #3");

   //output the chart as a PNG file
   c->makeChart("stackarea.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

Depth Area Chart



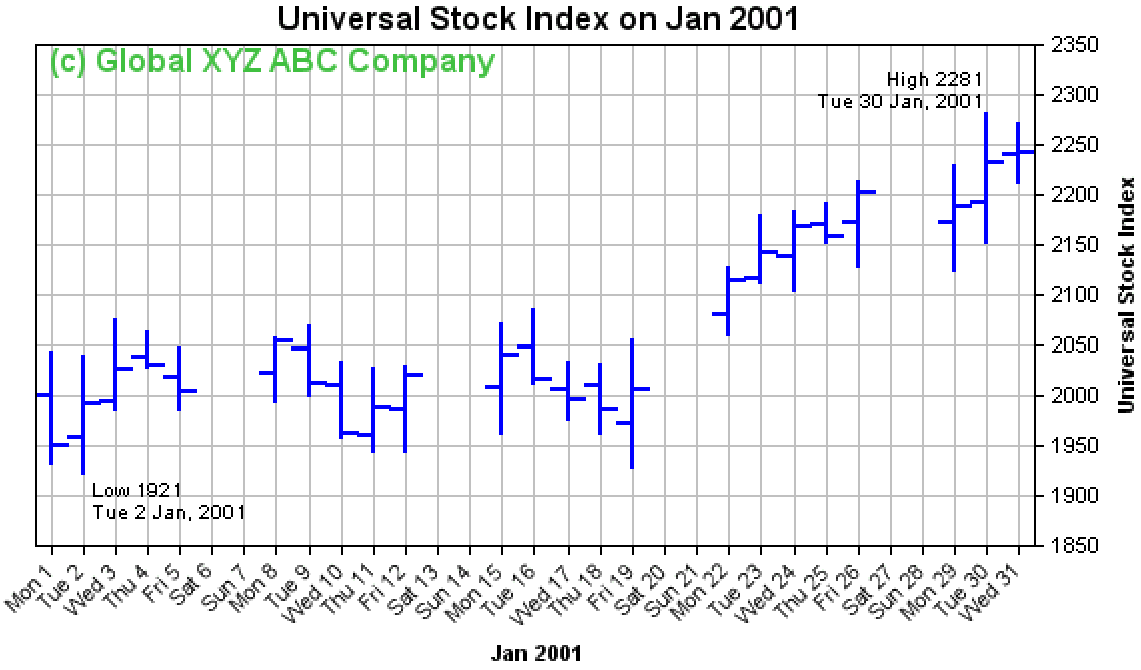
This example illustrates how you could display multiple data sets by using layers. ChartDirector allows any arbitrary XY chart types for each layer. In this particular example, all layers are of area chart type.

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 project is available in the “cppdemo\deptharea” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the area chart
   double data0[] = {25, 18, 9, 4, 5, 8, 14, 20, 40, 92, 75, 88, 75,
      58, 90, 95, 83, 75, 70, 66, 46, 38, 28, 15, 22};
   double data1[] = {36, 28, 25, 33, 38, 42, 44, 36, 50, 68, 60, 50, 55,
      67, 58, 52, 57, 46, 33, 38, 25, 33, 42, 37, 30};
   double data2[] = {88, 70, 43, 55, 35, 28, 17, 25, 30, 33, 36, 45, 28,
      45, 60, 47, 25, 30, 41, 49, 67, 82, 88, 95, 98};

   //The labels for the area chart
   const char *labels[] = {"0", "", "", "3", "", "", "6", "", "", "9", "", "",
      "12", "", "", "15", "", "", "18", "", "", "21", "", "", "24"};

   //First, create a XYChart of size 500 pixels x 320 pixels
   XYChart *c = XYChart::create(500, 320);

   //Set the plotarea rectangle to start at (100, 40) and of
   //280 pixels in width and 240 in height
   c->setPlotArea(100, 40, 280, 240);
   //Add a legend box at (400, 100)
   c->addLegend(400, 100);

   //Add a title to the chart
   c->addTitle("Daily Network Load");

   //Add a multiline title to the y axis. draw the title upright by setting the
   //font angle 0 (the default is to draw the title sideways for y axis)
   c->yAxis()->setTitle("Average\nThroughput\n(MBytes\nPer Hour)")
      ->setFontAngle(0);

   //Set the labels on the x axis
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add three area layers, each represent one data set
   c->addAreaLayer(sizeof(data0) / sizeof(data0[0]), data0, 0x808080ff,
      "Server #1", 5);
   c->addAreaLayer(sizeof(data1) / sizeof(data1[0]), data1, 0x80ff0000,
      "Server #2", 5);
   c->addAreaLayer(sizeof(data2) / sizeof(data2[0]), data2, 0x8000ff00,
      "Server #3", 5);

   //output the chart as a PNG file
   c->makeChart("deptharea.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

High-Low-Open-Close Chart



This example illustrates the high-low-open-close chart, as well as a number of advanced 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, and also the "High 2281" and "Low 1921" texts that annotate the chart.
- Dynamically compute the x and y coordinates of the custom texts using the getXCoor and getYCoor method. The x and y coordinates of the "High 2281" and "Low 1921" texts are dynamically computed 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, there are no values for 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.

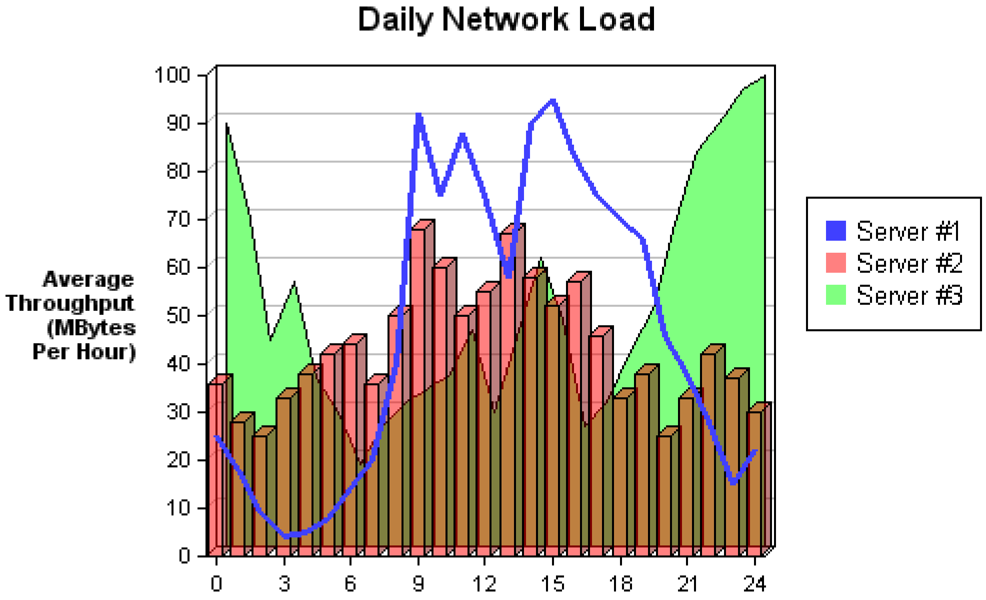
(The following project is available in the "cppdemo\hloc" subdirectory.)

```
#include "chartdir.h"
```

```
int main(int argc, char *argv[])
{
   //
   //Sample data for the HLOC chart. Represents the high, low, open and close
   //values for 31 days
   //
   double high[] = {
      2043, 2039, 2076, 2064, 2048, NoValue, NoValue, 2058, 2070, 2033, 2027,
      2029, NoValue, NoValue, 2071, 2085, 2034, 2031, 2056, NoValue, NoValue,
      2128, 2180,  2183, 2192, 2213, NoValue, NoValue, 2230, 2281, 2272
   };
   double low[] = {
      1931, 1921, 1985, 2028, 1986, NoValue, NoValue, 1994, 1999, 1958, 1943,
      1944, NoValue, NoValue, 1962, 2011, 1975, 1962, 1928, NoValue, NoValue,
      2059, 2112, 2103, 2151, 2127, NoValue, NoValue, 2123, 2152, 2212
   };
   double open[] = {
      2000, 1957, 1993, 2037, 2018, NoValue, NoValue, 2021, 2045, 2009, 1959,
      1985, NoValue, NoValue, 2008, 2048, 2006, 2010, 1971, NoValue, NoValue,
      2080, 2116, 2137, 2170, 2172, NoValue, NoValue, 2171, 2191, 2240
   };
   double close[] = {
      1950, 1991, 2026, 2029, 2004, NoValue, NoValue, 2053, 2011, 1962, 1987,
      2019, NoValue, NoValue, 2040, 2016, 1996, 1985, 2006, NoValue, NoValue,
      2113, 2142, 2167, 2158, 2201, NoValue, NoValue, 2188, 2231, 2242
   };

   //
   //The labels for the HLOC chart
   //
   const char *labels[] = {
      "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"
   };

   //First, create a XYChart of size 600 pixels x 350 pixels
   XYChart *c = XYChart::create(600, 350);

   //Set the plotarea at (50, 25) and of sizes 500(w) x 250(h). Make both the
   //horizontal and vertical grids visible by setting their colors
   c->setPlotArea(50, 25, 500, 250)->setGridColor(0xc0c0c0, 0xc0c0c0);

   //Add a title to the chart
   c->addTitle("Universal Stock Index on Jan 2001");

   //Add a custom text at (51, 21) (the upper left corner of the plotarea).
   //Use 12 point Arial bold (arialbd.ttf) as the font, and pale green as
   //the color (0x40c040).
   c->addText(51, 21, "(c) Global XYZ ABC Company", "arialbd.ttf", 12, 0x40c040);

   //Add a title to the x axis
   c->xAxis()->setTitle("Jan 2001");
```

```
    //Set the x axis labels. Display the labels at 45% rotation.
    c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels)
        ->setFontAngle(45);

    //Add a title to the y axis
    c->yAxis()->setTitle("Universal Stock Index");

    //Display the y axis on the right hand side of the plotarea
    c->setYAxisOnRight(true);

    //Leave a 10% margin on top and bottom when performing auto-scaling. The
    //margin ensures there are spaces on the top and bottom of the plot area
    //for custom text.
    c->yAxis()->setAutoScale(0.1, 0.1);

    //Add an HLOC layer using the supplied data. Set the color to blue (0x0000ff)
    HLOCLayer *layer = c->addHLOCLayer(31, high, low, open, close, 0x0000ff);

    //Set the line width to 2 for a thicker line
    layer->setLineWidth(2);

    //Layout the chart without drawing it. This computes the scaling factors
    //of the x and y axes, so we can use them to locate the custom text below.
    c->layout();

    //Add a custom text to annotate the highest point. For our sample data, the
    //highest point is 2281 at the 30th data point (index = 29). Note we use
    //the getXCoor and getYCoor to get the (x, y) coor of the text
    c->addText(layer->getXCoor(29), layer->getYCoor(2281),
        "High 2281\nTue 30 Jan, 2001", 0, 7.5, 0, BottomRight);

    //Similarly, add a custom text to annotate the lowest point, which is the
    //1921 at the 2nd sample.
    c->addText(layer->getXCoor(1), layer->getYCoor(1921),
        "Low 1921\nTue 2 Jan, 2001", 0, 7.5, 0, TopLeft);

    //output the chart as a PNG file
    c->makeChart("hloc.png");

    //destroy the chart to free up resources
    c->destroy();

    return 0;
}
```

Combination Chart

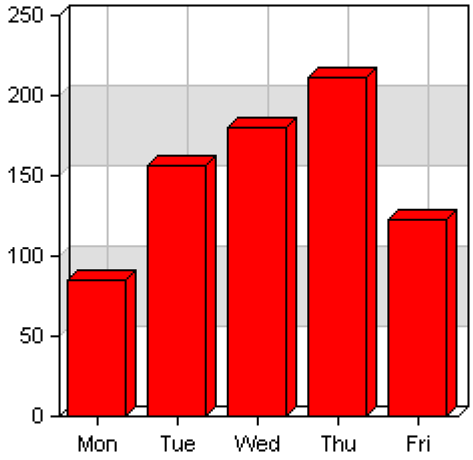


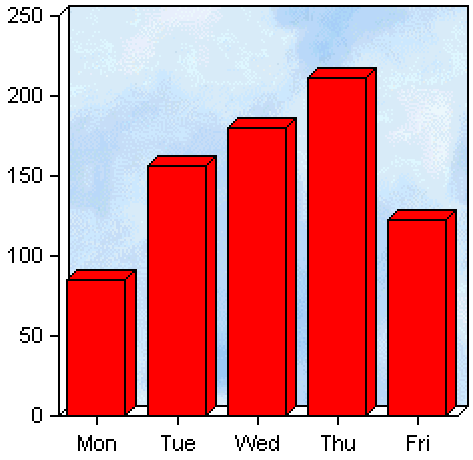
This example illustrates that you can combine various XYChart layers together to create combination charts of your choice. This example employs a line layer at the front, a bar layer at the middle and an area layer at the back. Note that each layer can be either flat or 3D and can have different depths. Also note that in this example we use a semi-transparent color for the middle bar layer so that the area layer at the back can be seen.

(The following project is available in the "cppdemo\combo" subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the combo chart
   double data0[] = {25, 18, 9, 4, 5, 8, 14, 20, 40, 92, 75, 88, 75,
      58, 90, 95, 83, 75, 70, 66, 46, 38, 28, 15, 22};
   double data1[] = {36, 28, 25, 33, 38, 42, 44, 36, 50, 68, 60, 50, 55,
      67, 58, 52, 57, 46, 33, 38, 25, 33, 42, 37, 30};
   double data2[] = {88, 70, 43, 55, 35, 28, 17, 25, 30, 33, 36, 45, 28,
      45, 60, 47, 25, 30, 41, 49, 67, 82, 88, 95, 98};

   //The labels for the combo chart
   const char *labels[] = {"0", "", "", "3", "", "", "6", "", "", "9", "", "",
      "12", "", "", "15", "", "", "18", "", "", "21", "", "", "24"};

   //First, create a XYChart of size 500 pixels x 320 pixels
   XYChart *c = XYChart::create(500, 320);

   //Set the plotarea rectangle to start at (100, 40) and of
   //280 pixels in width and 240 in height
   c->setPlotArea(100, 40, 280, 240);
   //Add a legend box at (400, 100)
   c->addLegend(400, 100);

   //Add a title to the chart
   c->addTitle("Daily Network Load");

   //Add a multiline title to the y axis. draw the title upright by setting the
   //font angle 0 (the default is to draw the title sideways for y axis)
   c->yAxis()->setTitle("Average\nThroughput\n(MBytes\nPer Hour)")
      ->setFontAngle(0);

   //Set the labels on the x axis
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add a flat line layer for the 1st data set, with a line width of 3 pixels
   c->addLineLayer(sizeof(data0) / sizeof(data0[0]), data0, 0x4040ff,
      "Server #1")->setLineWidth(3);

   //Add a 3D bar layer for the 2st data set, with a depth of 5 pixels. Use a
   //semi-transparent color so that layers at the back can be seen
   c->addBarLayer(sizeof(data1) / sizeof(data1[0]), data1, 0x80ff0000,
      "Server #2", 5);

   //Add a flat area layer for the 3rd data set
   c->addAreaLayer(sizeof(data2) / sizeof(data2[0]), data2, 0x80ff80,
      "Server #3");

   //output the chart as a PNG file
   c->makeChart("combo.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

Grids and Grid Background





The above charts illustrate different grid background styles.

