
Plotting Graphs in SVG format.

Jake Voytko
Paul A. Bristow

Copyright © 2007 to 2013 Jake Voytko and Paul A. Bristow

Distributed under the Boost Software License, Version 1.0. (See accompanying file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

Table of Contents

Preface	3
How To Use This Documentation	5
Colors	6
Fonts	8
1D Tutorials	14
1-D Vector Example	14
1-D STL Containers Examples	15
Tutorial: 1D Autoscale with Multiple Containers	18
Tutorial: 1D More Layout Examples	20
Tutorial: 1D Gridlines & Axes - more examples	22
1-D Axis Scaling	26
1-D Auto scaling Examples	31
1-D Autoscaling Various Containers Examples	40
1-D Data Values Examples	43
Real-life Heat flow data	48
Demonstration of using 1D data that includes information about its Uncertainty	54
2D Tutorial	58
Simple Code Example	58
Tutorial: Fuller Layout Example	60
Tutorial: 2D Special Features	62
2-D Data Values Examples	67
2-D Autoscaling Examples	73
Demonstration of using 2D data that includes information about its uncertainty	76
Demonstration of adding lines and curves, typically a least squares fit	80
Histograms of 2D data	82
Tutorial: Boxplot	84
Simple Example	84
Definitions of the Quartiles	86
SVG tutorial	91
Showing data values at Numerical Limits	92
Boost.SVG plot C++ Reference	94
Header <boost/svg_plot/detail/auto_axes.hpp>	94
Header <boost/svg_plot/detail/axis_plot_frame.hpp>	102
Header <boost/svg_plot/detail/fp_compare.hpp>	104
Header <boost/svg_plot/detail/functors.hpp>	107
Header <boost/svg_plot/detail/numeric_limits_handling.hpp>	107
Header <boost/svg_plot/detail/pair.hpp>	108
Header <boost/svg_plot/detail/svg_boxplot_detail.hpp>	108
Header <boost/svg_plot/detail/svg_style_detail.hpp>	108
Header <boost/svg_plot/detail/svg_tag.hpp>	108
Header <boost/svg_plot/detail/quantile.hpp>	168
Header <boost/svg_plot/show_1d_settings.hpp>	169
Header <boost/svg_plot/show_2d_settings.hpp>	170

Header <boost/svg_plot/svg.hpp>	171
Header <boost/svg_plot/svg_1d_plot.hpp>	178
Header <boost/svg_plot/svg_2d_plot.hpp>	215
Header <boost/svg_plot/svg_boxplot.hpp>	267
Header <boost/svg_plot/svg_color.hpp>	315
Header <boost/svg_plot/svg_style.hpp>	321
Header <boost/svg_plot/uncertain.hpp>	346
Using Inkscape	356
Implementation, History & Rationale	362
History	362
Compilers and Examples	362
Implementation and other notes	363
To Do List	368
Acknowledgements	369
Class Index	369
Function Index	372
Index	399

Preface

Preface

Humans have a fantastic capacity for visual understanding, and merely looking at data organized in one, two, or three dimensions allows us to see relations not otherwise visible in a list of numbers. Computers, however, deal with information numerically, and C++ and the [Standard Template Library \(STL\)](#) do not currently offer an easy way to bridge the gap.

This library aims to help the C++ programmer to easily plot data that is stored in STL containers.

This project is focused on using STL containers in order to display data on a one-dimensional, two-dimensional plot and boxplots. The plot is written to a [Scalable Vector Graphic SVG image](#). (SVG plots can also be converted to other graph types, for example, [Portable Network Graphics PNG](#)).

[Scalable Vector Graphics \(SVG\)](#) is an [XML specification](#) and file format for describing two-dimensional vector graphics.