The first chart illustrates alternative grid background colors, that is, two background colors (in the above example the colors are “white” and “light gray”) can be used alternatively to create horizontal bands that align with the grids.

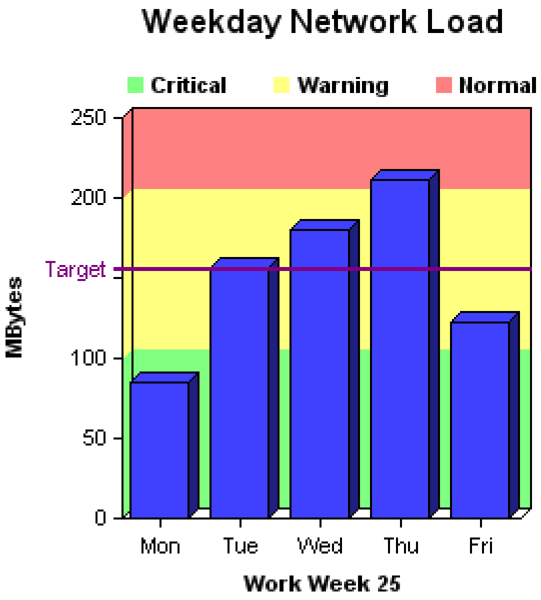
The second chart illustrates using a background image for the plot area background. It also illustrates that the grid lines can be turned off by setting their colors to Transparent.

Although this example uses bar charts for illustration, the features illustrated apply to all XY chart types.

(The following project is available in the “cppdemo\gridbg” subdirectory.)

```
#include "chartdir.h"

void gridBar(const char *filename, const char *bgimage = 0)
{
   //The data for the chart
   double data[] = {85, 156, 179.5, 211, 123};
   const char *labels[] = {"Mon", "Tue", "Wed", "Thu", "Fri"};

   //First, create a XYChart of size 250 pixels x 250 pixels
   XYChart *c = XYChart::create(250, 250);

   //Set the plotarea at (100, 40) and of size 200(w) x 200(h)
   PlotArea *p = c->setPlotArea(25, 15, 200, 200);

   if (bgimage != 0)
   {
      //use the given background image as the plot area background
      p->setBackground(bgimage);
      //turn off the grid by setting it to Transparent
      p->setGridColor(Transparent);
   }
   else
   {
      //no background image, use white and gray as two alternate
      //plotarea background colors
      p->setBackground(0xffffff, 0xe0e0e0);
      //enable horizontal and vertical grid by setting their colors
      p->setGridColor(0xc0c0c0, 0xc0c0c0);
   }

   //Set the labels on the x axis
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add a 3D bar layer with the given data
   c->addBarLayer(sizeof(data) / sizeof(data[0]), data)->set3D();

   //output the chart
   c->makeChart(filename);

   //destroy the chart to free up resources
   c->destroy();
}

int main(int argc, char *argv[])
{
   gridBar("gridband.png");
   gridBar("gridbg.png", "bg.png");
   return 0;
}
```

Marks and Zones



This example illustrates the marks and zones features of ChartDirector.

A “mark” is a line drawn on the front of the chart. The purple ”Target” line on the illustration chart on the left is a mark line. It is added to the chart using the addMark method.

A “zone” is a horizontal area on the back of the plot area. On the illustrate chart on the left, the red, yellow and green areas are zones. They are added to the chart using the addZone method.

This example also illustrates how you could position the legends on the top of the chart using the addLegend method. In addition, it illustrates how you could add custom legends using the addKey method of the LegendBox object. Note that the legends on this example are not representing the colors of the data. Rather they represent the colors of the zones.

Although this example uses bar charts for illustration, the features introduced apply to all XY chart types.

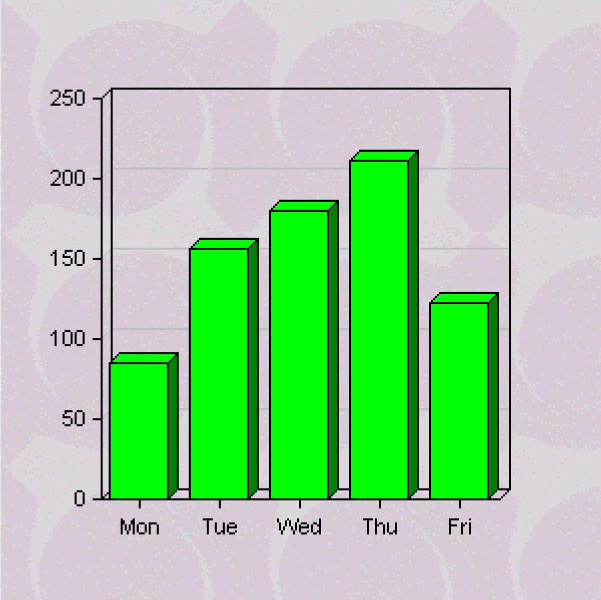
(The following project is available in the “cppdemo\markzone” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //The data for the chart
   double data[] = {85, 156, 179.5, 211, 123};
   const char *labels[] = {"Mon", "Tue", "Wed", "Thu", "Fri"};

   //First, create a XYChart of size 320 pixels x 320 pixels
   XYChart *c = XYChart::create(320, 320);
   //Set the plotarea at (60, 60) and size 200(w) x 200(h). Turn off the
   //grid lines by setting their colors to Transparent.
   c->setPlotArea(60, 60, 200, 200)->setGridColor(Transparent);

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

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

   //Add green (0x80ff80), yellow (0xffff80) and red (0xff8080) zones to
   //the y axis to represent the ranges 0 - 100, 100 - 200, and > 200.
   c->yAxis()->addZone(0, 100, 0x80ff80);
   c->yAxis()->addZone(100, 200, 0xffff80);
   c->yAxis()->addZone(200, 9999999, 0xff8080);

   //Add a purple (0x800080) mark at y = 155 using a line width of 2.
   c->yAxis()->addMark(155, 0x800080, "Target")->setLineWidth(2);

   //Add a legend box at the top of the chart at (55, 30). Use horizontal
   //layout, and set the font to 8 points Arial bold (arialbd.ttf).
   LegendBox *legend = c->addLegend(55, 30, 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("Critical", 0x80ff80);
   legend->addKey("Warning", 0xffff80);
   legend->addKey("Normal", 0xff8080);

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

   //Set the labels on the x axis
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add a 3D bar layer with the given data
   c->addBarLayer(sizeof(data) / sizeof(data[0]), data, 0x4040ff)->set3D();

   //output the chart as a PNG file
   c->makeChart("markzone.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

Wallpaper and Coloring Scheme

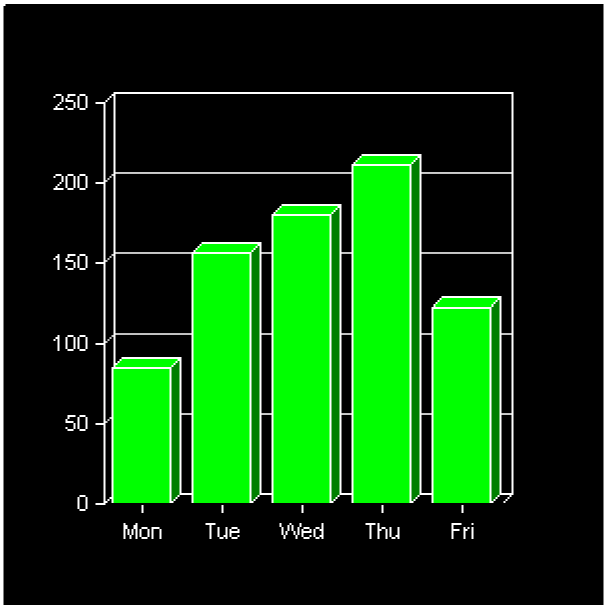
This example illustrates how to use a wallpaper as a background, and how to use an alternative coloring scheme, such as dark background coloring scheme.

The first chart on the right illustrates the effect of applying a wallpaper to the chart using the setWallpaper method. A wallpaper can be any GIF, PNG, JPG or WBMP file. The ChartDirector will tile the image to fit the chart.



The next chart on the right illustrates a dark background chart. This is done by using the setColors method to set the color palette to the default “whiteOnBlackPalette”. Note that the palette does not only change the background color. It also changes the text color and the line color to white, so that the entire chart looks good and clear.

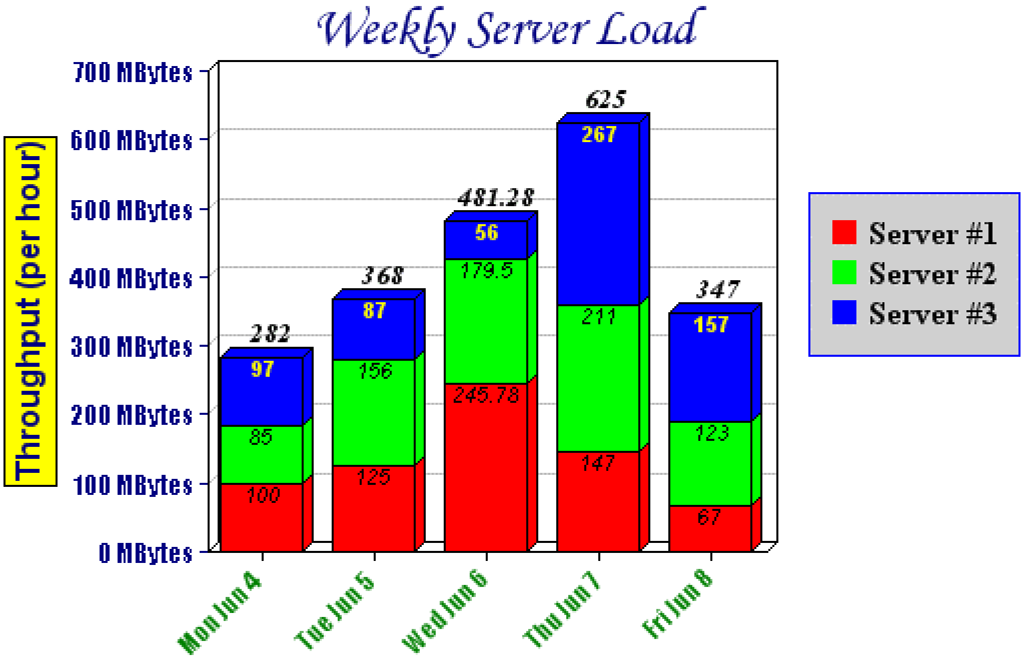
Note that although this example uses bar chart for illustration, the features introduced applies to all other XY chart types.



(The following project is available in the “cppdemo\background” subdirectory.)

```
#include "chartdir.h"

void background(const char *img, const char *filename)
{
   //data for chart
   double data[] = {85, 156, 179.5, 211, 123};
   const char *labels[] = {"Mon", "Tue", "Wed", "Thu", "Fri"};
   //create a chart object of 300(w) x 300(h)
   XYChart *c = XYChart::create(300, 300);

   if (img)
      //has wallpaper image, use it as background
      c->setWallpaper(img);
   else
      //no wallpaper image, use a dark background palette
      c->setColors(whiteOnBlackPalette);

   //set the plot area to start at (50, 50) and of size 200(w) x 200(h)
   c->setPlotArea(50, 50, 200, 200);

   //Set the labels on the x axis using the supplied labels
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add a 3D bar layer using the supplied data
   c->addBarLayer(sizeof(data) / sizeof(data[0]), data, 0x00ff00)->set3D();

   //output the chart
   c->makeChart(filename);

   //destroy the chart to free up resources
   c->destroy();
}


int main(int argc, char *argv[])
{
   background("tile.gif", "imgbg.png");
   background(0, "darkbg.png");
   return 0;
}
```

Text Style and Colors



This example illustrates you could control the fonts, colors, rotation angles, add a background box, etc., around 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. The y-axis title and the legend are surrounding by a box with a customize background color and border color, and the y-axis labels are appended with "Mbytes" as the unit. The x-axis labels are rotated 45 degrees.

These effects are achieved by using the following ChartDirector API:

- The title font and color are specified using the addTitle method.
- The legend box font is specified using the addLegend method. The legend box background and border colors are specified using the setBackground method of the LegendBox object. (The LegendBox object is returned by the addLegend method.)
- The y-axis title font and text color are specified using the setTitle method of the YAxis object. The background box of the title and its colors are specified using the setBackground method of the TextBox object returned when using the setTitle method.
- The y-axis label font and color are specified using the setLabelStyle method of the YAxis object. The y-axis label format is specified using the setLabelFormat method of the YAxis object.
- The x-axis label font and color are specified using the setLabelStyle method of the XAxis object. The x-axis label rotation angle is specified using the setFontAngle method of the TextBox object returned when using the setLabelStyle method.

- The default font of the bar label within each bar segment is specified using the setDataLabelStyle of the Layer object.
- The default font of the aggregate bar label (that is, the label on top of the stacked bar) is specified using the setAggregateLabelStyle of the Layer object.
- The font of one of the bar segment labels (the ones inside the blue segment) is specified using the setDataLabelStyle method of the DataSet object associated with that segment. This setting overrides the default bar segment label font.

(The following project is available in the “cppdemo\fontxy” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
   //data for the chart
   double data0[] = {100, 125, 245.78, 147, 67};
   double data1[] = {85, 156, 179.5, 211, 123};
   double data2[] = {97, 87, 56, 267, 157};
   const char *labels[] = {"Mon Jun 4", "Tue Jun 5", "Wed Jun 6", "Thu Jun 7",
      "Fri Jun 8"};

   //First, create a XYChart of size 540 pixels x 350 pixels
   XYChart *c = XYChart::create(540, 350);

   //Set the plot area to start at (120, 40) and of size 280(w) x 240(h)
   c->setPlotArea(120, 40, 280, 240);

   //Add a title to the chart using 20 point Monotype Corsiva (mtcorsva.ttf)
   //font and using a deep blue color (0x000080)
   c->addTitle("Weekly Server Load", "mtcorsva.ttf", 20, 0x000080);

   //Add a legend box at (420, 100) using 12 point Times New Roman Bold
   //(timesbd.ttf) font. Sets the background of the legend box to light grey
   //color (0xd0d0d0) and the border to blue color (0x0000ff)
   c->addLegend(420, 100, true, "timesbd.ttf", 12)
      ->setBackground(0xd0d0d0, 0x0000ff);

   //Add a title to the y-axis using 12 point Arial Bold (arialbd.ttf)
   //font and using a deep blue color (0x000080). Sets the background box
   //of the title to yellow (0xffff00) and the border to black (0x0)
   c->yAxis()->setTitle("Throughput (per hour)", "arialbd.ttf", 12, 0x000080)
      ->setBackground(0xffff00, 0);

   //Use 10 point Impact (impact.ttf) font as the y-axis label font and
   //deep blue (0x000080) as the font color
   c->yAxis()->setLabelStyle("impact.ttf", 10, 0x000080);

   //Set the axis label format to append "MBytes" to the numeric y value
   c->yAxis()->setLabelFormat("&value& MBytes");

   //Use 10 point Impact (impact.ttf) font as the x-axis label font and
   //deep green (0x008000) as the font color. Sets the label angle to 45 deg.
   c->xAxis()->setLabelStyle("impact.ttf", 10, 0x008000)->setFontAngle(45);
```

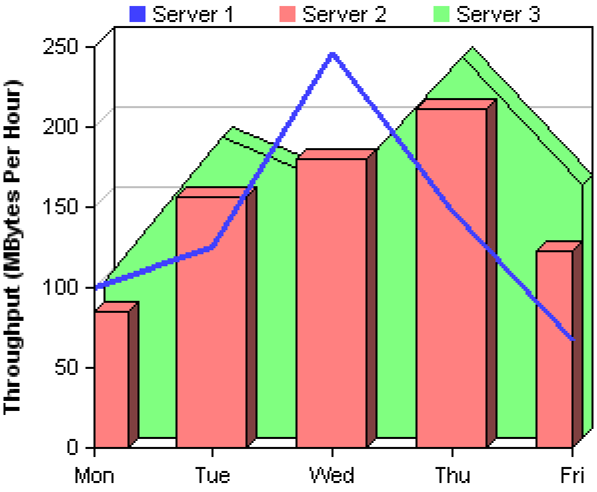
```
    //Set the x axis labels using the supplied labels
    c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

    //Add a stack bar layer
    BarLayer *layer = c->addBarLayer(Layer::Stack, 5);

    //Use Arial Italic (ariali.ttf) as the default data label font in the bar
    layer->setDataLabelStyle("ariali.ttf");

    //Use 10 point Times Bold Italic (timesbi.ttf) as the aggregate label font
    layer->setAggregateLabelStyle("timesbi.ttf", 10);

    //Add the three data sets using the supplied data. For the last data set,
    //set the data label font to Arial Bold (arialbd.ttf) with yellow color
    //to override the default data label font.
    layer->addDataSet(sizeof(data0) / sizeof(data0[0]), data0, -1, "Server #1");
    layer->addDataSet(sizeof(data1) / sizeof(data1[0]), data1, -1, "Server #2");
    layer->addDataSet(sizeof(data2) / sizeof(data2[0]), data2, -1, "Server #3")
       ->setDataLabelStyle("arialbd.ttf")->setFontColor(0xffff00);

    //output the chart as a PNG file
    c->makeChart("fontxy.png");

    //destroy the chart to free up resources
    c->destroy();

    return 0;
}
```

Legend Positioning

With the ChartDirector API, you can position you legend box in any arbitrary location in your chart.

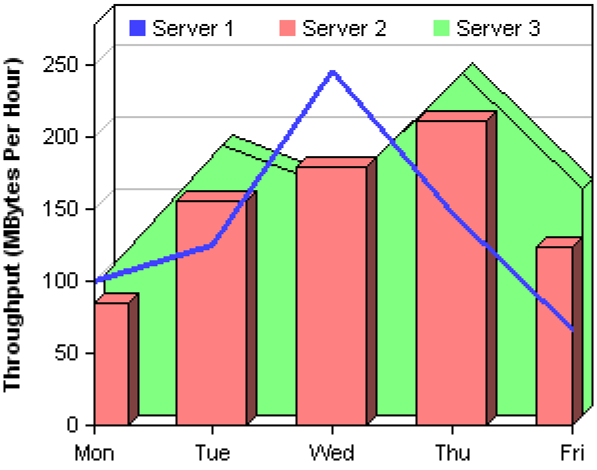
In previous examples, the legend box is usually located on the right of the chart and the legend keys are layout vertically. This example illustrates three common alternative legend locations with horizontal legend keys layout.

The first chart on the right illustrates how to position the legend keys on the top of the chart using the addLegend method.

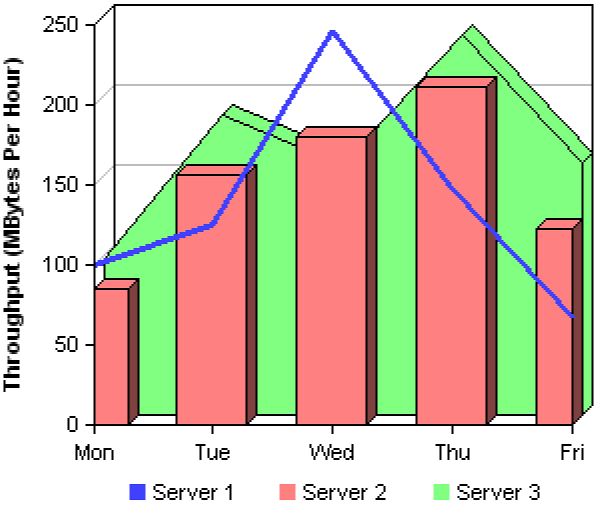


The second chart on the right also positions the legend keys on the top of the chart using the addLegend method, but the legend keys are located inside the plot area. The setTopMargin method of the YAxis object is used to ensure there is room for the legend keys inside the plot area. Note that on the second chart, there are some extra spaces at the top of the y axis.

In additional to the setTopMargin method, the setAutoScale method can also be used to leave some spaces on the top and bottom of the y axis (and hence the plot area). Please refer to the following sections on Y-Axis Scaling for details.



The third chart on the right illustrates how you can position the legend box on the bottom of the chart using the addLegend method.



Note also in this example, the first and last bars in the bar chart are only drawn in half. In previous examples, all charts are drawn in full. It is because in this example, the x-axis is layout using "non-indented" mode.

In "non-indented" x-axis, the first data point is at the start of of the x-axis, and the last data point is at the end of the x-axis. Non-indented x-axis is the default for all XY chart types except the bar chart.

For bar charts, the "indented" x-axis is the default. In this layout method, the first data point is shifted right, while the last data point is shifted left. This is so that all bars can be drawn in full. If a chart contains both bar chart layers and other XY chart layers, the default is to use indented x-axis. This can be changed by using the setIndent method of the XAxis object, as demonstrated in this example.

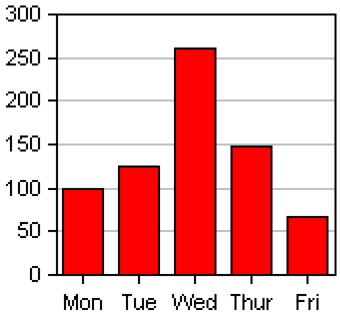
(The following project is available in the "cppdemo\legendpos" subdirectory.)

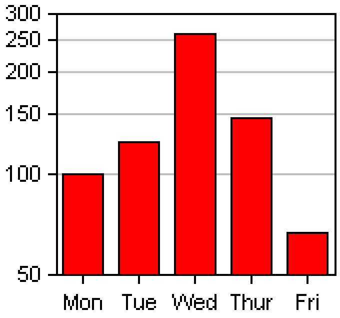
```
#include "chartdir.h"

void legendpos(int legendPos, const char *filename)
{
   //The data for the chart
   double data0[] = {100, 125, 245.78, 147, 67};
   double data1[] = {85, 156, 179.5, 211, 123};
   double data2[] = {97, 187, 156, 237, 157};
   const char *labels[] = {"Mon", "Tue", "Wed", "Thu", "Fri"};

   //Create a XYChart object of size 300 x 300
   XYChart *c = XYChart::create(300, 300);

   //Set the plot area at (50, 40) and of size 240 x 200
   c->setPlotArea(50, 40, 240, 200);

   switch (legendPos)
   {
   case 0 :
      //add legends on the top of the chart (60, 10) using horizontal layout.
      //set the font to 8 points, and hide the legend box boundary (Transparent)
      c->addLegend(60, 10, false, 0, 8)->setBackground(Transparent, Transparent);
      break;
   case 1 :
      //add legends on the top of the plot area (60, 28) using horizontal layout.
      //set the font to 8 points, and hide the legend box boundary (Transparent)
      c->addLegend(60, 28, false, 0, 8)->setBackground(Transparent, Transparent);

      //reserve 20 pixels at the top of the plot area to ensure the legend keys
      //will not overlap with the charts
      c->yAxis()->setTopMargin(20);
      break;
   default :
      //add legends on the bottom of the chart (60, 260) using horizontal layout.
      //set the font to 8 points, and hide the legend box boundary (Transparent)
      c->addLegend(60, 260, false, 0, 8)->setBackground(Transparent, Transparent);
   }

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

   //Set the labels on the x axis
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Use non-indented x-axis layout
   c->xAxis()->setIndent(false);

   //Add 3 differnet layers using the given data sets
   c->addLineLayer(sizeof(data0) / sizeof(data0[0]), data0, 0x4040ff, "Server 1")
      ->setLineWidth(3);
   c->addBarLayer(sizeof(data1) / sizeof(data1[0]), data1, 0xff8080, "Server 2")
      ->set3D();
   c->addAreaLayer(sizeof(data2) / sizeof(data2[0]), data2, 0x80ff80, "Server 3")
      ->set3D();

   //output the chart
   c->makeChart(filename);
   //destroy the chart to free up resources
   c->destroy();
}

int main(int argc, char *argv[])
{
   legendpos(0, "legendpos0.png");
   legendpos(1, "legendpos1.png");
   legendpos(2, "legendpos2.png");
   return 0;
}
```

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.

Note that although this example uses bar chart for illustration, the features introduced applies to all other XY chart types.





(The following project is available in the “cppdemo\logaxis” subdirectory.)

```
#include "chartdir.h"

void logaxis(bool logaxis, const char *filename)
{
   //data for the chart
   double data[] = {100, 125, 260, 147, 67};
   const char *labels[] = {"Mon", "Tue", "Wed", "Thur", "Fri"};

   //create a XYChart object of size 200 x 180
   XYChart *c = XYChart::create(200, 180);

   //set the plot area to start at (30, 10) and of size 140 x 130
   c->setPlotArea(30, 10, 140, 130);

   //use log scale axis if required
   if (logaxis)
      c->yAxis()->setLogScale();

   //Set the labels on the x axis
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //Add a bar layer using the supplied data
   c->addBarLayer(sizeof(data) / sizeof(data[0]), data);

   //output the chart as a PNG file
   c->makeChart(filename);

   //destroy the chart to free up resources
   c->destroy();
}

int main(int argc, char *argv[])
{
   logaxis(false, "linearaxis.png");
   logaxis(true, "logaxis.png");
   return 0;
}
```

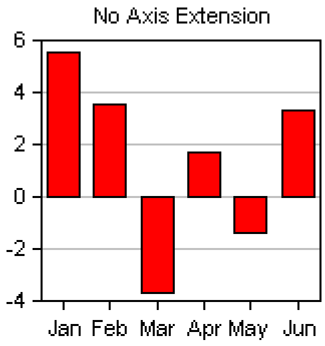
Y-Axis Scaling

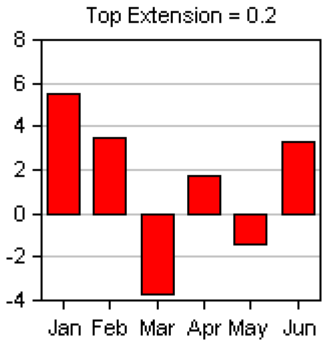
This example illustrates how you could control the scaling of the y-axis.

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 standing auto-scaling.

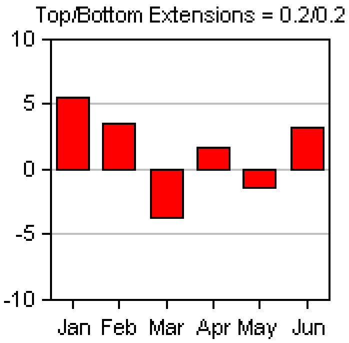
In many cases, it is desirable to leave some spaces at the top and/or bottom of the plot area. For example, you may want to add some custom text or legend box at those locations, or you may simply think it looks better this way. In these cases, the setAutoScale method can be used to inform the auto-scaling 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).



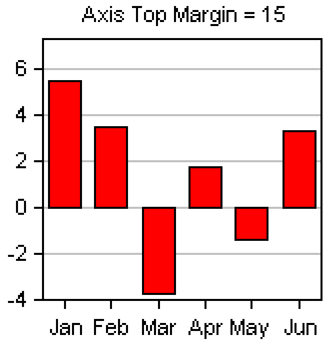


The second chart illustrates what the chart looks like when 20% of the spaces on the top are reserved by using setAutoScale(0.2).

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).

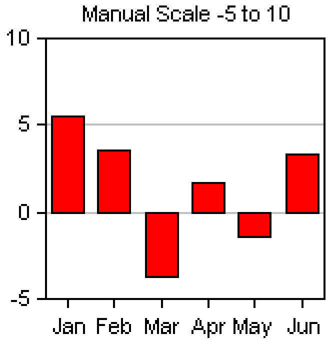


An alternative way to reserve space on the top of the plot area is to use the setTopMargin method. With this method, a segment on the top of the y-axis is excluded from scaling. There are no ticks and no scaling there. The chart on the right illustrates this feature. Note that the top of the y-axis has no scaling there.



Although auto-scaling is convenient, in many cases manual scaling may be more preferable. For example, in many percentage charts, you many 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.

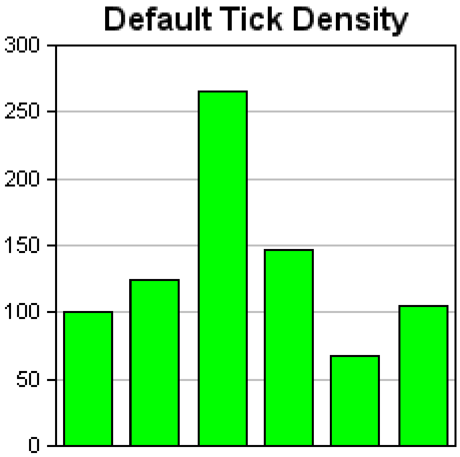


Note that although this example uses bar chart for illustration, the features introduced applies to all other XY chart types.

(The following project is available in the “cppdemo\axisscale” subdirectory.)

```
#include "chartdir.h"

void axisscale(int axisstyle, const char *filename)
{
   double data[] = {5.5, 3.5, -3.7, 1.7, -1.4, 3.3};
   const char *labels[] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun"};
   XYChart *c = XYChart::create(200, 180);
   c->setPlotArea(30, 20, 140, 130);

   switch (axisstyle)
   {
   case 0 :
      c->addTitle("No Axis Extension", "arial.ttf", 8);
      break;
   case 1 :
      c->addTitle("Top Extension = 0.2", "arial.ttf", 8);
      c->yAxis()->setAutoScale(0.2);
      break;
   case 2 :
      c->addTitle("Top/Bottom Extensions = 0.2/0.2", "arial.ttf", 8);
      c->yAxis()->setAutoScale(0.2, 0.2);
      break;
   case 3 :
      c->addTitle("Axis Top Margin = 15", "arial.ttf", 8);
      c->yAxis()->setTopMargin(15);
      break;
   default :
      c->addTitle("Manual Scale -5 to 10", "arial.ttf", 8);
      c->yAxis()->setLinearScale(-5, 10, 5);
   }

   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);
   c->addBarLayer(sizeof(data) / sizeof(data[0]), data);
   c->makeChart(filename);
   c->destroy();
}

int main(int argc, char *argv[])
{
   axisscale(0, "noextaxis.png");
   axisscale(1, "topextaxis.png");
   axisscale(2, "bothextaxis.png");
   axisscale(3, "marginaxis.png");
   axisscale(4, "manualaxis.png");
   return 0;
}
```

Tick Density

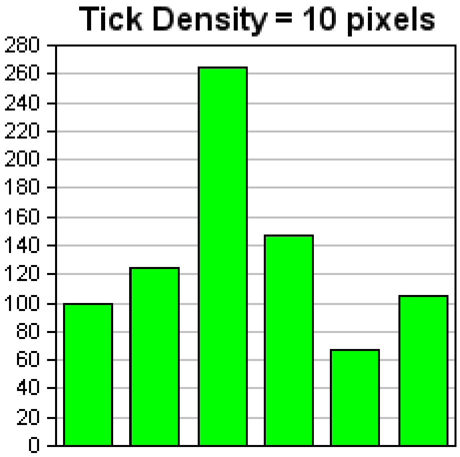


This example illustrates how to control the axis tick density during auto-scaling.

In manual scaling, you can directly control the axis tick density using the setLinearScale or setLogScale(2) 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 the requested one because the auto-scaling algorithm also needs to ensure that the ticks are while numbers, and the axis contains an integral number of ticks, etc..



The first chart in this example illustrates the default tick density settings. The second chart requested a higher density ticks by setting the tick spacing to 10 pixels. Note that the actual tick spacing is slightly larger than 10 pixels.

Also note that the scaling of the axis has changed as a result of the tick density. In the first char 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.

Note that although this example uses bar chart for illustration, the features introduced applies to all other XY chart types.

(The following project is available in the "cppdemo\ticks" subdirectory.)

```
#include "chartdir.h"

void ticks(bool denseticks, const char *filename)
{
   //data for the chart
   double data[] = {100, 125, 265, 147, 67, 105};

   //create a XYChart object of size 250 x 250
   XYChart *c = XYChart::create(250, 250);

   //set the plot area at (25, 25) and of size 200 x 200
   c->setPlotArea(25, 25, 200, 200);
   if (denseticks)
   {
      //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 bar layer to the chart using the given data
   c->addBarLayer(sizeof(data) / sizeof(data[0]), data, 0x00ff00);

   //output the chart
   c->makeChart(filename);

   //destroy the chart to free up resources
   c->destroy();
}

int main(int argc, char *argv[])
{
   ticks(false, "normalticks.png");
   ticks(true, "denseticks.png");
   return 0;
}
```

Dual Y-Axis

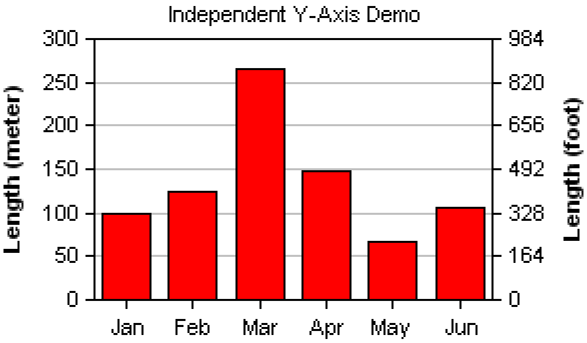
This example show demonstrates the dual y-axis feature of the ChartDirector. There are two types of dual y-axis that the ChartDirector supports – synchronous y-axes and independent y-axes.

In synchronous y-axes, the two y-axes are related by a linear relationship of the form:

$$y2 = m * y1 + c.$$

For example, if one axis represents length in meters, and the other axis represents length in feet, then they are synchronous y-axes.

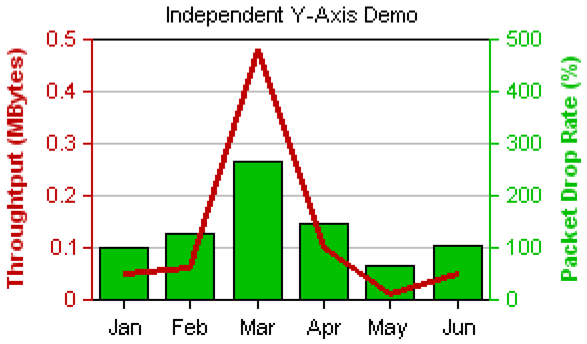
The example on the right demonstrates how to set synchronous y-axes using the syncYAxis method.



In independent y-axes, the two y-axes are independent. Usually this is used when you have two data sets with different units.

For example, if you have two data sets, one being the network throughput in mega bytes, and the other being the packet drop rate in %, they can be represented as two independent y-axes.

The example on the right demonstrates independent y-axes.



By default, all data sets used the primary (left) y-axis. The setUserYAxis2 can be used to bind a data set to the secondary y-axis (right).

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:

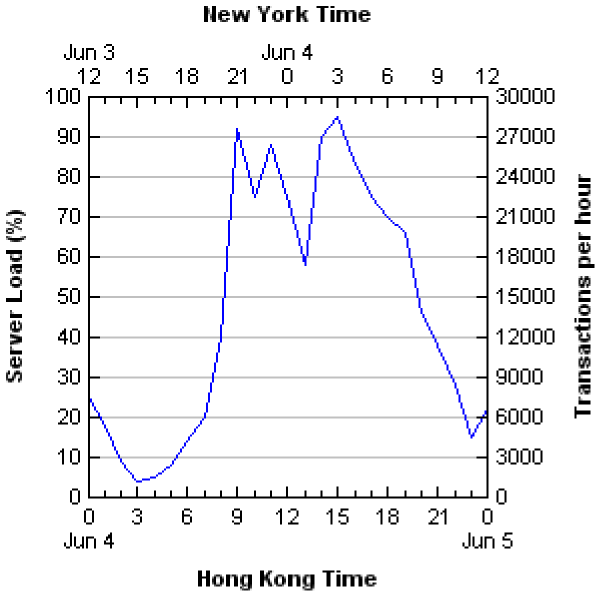
- you can bind all data sets to the secondary y-axis
- you can use default binding (that is, all data sets bind to the primary y-axis), and then use the setYAxisOnRight to draw the primary y-axis on the right side (and therefore the secondary axis, if used, will be drawn on the left side).

(The following project is available in the “cppdemo\dualyaxis” subdirectory.)

```
#include "chartdir.h"

/////////////////////////////////////////////////////////////////////////////
//
// This function demonstrates a chart with two synchronous y-axis.
//
/////////////////////////////////////////////////////////////////////////////
void syncaxis()
{
   //data for the chart
   double data[] = {100, 125, 265, 147, 67, 105};
   const char *labels[] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun"};

   //create a XYChart object of size 300(w) x 180(h)
   XYChart *c = XYChart::create(300, 180);

   //set the plot area at (50, 20) and of size 200(w) x 130(h)
   c->setPlotArea(50, 20, 200, 130);

   //add a title to the chart using 8 point Arial font (arial.ttf)
   c->addTitle("Independent Y-Axis Demo", "arial.ttf", 8);

   //Set the x axis labels using the supplied labels
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);
   //add a title to the primary (left) y-axis
   c->yAxis()->setTitle("Length (meter)");

   //add a title to the secondary (right) y-axis
   c->yAxis2()->setTitle("Length (foot)");

   //set the two axis so that y2 = 3.28 x y1  (1 meter = 3.28 feet)
   c->syncYAxis(3.28);

   //add a bar layer to represent the data
   c->addBarLayer(sizeof(data) / sizeof(data[0]), data);

   //output the chart as a PNG file
   c->makeChart("syncyaxis.png");

   //destroy the chart to free up resources
   c->destroy();
}

/////////////////////////////////////////////////////////////////////////////
//
// This function demonstrates a chart with two independent y-axis.
//
/////////////////////////////////////////////////////////////////////////////
void dualaxis()
{
   //data for the chart
   double data0[] = {0.05, 0.06, 0.48, 0.1, 0.01, 0.05};
   double data1[] = {100, 125, 265, 147, 67, 105};
   const char *labels[] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun"};

   //create a XYChart object of size 300(w) x 180(h)
   XYChart *c = XYChart::create(300, 180);

   //set the plot area at (50, 20) and of size 200(w) x 130(h)
   c->setPlotArea(50, 20, 200, 130);

   //add a title to the chart using 8 point Arial font (arial.ttf)
   c->addTitle("Independent Y-Axis Demo", "arial.ttf", 8);

   //Set the x axis labels using the supplied labels
   c->xAxis()->setLabels(sizeof(labels) / sizeof(labels[0]), labels);

   //add a title to the primary (left) y-axis
   c->yAxis()->setTitle("Throughtput (MBytes)");

   //set the axis, label and title colors for the primary y axis to red
   //to represent the first data set
   c->yAxis()->setColors(0xC00000, 0xC00000, 0xC00000);

   //add a title to the secondary (right) y-axis
   c->yAxis2()->setTitle("Packet Drop Rate (%)");

   //set the axis, label and title colors for the primary y axis to green
   //to represent the second data set
   c->yAxis2()->setColors(0x00C000, 0x00C000, 0x00C000);
   //add a line layer to represent the first data set using red color and
   //set the line width to 3 pixels
   c->addLineLayer(sizeof(data0) / sizeof(data0[0]), data0, 0xC00000)
      ->setLineWidth(3);

   //add a bar layer to represent the second data set using green color and
   //bind the data set to the secondary (right) y-axis
   c->addBarLayer()->addDataSet(sizeof(data1) / sizeof(data1[0]), data1, 0x00C000)
      ->setUseYAxis2();

   //output the chart as a PNG file
   c->makeChart("dualyaxis.png");

   //destroy the chart to free up resources
   c->destroy();
}

int main(int argc, char *argv[])
{
   syncaxis();
   dualaxis();
   return 0;
}
```

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.
- Use the setTickLength(2) method to control the major and minor tick length, and their direction (internal to the chart, or external to the chart).
- Use the “-” for an x-axis label to represent a minor tick.

(The following project is available in the “cppdemo\dualxaxis” subdirectory.)

```
#include "chartdir.h"

int main(int argc, char *argv[])
{
```

```
//The data for the chart
double data[] = {25, 18, 9, 4, 5, 8, 14, 20, 40, 92, 75, 88, 75,
   58, 90, 95, 83, 75, 70, 66, 46, 38, 28, 15, 22};

//The labels for the bottom x axis. Note the "-" means a minor tick.
const char *label0[] = {"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.
const char *label1[] = {"Jun 3\n12", "-", "-", "15", "-", "-", "18",
   "-", "-", "21", "-", "-", "Jun 4\n0", "-", "-", "3", "-", "-", "6",
   "-", "-", "9", "-", "-", "12"};

//First, create a XYChart of size 300 pixels x 320 pixels
XYChart *c = XYChart::create(320, 320);

//Set the plotarea at (60, 50) and of 200 pixel (w) x 200 pixels (h)
c->setPlotArea(60, 50, 200, 200);

//Add a title to the primary (left) y-axis
c->yAxis()->setTitle("Server Load (%)");

//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
c->yAxis2()->setTitle("Transactions per hour");

//Set the tick length to -4 pixels (-ve means ticks inside the plot area)
c->yAxis2()->setTickLength(-4);

//Set y2 = 300 x y1 (that means 1% load = 300 transactions per hour)
c->syncYAxis(300);

//Add a title to the bottom x-axis
c->xAxis()->setTitle("Hong Kong Time");

//Set the x axis labels using the supplied labels
c->xAxis()->setLabels(sizeof(label0) / sizeof(label0[0]), 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
c->xAxis2()->setTitle("New York Time");

//Set the x axis labels using the supplied labels
c->xAxis2()->setLabels(sizeof(label1) / sizeof(label1[0]), label1);

//Set the major tick length to -4 pixels and minor tick length to -2 pixels
//(-ve means ticks inside the plot area)
c->xAxis2()->setTickLength(-4, -2);

c->xAxis2()->setColors(0, 0);
```

```
   //Add an line layer to the chart
   c->addLineLayer(sizeof(data) / sizeof(data[0]), data, 0x0000ff);

   //output the chart as a PNG file
   c->makeChart("dualxaxis.png");

   //destroy the chart to free up resources
   c->destroy();

   return 0;
}
```

6

ChartDirector API Reference

Data Types

Color Specification

Color is specified as a 4-bytes number (that is, 32 bit number) in ARGB format.

The ARGB refers to Alpha transparency, Red, Green and Blue components of the color. Each component occupies 8 bits ranging from 0 – 255, representing the intensity of the color. The blue component occupies the least significant 8 bits, the green component occupies the next 8 bits, and the red components occupies the further next 8 bits, and the Alpha transparency occupies the most significant 8 bits.

For example, the pure red color is 0x00ff0000, the pure green color is 0x0000ff00, and the pure blue color is 0x000000ff. The 0x00ffffff is the white color, while the 0x00000000 is the black color.

The most significantly 8 bits are used for Alpha transparency. An Alpha transparency of 0 means the color is not transparent at all (fully opaque), and 255 means the color is totally transparent. For example, 0x80ff0000 is a partially transparent red color, while 0x00ff0000 is a fully opaque red color.

If a color is totally transparent, anything drawn will be invisible no matter what the RGB components are. That means all totally transparent colors are the same. Therefore in ChartDirector, only one legal total transparent color is used – 0xff000000. All other colors of the form 0xffnnnnnn are reserved for “palette colors” and other special usage, and should not be interpreted as the normal ARGB colors.

Colors of the format 0xffffnnnn are “palette colors”, 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 color table using the index. For example, the color 0xffff0001 is the second color in the palette color table (first color is index 0).

All charts are created with a default palette color table. You may modify the color tables using the setColor, setColors, or setColors(2) methods of the BaseChart object.

The first three palette colors have special significance. They are the background color, default line color, and default text color of the chart.

The 9th color (index = 8) onwards are used as the default colors for drawing the data sets. The 9th color is the default color for the 1st data set, the 10th color is for the 2nd data set, etc.

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.

The ChartDirector API pre-defines several constants for to facilitate using the color tables.

```
enum
{
   Transparent = 0xff000000,
   Palette = 0xffff0000,
   BackgroundColor = 0xffff0000,
   LineColor = 0xffff0001,
   TextColor = 0xffff0002,
   DataColor = 0xffff0008
};
```

The example code Coloring Scheme and Wallpaper illustrates the various ChartDirector palettes and how to modify the palette color table.

Font Specification

Font File

In the ChartDirector API, a font is specified by specifying the file name that contains the font. The ChartDirector does not come with any font files. However, the operating system probably already contains a lot of font files.

The ChartDirector will try to locate the font file using the font file name or path name. If it cannot find the font file, in windows platform, the ChartDirector will try to locate the font file in the “\windows\Fonts” subdirectory (where “\windows” is the directory where the operating system is installed).

In Windows, the font files are located in the “\windows\Fonts” subdirectory (where “\windows” is assumed to be the operating system directory). For example, the “Arial” font is in the file “arial.ttf” under that subdirectory, and the “Arial Bold” font is in the file “arialbd.ttf”. To see what fonts are installed, simply use the File Explorer to view that subdirectory.

If you want the ChartDirector to search other subdirectories for the font file, you may define an environment variable called “FONTPATH” and list the subdirectories you want ChartDirector to search. The subdirectories should be separated using the semi-color ‘;’ as the delimiter.

Besides the font files in the operating system, there are a lot of web sites that contains font files that you may download.

Font Index

In theory, a single font file may contain mutilple fonts. 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.

In practice, most font files contain only one font. Therefore in most cases, you can just use the default font index value of 0.

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 TextStyle 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.

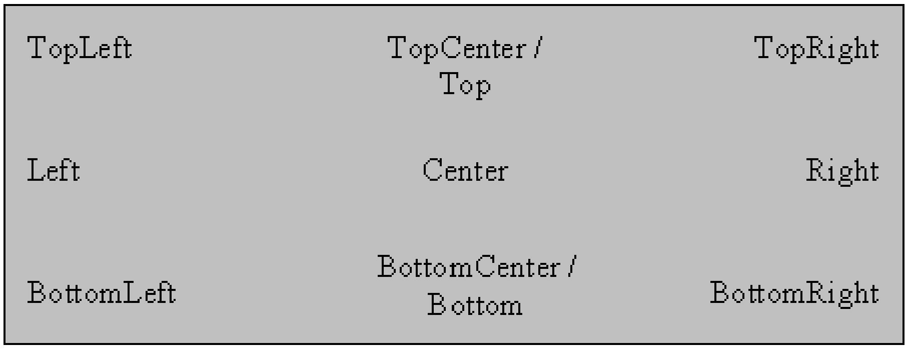
In horizontal layout, each additional character will be drawn on the right of the previous character. After all characters are drawn, the whole text will be rotated according to the font angle.

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.

```
enum Alignment
{
   TopLeft = 7,          TopCenter = 8,          TopRight = 9,
   Left = 4,             Center = 5,             Right = 6,
   BottomLeft = 1,       BottomCenter = 2,       BottomRight = 3,
   Top = TopCenter,
   Bottom = BottomCenter
};
```

No Value Specification

In many ChartDirector charts, you need to supply data to plot the charts. The data are typically supplied as an array of double precision floating point numbers that represents the value of the data points.

ChartDirector supports a special constant called “NoValue”. You may use this constant to specify that a data point contains no data.

For example, if you want to draw a trending line for certain statistics from 0:00 to 23:59, but your data set only contains data up to 12:00, then you can set the remaining data to be NoValue.

Another example is that if you have a trending line that represents the closing stock prices for each day of a month, there may be no data point on Saturdays and Sundays when the stock market is closed. In this case, you may specify the data point on those days to be NoValue.

If a data point is NoValue, nothing will be drawn on the chart to represent the data point. In the first example above, the line will only be drawn from 0:00 to 12:00. In the second example, the line will be broken in segments, where each segment represents one week.

Although the above example is based on line charts, the NoValue special constant is also applicable to other XY chart types.

Draw Objects

DrawObj

This is the base class of all drawable objects in ChartDirector. You never need to call any of the methods in this class. ChartDirector will call them automatically. This class is documented here for developers who want to implement their own drawable objects. In this case, the developers need to implement the DrawObj methods for their drawable objects.

Method	Description
paint	Implement this method to draw the object.
destroy	Implement this method to destroy the object.

paint

Prototype

virtual void paint(DrawArea *d) = 0;

Description

This method draws the object using the DrawArea tool. The ChartDirector will automatically call this method when it needs to draw the object. Developers who want to develop their own drawable objects should implement this method.

Arguments

Argument	Default Value	Description
d	(Mandatory)	The DrawArea tool that the implemention of this method can use to paint the drawable object. The ChartDirector will supply a suitable DrawArea tool when calling this method.

Return Value

None

destroy

Prototype

virtual void destroy() = 0;

Description

Release all resources used by the DrawObj, including the DrawObj itself. The ChartDirector will call this method when the DrawObj is no longer needed. After calling this method, the ChartDirector will not use the object any more. Note that the ChartDirector will not “delete” the object. It assumes the destroy method will delete the object itself.

Arguments

None

Return Value

None

TextStyle

The TextStyle object encapsulates the appearance of text. You may use the methods of the TextStyle object to define how the text should appear. You may also add a bounding box to contain the text. The size of the bounding box will be automatically adjusted to fit the text.

Method	Description
setFontStyle	Set the font of the text.
setFontSize	Set the font size of the text.
setFontAngle	Set the rotation angle of the text.
setFontColor	Set the color of the text.
setBackground	Set the background and border colors of the bounding box of the text.
setMargin	Set the left, right, top and bottom margins of the bounding box of the text.
setMargin(2)	Short cut to set all four bounding box margins to the same value.

setFontStyle

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 font file that contains the font.
fontIndex	0	The font index 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

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 font height in a font unit called points.
fontWidth	0	The font width 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

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 font rotation angle. Rotation is measured in counter-clockwise director in degrees.
vertical	false	Determine whether the font is layout 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 means vertical layout, while a “false” value means horizontal layout.

Return Value

None

setFontColor

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 font color.

Return Value

None

setBackground

Prototype

virtual void setBackground(int color, int edgeColor = Transparent) = 0;

Description

Set the background color and edge color of the bounding box. By default, both colors are transparent, which means the bounding box is invisible. See Color Specification for how colors are represented in ChartDirector.

Arguments

Argument	Default Value	Description
color	(Mandatory)	The background color of the bounding box.
edgeColor	Transparent	The border color of the bounding box.

Return Value

None

setMargin

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

setMargin(2)

Prototype

virtual void setMargin(int m) = 0;

Description

A short cut for setMargin. 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

Box

The class Box, as it 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
setPos	Set the coordinates of the top left corner of the box.
setSize	Set the width and height of the box.
setBackground	Set the background color and edge color of the box.

setPos

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

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

Prototype

virtual void setBackground(int color, int edgeColor = Transparent) = 0;

Description

Set the background color and edge color of the box. By default, both colors are transparent, which means the box is invisible. See Color Specification for how colors are represented in ChartDirector.

Arguments

Argument	Default Value	Description
color	(Mandatory)	The background color of the box.
edgeColor	Transparent	The border color of the box.

Return Value

None

TextBox

Inherit from TextStyle and Box.

The class TextBox, as its name implies, represents a text box. The TextBox class inherits from the TextStyle class and the Box class.

If the width of the text box is set to 0, the width will automatically be adjusted to fit the text. Similarly, if the height is set to 0, the height will automatically be adjusted to fit the text. So if both the width and height are 0, the TextBox object will behave like a TextStyle object.

Method	Description
setText	Sets the text to be shown in the text box.
setAlignment	Sets the alignment of the text relative to the container box
Methods inherited from TextStyle	

setFontStyle	Set the font of the text.
setFontSize	Set the font size of the text.
setFontAngle	Set the rotation angle of the text.
setFontColor	Set the color of the text.
setMargin	Set the left, right, top and bottom margins of the bounding box of the text.
setMargin(2)	Short cut to set all four bounding box margins to the same value.
Methods inherited from Box	
setPos	Set the coordinates of the top left corner of the box.
setSize	Set the width and height of the box.
setBackground	Set the background color and edge color of the box.

setText

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

setAlignment

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 Alignment Specification for possible alignment types.

Return Value

None

Line

A Line, as its name implies, represents a straight line.

Method	Description
setPos	Set the end points (x1, y1) and (x2, y2) of the line.
setColor	Set the color of the line.
setWidth	Set the width of the line in pixels.

setPos

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

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

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
destroy	Destroy the chart object.
setSize	Set the size of the chart to the specified width and height in pixels.
setBorder	Set the border color of the chart.
setWallpaper	Specify an image as the background wallpaper of the chart.
setBgImage	Specify an image as the background image of the chart.
addTitle	Add a title to the chart on the TopCenter position of the chart
addTitle(2)	Add a title to the chart at the top, bottom, left or right position of the chart.
addLegend	Add a legend box to the chart.
getDrawArea	Returns the DrawArea object that the chart is drawn with to allow drawing custom text, line or shapes.
addDrawObj	Add a custom-developed DrawObj to the chart.
addText	Add a text box to the chart.
addLine	Add a line to the chart.
setColor	Change the color of the specified position in the palette color table.
setColors	Change the colors of the color table starting with the specified position in the palette color table.
setColors(2)	Change the colors of the color table starting from the first color.

getColor	Get the color of the specified position in the palette color table.
layout	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.
makeChart	Generate the chart and save it into a file.
makeChart(2)	Generate the chart in memory.

destroy

Prototype

virtual void destroy() = 0;

Description

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

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

setBorder

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 invible.

Arguments

Argument	Default Value	Description
color	(Mandatory)	The border color of the chart.

Return Value

None

setWallpaper

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

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

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 Alignment Specification for possible alignment types.

Return Value

None

addTitle

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 addTitle(2) 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 pointer to a TextBox object representing the title to allow fine-tuning of the title appearance.

addTitle(2)

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 the 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 Alignment Specification 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

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 the 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	true	Determines whether the keys inside the legend box are layout vertically or horizontally. A “true” value means the keys are layout vertically. A “false” value means the keys are layout horizontally.
font	0	The font file for the legend font. See Font Specification 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

Prototype

virtual DrawArea *getDrawArea() = 0;

Description

Returns 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 so that you may use this tool to add custom drawings to the chart or even to develop your own custom chart type.

Arguments

None.

Return Value

This method returns the DrawArea object that can be used to add custom text and shapes to the chart.

addDrawObj

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

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	0	The font used to draw the text. A null pointer '0' means using the default font (Arial). See Font Specification 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	false	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

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

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

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)	An array of colors to change to. The last color must be –1 to indicate the end of the colors array.

Return Value

None

setColors(2)

Prototype

virtual void setColors(const int *colors) = 0;

Description

Change the colors of the color table starting from the first color. This method is typically used to change the entire color table. The ChartDirector comes with several predefined color tables that you may use. (See the section on Color Specification on the details of the color table.)

Arguments

Argument	Default Value	Description
colors	(Mandatory)	An array of colors to change to. The last color must be –1 to indicate the end of the colors array.

Return Value

None

getColor

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.

layout

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

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 if 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 makeChart(2) method instead.

Arguments

Argument	Default Value	Description

filename	(Mandatory)	The name of the file to save the image.

Return Value

Returns true if no error, otherwise returns false.

makeChart(2)

Prototype

virtual bool makeChart(ImgFormat format, const char **data, int *len) = 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: • BaseChart::PNG • BaseChart::GIF • BaseChart::JPG • BaseChart::WMP
data	(Mandatory)	The address of a character pointer to receive the address of the memory block that holds the WAP bitmap image. The memory block is allocated by the BaseChart, and will automatically free when the BaseChart is destroy. There is no need to explicitly free the memory block.
len	(Mandatory)	The address of an integer variable to receive the length of the memory block.

Return Value

Returns true if no error, otherwise returns false.

LegendBox

The class LegendBox represents a legend box. It is a subclass of TextBox, which in turn is a subclass of TextStyle and 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 the 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
addKey	Add an additional entry to the legend box.
Methods inherited from TextBox	
setText	Sets the text to be shown in the text box.
setAlignment	Sets the alignment of the text relative to the container box
Methods inherited from TextStyle	
setFontStyle	Set the font of the text.
setFontSize	Set the font size of the text.
setFontAngle	Set the rotation angle of the text.
setFontColor	Set the color of the text.
setMargin	Set the left, right, top and bottom margins of the bounding box of the text.
setMargin(2)	Short cut to set all four bounding box margins to the same value.
Methods inherited from Box	
setPos	Set the coordinates of the top left corner of the box.
setSize	Set the width and height of the box.
setBackground	Set the background color and edge color of the box.

addKey

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

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.

Because the PieChart is an abstract class, you cannot create it directly. Instead, you must use the PieChart::create method to create a PieChart object. After you have finished using the PieChart object, you must use the destroy method to free the PieChart object.

Method	Description
create	Create a PieChart object.
setPieSize	Set the position and size of the pie within the pie chart.
set3D	Add 3D effects to the pie.
setStartAngle	Set the angle of the first sector in the pie, and the direction (clockwise or anticlockwise) to layout subsequent sectors.
setLabelFormat	Sets the format of the all sector labels.
setLabelStyle	Sets the style used to draw all sector labels.
setLabelPos	Set the location of the sector labels, and specify whether join lines are used to connect the sector labels to the sector perimeter,
setData	Sets the data used to draw the pie chart.
sector	Retrieve the Sector object representing the specified sector in the pie chart.
Methods inherited from BaseChart	
destroy	Destroy the chart object.
setSize	Set the size of the chart to the specified width and height in pixels.
setBorder	Set the border color of the chart.
setWallpaper	Specify an image as the background wallpaper of the chart.
setBgImage	Specify an image as the background image of the chart.
addTitle	Add a title to the chart on the TopCenter position of the chart
addTitle(2)	Add a title to the chart at the top, bottom, left or right position of the chart.
addLegend	Add a legend box to the chart.
getDrawArea	Returns the DrawArea object that the chart is drawn with to allow drawing custom text, line or shapes.
addDrawObj	Add a custom-developed DrawObj to the chart.
addText	Add a text box to the chart.
addLine	Add a line to the chart.
setColor	Change the color of the specified position in the palette color table.

setColors	Change the colors of the color table starting with the specified position in the palette color table.
setColors(2)	Change the colors of the color table starting from the first color.
getColor	Get the color of the specified position in the palette color table.
layout	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.
makeChart	Generate the chart and save it into a file.
makeChart(2)	Generate the chart in memory.

create

Prototype

static PieChart *create(int width, int height, int bgColor = BackgroundColor, int edgeColor = Transparent);

Description

Create a PieChart object. After finish using the PieChart object, it should be freed using the destroy method.

Arguments

Argument	Default Value	Description
width	(Mandatory)	The width of the chart in pixels.
height	(Mandatory)	The height of the chart in pixels.
bgColor	BackgroundColor	The background color of the chart.
edgeColor	Transparent	The border color of the chart.

Return Value

Return the PieChart object created.

setPieSize

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

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	false	Flag to indicate whether the pie is in standard 3D mode or shadow 3D mode. A true value means shadow 3D mode, while a false value means standard 3D mode.

Return Value

None

setStartAngle

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

startAngle	(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.
clockWise	true	A flag to control the layout direction of the sectors. A true value indicates clockwise, while a false value indicates counter-clockwise.

Return Value

None

setLabelFormat

Prototype

virtual void setLabelFormat(const char *formatString) = 0;

Description

Sets 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, the sector labels will contain two lines. The first line is the sector name, while the second line is the percentage of the sector. You may change the format of the sector labels by supplying a format string.

For example, suppose you want the sector labels to contain three lines, which display the sector name, the value of the sector, and the percentage of the sector. The format string to do that this would be "&label&\n&value&\n&percent&%". The explanation is as follows:

- &label& is a place holder for the sector name
- &value& is a place holder for the value
- &percent& is the place holder for the percentage

The new line character '\n' separates the &label&, &value& and &percent&, so they are displayed on three lines. Also, after &percent& there is a '%' character, so the percentage value will contain a '%' ending character.

As a second example, suppose you want to:

- display label on one line, and the value and percentage together on the other line
- separate the value and percentage with a space, and enclose the percentage in parenthesis
- add a '$' in front of the value, and add a 'K' after the value (that is, instead of displaying 123, you want to display $123K)

The format string to do the above would be "&label&\n$&value&K (&percent&%)".

For the &value& and &percent&, you can specify the precision, decimal point character and thousand separator character.

For example, if you want the 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 the "&value|2.,&", the '|' character means that there is further formatting options. The first formatting option is the number of decimal points, the following character is the decimal point character, and then the following character is the thousand separator.

By default, if you leave out the thousand separator character, there will be no thousand separator, so "123456" will be displayed as exactly "123456" and not "123,456". If you leave out the decimal point character, it will be '.' (dot).

Arguments

Argument	Default Value	Description
formatString	(Mandatory)	The format string. See above for description.

Return Value

None

setLabelStyle

Prototype

virtual TextBox *setLabelStyle(const char *font, double fontSize = 8, int fontColor = TextColor) = 0;

Description

Sets 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	0	The font used to draw the sector labels. A null pointer '0' means using the default font (Arial).
fontSize	8	The font size used to draw the sector labels.
fontColor	TextColor	The color used to draw the sector labels.

Return Value

A TextBox object that represents the prototype of the sector labels. You may use the methods of the TextBox object to fine tune the appearance of the sector labels.

setLabelPos

Prototype

virtual void setLabelPos(int pos, int joinLineColor = Transparent) = 0;

Description

Set the location of the sector labels, 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 join the sector perimeter with the sector label. The default value is Transparent, which means the line is not drawn. Note that join lines do not apply if the sector labels are inside the sectors.

Return Value

None

setData

Prototype

virtual void setData(int noOfPoints, double *data, const char* const* labels = 0) = 0;

Description

Sets the data used to draw the pie chart.

Arguments

Argument	Default Value	Description
noOfPoints	(Mandatory)	The number of data points, that is, the number of sectors in the pie chart.
data	(Mandatory)	An array of double precision numbers representing the data points. The length of the array should be at least the noOfPoints.
labels	0 (NULL)	An array of text strings (const char *) that represents the labels of the sectors. The length of the array should be at least the noOfPoints. If this argument is 0 (NULL), the sectors will have no label.

Return Value

None

sector

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 setData 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
setExplode	Explode the sector.
setLabelFormat	Sets the format of the sector label.
setLabelStyle	Sets the style used to draw the sector label.
setLabelPos	Set the location of the sector label, and specify whether a join line is used to connect the sector label to the sector perimeter.

setExplode

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

Prototype

virtual void setLabelFormat(const char *formatString) = 0;

Description

Sets 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 setLabelFormat method of the PieChart object for details.

Return Value

None

setLabelStyle

Prototype

virtual TextBox *setLabelStyle(const char *font, double fontSize = 8, int fontColor = TextColor) = 0;

Description

Sets 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	0	The font used to draw the sector label. A null pointer ‘0’ means using the default font (Arial).
fontSize	8	The font size used to draw the sector label.

fontColor	TextColor	The color used to draw the sector label.

Return Value

A TextBox object that represents the prototype of the sector label. You may use the methods of the TextBox object to fine tune the appearance of the sector label.

setLabelPos

Prototype

virtual void setLabelPos(int pos, int joinLineColor = Transparent) = 0;

Description

Set the location of the sector label, 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

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.

Because the XYChart is an abstract class, you cannot create it directly. Instead, you must use the XYChart::create method to create a XYChart object. After you have finished using the XYChart object, you must use the destroy method to free the XYChart object.

Method	Description
create	Creates an XYChart object.

yAxis	Retrieve the YAxis object representing primary y-axis object of the XYChart.
yAxis2	Retrieve the YAxis object representing secondary y-axis object of the XYChart.
syncYAxis	Specify that the secondary y-axis be derived from the primary y-axis through a formula y2 = y1 * slope + intercept.
setYAxisOnRight	Specifies 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).
xAxis	Retrieve the XAxis object representing primary x-axis object of the XYChart.
xAxis2	Retrieve the XAxis object representing secondary x-axis object of the XYChart.
setPlotArea	Sets the position, size, background colors, edge color and grid colors of the plot area.
addBarLayer	Add a bar chart layer to the XYChart, and specify the data set to use for drawing the bars.
addBarLayer(2)	Add an empty bar chart layer to the XYChart.
addLineLayer	Add a line chart layer to the XYChart, and specify the data set to use for drawing the line.
addLineLayer(2)	Add an empty line chart layer to the XYChart.
addAreaLayer	Add an area chart layer to the XYChart, and specify the data set to use for drawing the area.
addAreaLayer(2)	Add an empty area chart layer to the XYChart.
addHLOCLayer	Add a high-low-open-close (HLOC) chart layer to the XYChart, and specify the data sets to use for drawing the layer.
addHLOCLayer(2)	Add an empty high-low-open-close (HLOC) chart layer to the XYChart.
Methods inherited from BaseChart	
destroy	Destroy the chart object.
setSize	Set the size of the chart to the specified width and height in pixels.
setBorder	Set the border color of the chart.
setWallpaper	Specify an image as the background wallpaper of the chart.
setBgImage	Specify an image as the background image of the chart.
addTitle	Add a title to the chart on the TopCenter position of the chart
addTitle(2)	Add a title to the chart at the top, bottom, left or right position of the chart.
addLegend	Add a legend box to the chart.

getDrawArea	Returns the DrawArea object that the chart is drawn with to allow drawing custom text, line or shapes.
addDrawObj	Add a custom-developed DrawObj to the chart.
addText	Add a text box to the chart.
addLine	Add a line to the chart.
setColor	Change the color of the specified position in the palette color table.
setColors	Change the colors of the color table starting with the specified position in the palette color table.
setColors(2)	Change the colors of the color table starting from the first color.
getColor	Get the color of the specified position in the palette color table.
layout	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.
makeChart	Generate the chart and save it into a file.
makeChart(2)	Generate the chart in memory.

create

Prototype

static XYChart *create(int width, int height, int bgColor = BackgroundColor, int edgeColor = Transparent);

Description

Creates 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	BackgroundColor	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.

yAxis

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

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

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 * slope + 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

Prototype

```
virtual void setYAxisOnRight(bool b) = 0;
```

Description

Specifies 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	(Mandatory)	If this argument is “true”, 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.

xAxis

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

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.

setPlotArea

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	LineColor	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

Prototype

virtual BarLayer *addBarLayer(int noOfPoints, const double *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 addBarLayer(2) method instead.

Arguments

Argument	Default Value	Description
noOfPoints	(Mandatory)	The number of points in the data set.
data	(Mandatory)	An array of double precision numbers representing the data set. Note that ChartDirector supports the special constant NoValue 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	0	The name of the data set. The name will be used in the legend box, if one is available. The default value of 0 (NULL pointer) 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.

addBarLayer(2)

Prototype

virtual BarLayer *addBarLayer(Layer::DataCombineMethod dataCombineMethod = Layer::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	Layer::Side	The method to combine the data sets together in a single bar layer. The followings are the supported methods: Layer::Side: The data sets are combined by plotting the bars side by side. Layer::Stack: The data sets are combined by stacking up the bar. Layer::Overlay: The data sets are represented by stacked bars similar to Layer::Stack. However, in Layer::Overlay, one data set is assumed to include the other data set. 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. Instead, we should stack the “peak loading – average loading” on top of the “average loading”. The Layer::Overlay will automatically do this. In general, if there are multiple data sets, the Layer::Overlay will sort the data sets by their values, and assume the large data include the smaller data.
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

Prototype

virtual LineLayer *addLineLayer(int noOfPoints, const double *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 addLineLayer(2) method instead.

Arguments

Argument	Default Value	Description
noOfPoints	(Mandatory)	The number of points in the data set.
data	(Mandatory)	An array of double precision numbers representing the data set. Note that ChartDirector supports the special constant NoValue as values in the array to specify that certain data points have no value.
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	0	The name of the data set. The name will be used in the legend box, if one is available. The default value of 0 (NULL pointer) 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.

addLineLayer(2)

Prototype

virtual LineLayer *addLineLayer(Layer::DataCombineMethod dataCombineMethod = Layer::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	Layer::Overlay	In this version of the ChartDirector, the only supported method is Layer::Overaly. 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.

addAreaLayer

Prototype

virtual AreaLayer *addAreaLayer(int noOfPoints, const double *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 AddAreaLayer(2) method instead.

Arguments

Argument	Default Value	Description
noOfPoints	(Mandatory)	The number of points in the data set.
data	(Mandatory)	An array of double precision numbers representing the data set. Note that ChartDirector supports the special constant NoValue as values in the array to specify that certain data points have no value.
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	0	The name of the data set. The name will be used in the legend box, if one is available. The default value of 0 (NULL pointer) 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.

addAreaLayer(2)

Prototype

virtual AreaLayer *addAreaLayer(Layer::DataCombineMethod dataCombineMethod = Layer::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	Layer::Stack	In this version of the ChartDirector, the only supported method is Layer::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

Prototype

virtual HLOCLayer *addHLOCLayer(int noOfPoints, const double *highData, const double *lowData, const double *openData, const double *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
noOfPoints	(Mandatory)	The number of points in the data sets.

highData	(Mandatory)	An array of double precision numbers representing the highest value data set. Note that ChartDirector supports the special constant NoValue as values in the array to specify that certain data points have no value.
lowData	(Mandatory)	An array of double precision numbers representing the lowest value data set. Note that ChartDirector supports the special constant NoValue as values in the array to specify that certain data points have no value.
openData	0	An array of double precision numbers representing the opening value data set. A value of 0 (NULL pointer) means there is no opening value data set available. Note that ChartDirector supports the special constant NoValue as values in the array to specify that certain data points have no value.
closeData	0	An array of double precision numbers representing the closing value data set. A value of 0 (NULL pointer) means there is no closing value data set available. Note that ChartDirector supports the special constant NoValue as values in the array to specify that certain data points have no value.
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.

addHLOCLayer(2)

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.

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
setBackground	Sets the background colors and the border color of the plot area.
setBackground(2)	Specify an image as the background image of the plot area.
setGridColor	Sets the horizontal and vertical grid colors of the plot area.

setBackground

Prototype

virtual void setBackground(int color, int altBgColor = -1, int edgeColor = LineColor) = 0;

Description

Sets 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

setBackground(2)

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 Alignment Specification for possible alignment types.

Return Value

None

setGridColor

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

BaseAxis

The BaseAxis is the base class of the XAxis and YAxis in ChartDirector. It represents methods that area common to both axis classes.

Method	Description
setLabelStyle	Sets the font style used to for the axis labels.
setLabelGap	Sets the distance between the axis labels and the ticks on the axis.
setTitle	Add a title to the axis.
setTitlePos	Set the title position relative to the axis.
setColors	Sets the axis color, axis label color, axis title color and axis tick color.
setTickLength	Sets the axis ticks length in pixels.
setTickLength(2)	Sets the major and minor axis ticks lengths in pixels.

setLabelStyle

Prototype

virtual TextBox *setLabelStyle(const char *font = 0, double fontSize = 8, int fontColor = TextColor, double fontAngle = 0) = 0;

Description

Sets 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	0	The font used to draw the labels. A null pointer ‘0’ 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 prototype of the axis labels. You may use the methods of the TextBox object to fine-tune the appearance of the axis labels.

setLabelGap

Prototype

virtual void setLabelGap(int d) = 0;

Description

Sets 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

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 TextColor 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

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

Prototype

virtual void setColors(int axisColor, int labelColor = TextColor, int titleColor = -1, int tickColor = -1) = 0;

Description

Sets 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	TextColor	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

setTickLength

Prototype

virtual void setTickLength(int majorTickLen) = 0;

Description

Sets 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

setTickLength(2)

Prototype

virtual void setTickLength(int majorTickLen, int minorTickLen) = 0;

Description

Sets 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

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 method of the XYChart object. The top x-axis is the secondary x-axis and can be obtained by using the xAxis2 method of the XYChart object.

The XAxis class is a subclass of the BaseAxis class.

Method	Description
setLabels	Sets the labels to be used on the x-axis.
setIndent	Specify where the x-axis should be "indented" or not.
Methods inherited from BaseAxis	
setLabelStyle	Sets the font style used to for the axis labels.
setLabelGap	Sets the distance between the axis labels and the ticks on the axis.
setTitle	Add a title to the axis.
setTitlePos	Set the title position relative to the axis.
setColors	Sets the axis color, axis label color, axis title color and axis tick color.
setTickLength	Sets the axis ticks length in pixels.
setTickLength(2)	Sets the major and minor axis ticks lengths in pixels.

setLabels

Prototype

virtual TextBox *setLabels(int noOfLabels, const char* const* text) = 0;

Description

Sets the labels to be used on the x-axis.

If you do not call this method, there will be no labels and no axis ticks on the x-axis.

This method accepts an array of strings containing the labels. The text also specifies what kind of axis tick to use for a label.

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 tick that really begins with '-' or '~' and does not want the ChartDirector to interpret it as special characters, add the '\' character as the first character of the label.

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
noOfLabels	(Mandatory)	No of labels in the text array below.
text	(Mandatory)	An array of strings (const char *), that is, an array to pointer of characters. This array contains 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 prototype of the axis labels. You may use the methods of the TextBox object to fine-tune the appearance of the axis labels.

setIndent

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" 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 method of the XYChart object, while the secondary y-axis can be obtained by using the yAxis2 method 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
addMark	Add a mark line to the chart.
addZone	Add a zone to the chart.
setLinearScale	Set the axis to use linear scaling and manually determine the scale.
setLogScale	Specify the type of auto-scaling (log or linear) for the y-axis.
setLogScale(2)	Set the axis to use log scaling and manually determine the scale.
setAutoScale	Reserve some space at the top and/or bottom of the plot area by using a larger axis scaling than is necessary.
setTickDensity	Sets the density of the axis ticks.
setTopMargin	Reserve a range at the top of the plot area that is not scaled at all.
setLabelFormat	Sets the number format for the axis labels.
Methods inherited from BaseAxis	
setLabelStyle	Sets the font style used to for the axis labels.
setLabelGap	Sets the distance between the axis labels and the ticks on the axis.
setTitle	Add a title to the axis.
setTitlePos	Set the title position relative to the axis.
setColors	Sets the axis color, axis label color, axis title color and axis tick color.
setTickLength	Sets the axis ticks length in pixels.
setTickLength(2)	Sets the major and minor axis ticks lengths in pixels.

addMark

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 horizontal line drawn on the front of the plot area. This line is usually used to indicate some special values, such as a “target value”, “threshold value”, etc. The mark line will include a tick and label on the y-axis to describe the mark.

Arguments

Argument	Default Value	Description
value	(Mandatory)	The y value of the mark line.
lineColor	(Mandatory)	The color of the mark line. By default, the text label and the corresponding tick on the y-axis will be drawn using the same color as the mark line. You can modify the colors by using the setMarkColors method of the returned Mark object.
text	0	The text label for the mark line. The default value of 0 (NULL pointer) means there is no text label.
font	0	The font used to draw the text label. The default value of 0 (NULL pointer) means using the default font (Arial).
fontSize	8	The font size used to draw the text label. 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

Prototype

virtual void addZone(double startValue, double endValue, int color) = 0;

Description

Add a zone to the chart.

A zone is a range of y 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 is drawn on the back of the plot area as a horizontal strip using a user specified color. For example, you may draw the normal zone in green color, the warning zone in yellow color and the critical zone in red color.

Arguments

Argument	Default Value	Description
startValue	(Mandatory)	The start y value (the lower bound) of the zone.
endValue	(Mandatory)	The end y value (the upper bound) of the zone.
color	(Mandatory)	The color of the zone.

Return Value

None.

setLinearScale

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

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	true	The default value of "true" means the axis will use log scaling. A "false" value means linear scaling.

Return Value

None.

setLogScale(2)

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

Prototype

virtual void setAutoScale(double topExtension = 0, double bottomExtension = 0) = 0;

Description

Reserve some space at the top and/or bottom of the plot area by using a larger axis scaling than is necessary. 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.

By default, the ChartDirector will auto-scale the y-axis so that it approximately matches the range of the data. The setAutoScale method allows you to specify a portion on the top and on the bottom of the axis where no data value falls.

This method is usually used when you want to reserve some space at the top and/or bottom 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 of the plot area – the setTopMargin method.

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 .

Return Value

None.

setTickDensity

Prototype

virtual void setTickDensity(int tickDensity) = 0;

Description

Sets 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

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

Prototype

virtual void setLabelFormat(const char *formatString) = 0;

Description

Sets the number format for the axis labels.

By default, the axis label will be automatically formatted. The ChartDirector will display a value as an integer if it looks like an integer. If the value is not an integer, it will display it using the least possible decimal points.

You may modify the number format by specifying the number of decimal points, add thousand separators, or add additional text before or after the data value.

For example, if you want to display the value 100 as “USD 100K”, you could use the following format string “USD &value&K”. The ChartDirector will replace the “&value& will the actual data value when drawing the data label.

The ChartDirector supports multi-line data labels. Simply use the new line character ‘\n’ for multiple lines.

As an other example, suppose you want the value to have a precision of two decimal points, using ‘.’ (dot) as the decimal point, and using ‘,’ (comma) as the thousand separator. In this case the value 123456.789 will be displayed as “123,456.79”. The format string to use is “&value|2.,&”.

In the “&value|2.,&” format string, the ‘|’ character means that there are formatting options for the value. The first character after ‘|’ is the number of decimal points, the following character is the decimal point character, and next character is the thousand separator.

If you leave out the thousand separator character, there will be no thousand separator, so “123456” will be displayed as exactly “123456” and not “123,456”. If you leave out the decimal point character, it will be ‘.’ (dot).

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 horizontal line drawn on the front of the plot area. This line is usually used to indicate some special values, such as a “target value”, “threshold value”, etc.

A mark line is created using the addMark method of a YAxis object. The mark line will include a tick and label on the y-axis to describe the mark.

Method	Description
setValue	Set the y value of the mark line.
setMarkColor	Set the line, text and tick colors of the mark line.
setLineWidth	Set the line width of the mark line**.**

setValue

Prototype

virtual void setValue(double value) = 0;

Description

Set the y value of the mark line.

Arguments

Argument	Default Value	Description
value	(Mandatory)	The y value of the mark.

Return Value

None

setMarkColor

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

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

Layer

The Layer class is a base class for all XYChart layer classes. Currently these include the BarLayer, LineLayer, AreaLayer and HLOCLayer.

Method	Description
set3D	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
setLineWidth	Set the default line width of data lines when drawing data sets on the layer.
setDataCombineMethod	Set the method used to combine multiple data sets in a layer.
addDataSet	Add a data set to the chart layer.
getXCoor	Get the x coordinate of a point given the x value.
getYCoor	Get the y coordinate of a point given the y value.
setDataLabelFormat	Sets the data label format of the all data labels for all data sets in the layer.
setDataLabelStyle	Sets the style used to draw data labels for all data sets in the layer.
setAggregateLabelFormat	Sets the data label format of the aggregate data labels.
setAggregateLabelStyle	Enables and sets the style used to draw aggregate data labels in the layer.

set3D

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 width the front of the next layer.

Return Value

None.

setLineWidth

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. In the current version of ChartDirector, these include line layers, HLOC layers and area layers. (For an area layer, the line is boundary line of the area.)

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.

setDataCombineMethod

Prototype

virtual void setDataCombineMethod(DataCombineMethod 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.

Not all layers support all data combine methods. The following table describes what are the supported data combine methods for each layer type.

Data Combine Method	Description	Applies to
Layer::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.	BarLayer, AreaLayer

Layer::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.	BarLayer LineLayer
Layer::Side	The multiple data sets are plotted side-by-side. Currently, this method only applies to bar charts.	BarLayer

Arguments

Argument	Default Value	Description
m	(Mandatory)	The data combine method, which must be Layer::Side, Layer::Stack or Layer::Overlay.

Return Value

None.

addDataSet

Prototype

virtual DataSet *addDataSet(int noOfPoints, const double *d, int color = -1, const char *name = 0) = 0;

Description

Add a data set to the chart layer. A layer can contain multiple data sets.

Arguments

Argument	Default Value	Description
noOfPoints	(Mandatory)	The number of values in the data set.
d	(Mandatory)	An array of double precision floating point numbers representing the values in the data set. Note that ChartDirector supports the special constant NoValue 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	0	The name of the data set. If a legend box is available in the XYChart, a key will be automatically added to the legend box to describe the data set. A value of 0 (NULL pointer) 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.

getXCoor

Prototype

virtual int getXCoor(double v) = 0;

Description

Get the x coordinate of a point given the x value. The x coordinate for the first data point is given by getXCoor(0), and the second data point is givenby getXCoor(1). In general, the nth data point has an x coordinate of getXCoor(n – 1).

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 x value. The nth data point has an x value of n – 1.

Return Value

The x coordinate of the x value.

getYCoor

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	true	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" 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

Prototype

virtual void setDataLabelFormat(const char *formatString) = 0;

Description

Sets 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 besides the data points in the chart. In the current version of the ChartDirector, only bar chart layers support data labels (the data labels, if enabled, are drawn on top of the bars). For layers that does not support data labels, the data label settings are ignored.

Note that data labels in a layer are disabled by default. You need to call the setDataLabelStyle method to enable them.

By default, the data label will be automatically formatted. The ChartDirector will display data value as an integer if it looks like an integer. If the data value is not an integer, it will display it using the least possible decimal points.

You may modify the data label format by specifying the number of decimal points, add thousand separators, or add additional text before or after the data value.

For example, if you want to display the data value 10.5 as "USD 10.5 K", you could use the following format string "USD &value& K". The ChartDirector will replace the "&value& will the actual data value when drawing the data label.

The ChartDirector supports multi-line data labels. Simply use the new line character '\n' for multiple lines.

For example, suppose you want the value to have a precision of two decimal points, using '.' (dot) as the decimal point, and using ',' (comma) as the thousand separator. In this case the value 123456.789 will be displayed as "123,456.79". The format string to use is "&value|2.,&".

In the "&value|2.,&" format string, the '|' character means that there are formatting options for the value. The first character after '|' is the number of decimal points, the following character is the decimal point character, and next character is the thousand separator.

If you leave out the thousand separator character, there will be no thousand separator, so "123456" will be displayed as exactly "123456" and not "123,456". If you leave out the decimal point character, it will be '.' (dot).

Arguments

Argument	Default Value	Description
formatString	(Mandatory)	The format string. See above for description.

Return Value

None

setDataLabelStyle

Prototype

virtual TextStyle *setDataLabelStyle(const char *font = 0, double fontSize = 8, int fontColor = TextColor, double fontAngle = 0) = 0;

Description

Sets 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 besides the data points in the chart. In the current version of the ChartDirector, only bar chart layers support data labels. The exact locations of the data labels depend on the type of bar chart. For stacked bar chart or overlay bar chart, the data labels are drawn just under the top edge of the bar segments. For multi-bar chart, the data labels are drawn on top of the bar.

For bar chart layer that only have one data set and created using the addBarLayer method, it will be considered as a special case of a multi-bar chart that only has one data set. Therefore the data labels will be drawn on the top of the bar. If you want to draw the data label under the top edge of the bar, you need to use the addBarLayer(2) method to specify that the bar is of Layer::Stack type, and then add the data using the addDataSet method.

Note that data labels in a layer are disabled by default. You need to call the setDataLabelStyle method to enable them.

For details about how to specify font style, please refer to the section on Font Specification.

Arguments

Argument	Default Value	Description
font	0	The font used to draw the labels. A null pointer '0' 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 prototype of the data labels. You may use the methods of the TextBox object to fine-tune the appearance of the data labels.

setAggregateLabelFormat

Prototype

virtual void setAggregateLabelFormat(const char *formatString) = 0;

Description

Sets the data label format of the aggregate data labels.

Arguments

Argument	Default Value	Description
formatString	(Mandatory)	The format string. The syntax is the same as that of the setDataLabelFormat method. Please refer to setDataLabelFormat for details.

Return Value

None

setAggregateLabelStyle

Prototype

virtual TextStyle *setAggregateLabelStyle(const char *font = 0, double fontHeight = 8, int fontColor = TextColor, double fontAngle = 0) = 0;

Description

Enables and sets 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.

In the current version of the ChartDirector, only bar chart layers support data labels. The aggregate data labels are drawn on top of the bar.

Note that aggregate data labels in a layer are disabled by default. You need to call the setAggregateLabelStyle method to enable them.

For details about how to specify font style, please refer to the section on Font Specification.

Arguments

Argument	Default Value	Description
font	0	The font used to draw the labels. A null pointer ‘0’ 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 prototype of the aggregate data labels. You may use the methods of the TextBox object to fine-tune the appearance of the aggregate data labels.

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 or addBarLayer(2) methods of the XYChart object.

Method	**Description**
setBarGap	Sets the gap between the bars in a bar chart layer.
Methods inherited from Layer	
set3D	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
setLineWidth	Set the default line width of data lines when drawing data sets on the layer.
setDataCombineMethod	Set the method used to combine multiple data sets in a layer.
addDataSet	Add a data set to the chart layer.
getXCoor	Get the x coordinate of a point given the x value.
getYCoor	Get the y coordinate of a point given the y value.
setDataLabelFormat	Sets the data label format of the all data labels for all data sets in the layer.
setDataLabelStyle	Sets the style used to draw data labels for all data sets in the layer.
setAggregateLabelFormat	Sets the data label format of the aggregate data labels.
setAggregateLabelStyle	Enables and sets the style used to draw aggregate data labels in the layer.

setBarGap

Prototype

virtual void setBarGap(double barGap, double subBarGap = 0.2) = 0;

Description

Sets 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 must be in the range 0 – 1. A value of 0 means the bars are tightly packed together with no gap in between. 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 must be 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.

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 addLineLayer(2) methods of the XYChart object.

The LineLayer has no additional method other than implementing methods inherited from the Layer class.

Method	Description
Methods inherited from Layer	
set3D	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
setLineWidth	Set the default line width of data lines when drawing data sets on the layer.
setDataCombineMethod	Set the method used to combine multiple data sets in a layer.
addDataSet	Add a data set to the chart layer.
getXCoor	Get the x coordinate of a point given the x value.
getYCoor	Get the y coordinate of a point given the y value.
setDataLabelFormat	Sets the data label format of the all data labels for all data sets in the layer.
setDataLabelStyle	Sets the style used to draw data labels for all data sets in the layer.

setAggregateLabelFormat	Sets the data label format of the aggregate data labels.
setAggregateLabelStyle	Enables and sets the style used to draw aggregate data labels in the layer.

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 addAreaLayer(2) methods of the XYChart object.

The AreaLayer has no additional method other than implementing methods inherited from the Layer class.

Method	Description
Methods inherited from Layer	
set3D	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
setLineWidth	Set the default line width of data lines when drawing data sets on the layer.
setDataCombineMethod	Set the method used to combine multiple data sets in a layer.
addDataSet	Add a data set to the chart layer.
getXCoor	Get the x coordinate of a point given the x value.
getYCoor	Get the y coordinate of a point given the y value.
setDataLabelFormat	Sets the data label format of the all data labels for all data sets in the layer.
setDataLabelStyle	Sets the style used to draw data labels for all data sets in the layer.
setAggregateLabelFormat	Sets the data label format of the aggregate data labels.
setAggregateLabelStyle	Enables and sets the style used to draw aggregate data labels in 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 addHLOCLayer(2) methods of the XYChart object.

The HLOCLayer has no additional method other than implementing methods inherited from the Layer class.

Method	Description

Methods inherited from Layer	
set3D	Set the 3D depth of the layer, and the 3D gap between the current layer and the next layer.
setLineWidth	Set the default line width of data lines when drawing data sets on the layer.
setDataCombineMethod	Set the method used to combine multiple data sets in a layer.
addDataSet	Add a data set to the chart layer.
getXCoor	Get the x coordinate of a point given the x value.
getYCoor	Get the y coordinate of a point given the y value.
setDataLabelFormat	Sets the data label format of the all data labels for all data sets in the layer.
setDataLabelStyle	Sets the style used to draw data labels for all data sets in the layer.
setAggregateLabelFormat	Sets the data label format of the aggregate data labels.
setAggregateLabelStyle	Enables and sets the style used to draw aggregate data labels in the layer.

DataSet

The DataSet class represents data sets. It is created by using the addDataSet method of the Layer class.

Method	Description
setDataName	Sets the name of the data set.
setDataColor	Sets the colors used to draw the data set.
setUseYAxis2	Determine whether the primary y-axis or secondary y-axis to use when drawing the data set on the chart.
setLineWidth	Sets the width of data lines when drawing the data set on the layer.
setDataLabelFormat	Sets the data label format of the data labels of the data set.
setDataLabelStyle	Sets the style used to draw data labels for the data set.

setDataName

Prototype

virtual void setDataName(const char *name) = 0;

Description

Sets 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

Prototype

virtual void setDataColor(int dataColor, int edgeColor = LineColor, int shadowColor = -1, int shadowEdgeColor = -1) = 0;

Description

Sets 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	LineColor	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. The default value of LineColor means that the edges are drawn using the default line color from the color palette.
shadowColor	-1	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.
shadowEdgeColor	-1	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.

Return Value

None.

setUseYAxis2

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	true	A “true” value means the secondary y-axis will be used. A “false” value means the primary y-axis will be used.

Return Value

None.

setLineWidth

Prototype

virtual void setLineWidth(int w) = 0;

Description

Sets the width of data lines when drawing the data set on the layer. This only applies to layers that employ lines to represent data. In the current version of ChartDirector, these include line layers, HLOC layers and area layers. (For an area layer, the line is boundary line of the area.)

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.

setDataLabelFormat

Prototype

virtual void setDataLabelFormat(const char *formatString) = 0;

Description

Sets the data label format of the data labels of the data set. If you just 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. In the current version of the ChartDirector, only bar chart layers support data labels (the data labels, if enabled, are drawn on top of the bars). For layers that does not support data labels, the data label settings are ignored.

Note that data labels in a layer are disabled by default. You need to call the setDataLabelStyle method to enable them.

The syntax of the format string is the same as that for the setDataLabelFormat method of the Layer object. Please refer to the section on the setDataLabelFormat method of the Layer object for details.

Arguments

Argument	Default Value	Description
formatString	(Mandatory)	The format string. See above for description.

Return Value

None

setDataLabelStyle

Prototype

virtual TextStyle *setDataLabelStyle(const char *font = 0, double fontHeight = 8, int fontColor = TextColor, double fontAngle = 0) = 0;

Description

Sets the style used to draw data labels for the data set. If you just 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. In the current version of the ChartDirector, only bar chart layers support data labels. The exact locations of the data labels depend on the type of bar chart. For stacked bar chart or overlay bar chart, the data labels are drawn just under the top edge of the bar segments. For multi-bar chart, the data labels are drawn on top of the bar.

For bar chart layer that only have one data set and created using the addBarLayer method, it will be considered as a special case of a multi-bar chart that only has one data set. Therefore the data labels will be drawn on the top of the bar. If you want to draw the data label under the top edge of the bar, you need to use the addBarLayer(2) method to specify that the bar is of Layer::Stack type, and then add the data using the addDataSet method.

Note that data labels in a layer are disabled by default. You need to call the setDataLabelStyle method to enable them.

For details about how to specify font style, please refer to the section on Font Specification.

Arguments

Argument	Default Value	Description
font	0	The font used to draw the labels. A null pointer ‘0’ 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 prototype 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. Simply uses the DrawArea::create method to create a DrawArea object, apply the desired drawing methods, then use the out method to output the drawings to a graphics file (GIF, JPEG, PNG or WBMP), and finally uses the DrawArea::destroy method to free the DrawArea object.

Method	Description
create	Create a DrawArea object.
destroy	Destroy the DrawArea object.
setSize	Set the size and background color of the image.
getWidth	Get the width of the image.
getHeight	Get the height of the image.
setBgColor	Set the background color of the image.
pixel	Apply the specified color to a pixel.
getPixel	Get the color of a pixel.
line	Draw a straight line.
hline	Draw a horizontal line.
vline	Draw a vertical line.
arc	Draw an arc.
rect	Draw a rectangle.
polygon	Draw a polygon.

surface	Draw a 3D surface.
sector	Draw a sector.
cylinder	Draw a cylinder surface.
circle	Draw a circle or an ellipse.
fill	Fill an area using the specified color, where the area is bounded by a given border color.
fill(2)	Fill an area using the specified color, where the area is defined as a continuous region having the same color.
text	Draw text on the image.
text(2)	Draw Unicode text on the image.
text(3)	Draw text on the image. This method is exactly the same as the text method except that it is simplied to contain less arguments.
text(4)	Draw Unicode text on the image. This method is exactly the same as the text(2) method except that it is simplied to contain less arguments.
text(5)	Create a TTFText object that represents the text to be drawn.
text(6)	Create a TTFText object that represents the Unicode text to be drawn.
text(7)	Create a TTFText object that represents the text to be drawn. This method is exactly the same as the text(5) method except that it is simplied to contain less arguments.
text(8)	Create a TTFText object that represents the Unicode text to be drawn. This method is exactly the same as the text(6) method except that it is simplied to contain less arguments.
close	Destroy the TTFText object created using text(5), text(6), text(7) or text(8).
merge	Apply another DrawArea image on top of the current DrawArea image.
tile	Apply another DrawArea image on top of the current DrawArea image as a wallpaper.
load	Load the specified image into the current DrawArea. This method will determine the image type by using the extension of the filename.
loadGIF	Load the specified GIF image into the current DrawArea.
loadPNG	Load the specified PNG image into the current DrawArea.
loadJPG	Load the specified JPEG image into the current DrawArea.
loadWMP	Load the specified WAP bitmap image into the current DrawArea.
out	Write the current DrawArea image to an image file. This method will determine the image type by using the extension of the filename.

outGIF	Write the current DrawArea image to an alternative GIF image file.
outGIF(2)	Write the current DrawArea image as an alternative GIF image in memory.
outPNG	Write the current DrawArea image to a PNG image file.
outPNG(2)	Write the current DrawArea image as a PNG image in memory.
outJPG	Write the current DrawArea image to a JPEG image file.
outJPG(2)	Write the current DrawArea image as a JPG image in memory.
outWMP	Write the current DrawArea image to a WAP bitmap image file.
outWMP(2)	Write the current DrawArea image as a WAP bitmap image in memory.
setPaletteMode	Set the palette mode when writing the image to a PNG file.
setDitherMethod	Set the dithering method if dithering is required.
setTransparentColor	Set the transparent color for the image when writing the image to an image file.
setAntiAliasText	Set whether anti-alias methods are when drawing text.
setInterlace	Set the interlace mode when writing the image to an image file.
setColorTable	Change the colors of the palette color table.
getARGBColor	Change the given color to ARGB format if the given color is a palette table color.

create

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

Prototype

virtual void destroy() = 0;

Description

Destroy the 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 and destroyed by the BaseChart.

Arguments

None.

Return Value

None.

setSize

Prototype

virtual void setSize(int width, int height, int bgColor = 0xffffff) = 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	0xffffff	The background color of the image.

Return Value

None.

getWidth

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

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

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

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

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

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

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

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

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

Prototype

virtual void rect(int x1, int y1, int x2, int y2, int edgeColor, int fillColor) = 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.

Return Value

None.

polygon

Prototype

virtual void polygon(const int *x, const int *y, int noOfPoints, int edgeColor, int fillColor) = 0;

Description

Draw a polygon.

Arguments

Argument	Default Value	Description
x	(Mandatory)	An array containing the x coordinates of the vertices of the polygon.
y	(Mandatory)	An array containing the y coordinates of the vertices of the polygon.
noOfPoints	(Mandatory)	The no of vertices the polygon has.
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

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 parallologram 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 parallolgram.)
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 parallolgram.)

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 parallolgram.)
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 parallolgram.)
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

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

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

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

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

fill(2)

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

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 Font Specification 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 Font Specification 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" 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 Alignment Specification for possible alignment positions.

Return Value

None

text(2)

Prototype

virtual void text(const wchar_t *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 Unicode text on the image. This method is exactly the same as the text method except that it employs wide Unicode characters to represent 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 Font Specification 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 Font Specification 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" 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 Alignment Specification for possible alignment positions.

Return Value

None

text(3)

Prototype

virtual void text(const char *str, const char *font, double fontSize, int x, int y, int color, Alignment alignment) = 0;

Description

Draw text on the image. This method is exactly the same as the text method except that it is simplied 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 Font Specification 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.
alignment	TopLeft	The location of the reference point of the text. See Alignment Specification for possible alignment positions.

Return Value

None

text(4)

Prototype

virtual void text(const wchar_t *str, const char *font, double fontSize, int x, int y, int color, Alignment alignment) = 0;

Description

Draw Unicode text on the image. This method is exactly the same as the text(2) method except that it is simplied 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 Font Specification 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.
alignment	TopLeft	The location of the reference point of the text. See Alignment Specification for possible alignment positions.

Return Value

None

text(5)

Prototype

virtual TTFText* text(const char *text, const char *font, int fontIndex, double fontHeight, double fontWidth, double angle, bool vertical) = 0;

Description

Create a TTFText object that represents the text to be drawn. You may later call the draw method of the TTFText object to draw the text. After you have drawn the TTFText object, you should use the close method to destroy the TTFText object.

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 TTFText 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 Font Specification 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 Font Specification 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” 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 TTFText object created.

text(6)

Prototype

virtual TTFText* text(const wchar_t *text, const char *font, int fontIndex, double fontHeight, double fontWidth, double angle, bool vertical) = 0;

Description

Create a TTFText object that represents the Unicode text to be drawn. You may later call the draw method of the TTFText object to draw the text. After you have drawn the TTFText object, you should use the close method to destroy the TTFText object.

This method is exactly the same as the text(5) method except that it employs wide Unicode characters to represent 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 TTFText 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 Font Specification 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 Font Specification 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” 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 TTFText object created.

text(7)

Prototype

virtual TTFText* text(const char *str, const char *font, double fontSize) = 0;

Description

Create a TTFText object that represents the text to be drawn. You may later call the draw method of the TTFText object to draw the text. After you have drawn the TTFText object, you should use the close method to destroy the TTFText object.

This method is exactly the same as the text(5) method except that it is simplied 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 TTFText 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 Font Specification 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 TTFText object created.

text(8)

Prototype

virtual TTFText* text(const wchar_t *str, const char *font, double fontSize) = 0;

Description

Create a TTFText object that represents the Unicode text to be drawn. You may later call the draw method of the TTFText object to draw the text. After you have drawn the TTFText object, you should use the close method to destroy the TTFText object.

This method is exactly the same as the text(6) method except that it is simplied 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 TTFText 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 Font Specification 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 TTFText object created.

close

Prototype

virtual void close(TTFText *text) = 0;

Description

Destroy the TTFText object created using text(5), text(6), text(7) or text(8).

Arguments

Argument	Default Value	Description
text	(Mandatory)	The TTFText object to destroy.

Return Value

None.

merge

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 Alignment Specification 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

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

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 "true" value indicates the load operation is successful, otherwise a "false" value is returned.

loadGIF

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 "true" value indicates the load operation is successful, otherwise a "false" value is returned.

loadPNG

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 "true" value indicates the load operation is successful, otherwise a "false" value is returned.

loadJPG

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 “true” value indicates the load operation is successful, otherwise a “false” value is returned.

loadWMP

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 “true” value indicates the load operation is successful, otherwise a “false” value is returned.

out

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” value indicates the write operation is successful, otherwise a “false” value is returned.

outGIF

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 outGIF(2) method instead.

Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the output image file.

Return Value

A "true" value indicates the write operation is successful, otherwise a "false" value is returned.

outGIF(2)

Prototype

virtual bool outGIF(const char **data, int *len) = 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

Argument	Default Value	Description
data	(Mandatory)	The address of a character pointer to receive the address of the memory block that holds the alternative GIF image. The memory block is allocated by the DrawArea, and will automatically free when the DrawArea is destroy. There is no need to explicitly free the memory block. It is possible to output the image in several formats by calling the several memory image output methods (that is, outGIF(2), outPNG(2), outJPG(2) and outWMP(2)) sequentially. However, note that the DrawArea only allocate one memory block at a time. Calling any memory image output method will automatically free the memory block for the previous memory image output method. Depending on your program logic, you may need to copy the memory blocks.
len	(Mandatory)	The address of an integer variable to receive the length of the memory block.

Return Value

A "true" value indicates the write operation is successful, otherwise a "false" value is returned.

outPNG

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 outPNG(2) method instead.

Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the output image file.

Return Value

A "true" value indicates the write operation is successful, otherwise a "false" value is returned.

outPNG(2)

Prototype

virtual bool outPNG(const char **data, int *len) = 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

Argument	Default Value	Description
data	(Mandatory)	The address of a character pointer to receive the address of the memory block that holds the PNG image. The memory block is allocated by the DrawArea, and will automatically free when the DrawArea is destroy. There is no need to explicitly free the memory block. It is possible to output the image in several formats by calling the several memory image output methods (that is, outGIF(2), outPNG(2), outJPG(2) and outWMP(2)) sequentially. However, note that the DrawArea only allocate one memory block at a time. Calling any memory image output method will automatically free the memory block for the previous memory image output method. Depending on your program logic, you may need to copy the memory blocks.

len	(Mandatory)	The address of an integer variable to receive the length of the memory block.

Return Value
A "true" value indicates the write operation is successful, otherwise a "false" value is returned.

outJPG

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 outJPG(2) 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" value indicates the write operation is successful, otherwise a "false" value is returned.

outJPG(2)

Prototype
virtual bool outJPG(const char **data, int *len, 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

data	(Mandatory)	The address of a character pointer to receive the address of the memory block that holds the JPG image. The memory block is allocated by the DrawArea, and will automatically free when the DrawArea is destroy. There is no need to explicitly free the memory block. It is possible to output the image in several formats by calling the several memory image output methods (that is, outGIF(2), outPNG(2), outJPG(2) and outWMP(2)) sequentially. However, note that the DrawArea only allocate one memory block at a time. Calling any memory image output method will automatically free the memory block for the previous memory image output method. Depending on your program logic, you may need to copy the memory blocks.
len	(Mandatory)	The address of an integer variable to receive the length of the memory block.

Return Value

A "true" value indicates the write operation is successful, otherwise a "false" value is returned.

outWMP

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 outWMP(2) method instead.

Arguments

Argument	Default Value	Description
filename	(Mandatory)	The filename of the output image file.

Return Value

A "true" value indicates the write operation is successful, otherwise a "false" value is returned.

outWMP(2)

Prototype

virtual bool outWMP(const char **data, int *len) = 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

Argument	Default Value	Description
data	(Mandatory)	The address of a character pointer to receive the address of the memory block that holds the WAP bitmap image. The memory block is allocated by the DrawArea, and will automatically free when the DrawArea is destroy. There is no need to explicitly free the memory block. It is possible to output the image in several formats by calling the several memory image output methods (that is, outGIF(2), outPNG(2), outJPG(2) and outWMP(2)) sequentially. However, note that the DrawArea only allocate one memory block at a time. Calling any memory image output method will automatically free the memory block for the previous memory image output method. Depending on your program logic, you may need to copy the memory blocks.
len	(Mandatory)	The address of an integer variable to receive the length of the memory block.

Return Value

A “true” value indicates the write operation is successful, otherwise a “false” value is returned.

setPaletteMode

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
DrawArea::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.
DrawArea::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.
DrawArea::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

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
DrawArea::Quantize	For any pixel, adjust it to the nearest color in the standard 216 colors web-safe color palette.
DrawArea::OrderedDither	Use ordered dithering algorithm with a 4 x 4 matrix.
DrawArea::ErrorDiffusion	Use the Floyd and Steinberg error diffusion algorithm. This is the default setting.

Arguments

Argument	Default Value	Description
m	(Mandatory)	The dithering alogorithm to use, in case it is necessary to do dithering. See above for description.

Return Value

None.

setTransparentColor

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 of 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

Prototype

virtual void setAntiAliasText(TTFText::AntiAliasMode a) = 0;

Description

Set whether anti-alias methods are when drawing text. Currently, three anti-alias modes are supported in ChartDirector as follows:

Dithering Mode	Description
TTFText::NoAntiAlias	Do not use anti-alias method to draw text.
TTFText::AntiAlias	Always use anti-alias method to draw text.
TTFText::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

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" 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

Prototype

virtual void setColorTable(const int *colors, int noOfColors, 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)	An array of colors to replace the colors in the palette color table.
noOfColors	(Mandatory)	The no of colors in the colors array.
offset	(Mandatory)	The offset position that marks start position within the palette color table where the colors will be replaced.

Return Value

None

getARGBColor

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.

TTFText

The TTFText object represents text strings. It is created by text(5), text(6), text(7) or text(8) method of the DrawArea object.

Method	Description
getWidth	Get the width of the bounding box of the text.
getHeight	Get the height of the bounding box of the text.
getLineHeight	Get the height of the bounding box of one line of text in pixels.
getLineDistance	Get the line distance of the text in pixels.
draw	Draw the text.

getWidth

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

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

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

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

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 Alignment Specification for possible alignment positions.

Return Value

None.

ChartDirector API Index