SVG files (.svg) can be viewed with most Internet Browsers:

- [Mozilla Firefox \(version 3 and later\)](#) You may wish to associate Firefox with SVG files (or perhaps Inkscape - see below).
- [Adobe Illustrator](#).
- [Google Chrome browser](#).
- [Opera](#) has good [SVG support](#).
- Microsoft Internet Explorer IE9 and later, but the quality of rendering before IE11 is poor compared to all other browsers and cannot yet be fully recommended. Earlier versions of Microsoft Internet Explorer can adequately render SVG files provided a suitable [Adobe SVG Viewer plug-in for SVG files](#) is installed. (Adobe have stopped offering this for download (as they warned years ago) but the download software can still be obtained elsewhere, for example from [software.informer](#). Information about limitations of IE with Vista and the Adobe add-in are given on the [Adobe End of Line FAQ](#)).
- [Inkscape](#), a fine Open Source SVG editor/viewer with excellent rendering, full scaling and other editing features. Inkscape is the 'Gold Standard' for viewing and creating and editing SVG files.

Inkscape also allows window resizing, showing the scalability of SVG files without any loss of quality, unlike bit image files that eventually reveal pixels as you expand the view. See [Using Inkscape](#) for details of how to use Inkscape to edit your plot and convert to other format like Portable Network Graphics (PNG) file type .png.

(Inkview, a view-only version, has been discontinued in favour of making the SVG file readonly and using Inkscape).

- Many other graphics programs, for example see [Most popular SVG software](#).

The goals of the project are:

- To let users produce simple plots with minimal intervention by using sane defaults.
- To allow users to easily customize plots but allow very fine-grained control of appearance.
- To provide very high quality plots suitable for publication, including printing, but with tiny file sizes.
- To produce and transfer plots quickly enough to be useful in real-time.
- To represent uncertainty visually and numerically by showing only significant digits, and optionally adding uncertainty and degrees of freedom estimates.
- To allow the user to talk to the plot classes using coordinate units rather than pixels or other arbitrary units.
- To create the backbone of a `svg` class that can be extended to fully support the SVG standard.

- Compliance with the [W3C Scalable Vector Graph standard](#).
- Validation using [W3C Markup Validation Service](#).
- Copyright and license conditions built into the SVG file.

How To Use This Documentation

- Tutorials are listed in the Table of Contents and include many examples that should help you get started quickly.
- Source code of the many Examples will often be your quickest option. They often deliberately use many features, often producing examples of outstandingly bad taste!
- Reference section prepared using Doxygen will help fine-tuning the appearance of your graphs.
- Several indexes (especially the function index) will also help you find which of the several hundred options you need.
- If you have a feature request, or if it appears that the implementation is in error, please check the TODO page first, as well as the rationale section.

If you do not find your idea/complaint, please reach me either through the Boost development list, or email me direct at pbristow (at) htp (dot) u-net (dot) com or jakevoytko (at) gmail (dot) com.

Admonishments



Note

These blocks typically go into more detail about an explanation given above.



Tip

These blocks contain information that you may find helpful while coding.



Important

These contain information that is imperative to understanding a concept. Failure to follow suggestions in these blocks will probably result in undesired behavior. Read all of these you find.



Warning

It is imperative that you follow these. Failure to do so will lead to incorrect, and very likely undesired, results in the plot.

Colors

The project supports any [RGB color](#), as well as a number of colors (for example, red, pink, aliceblue...) that are [named by the SVG standard](#).

[svg_color_constant](#)

`svg_color_constant` is simply an enumerated list of these names colors.

The constants are defined at [svg_color.hpp](#).

Examples of colors are at [svg_colors.cpp](#).

The list of [SVG standard named colors](#) contains all the colors you might expect, such as red, blue, green, magenta, cyan, yellow, pink and orange, as well as nearly 150 other shades.

You will probably still find that the colors names do not meet your requirements and so it is also possible to define any RGB combination.

The list also contains one extra color element, `blank`, defined as `false`, used when you need to pass a color, but would not like it to be visible. This comes in handy for defining defaults for functions, for example.

[Example of using svg_color_constant](#)

```
using namespace boost::svg; // Convenient to use all SVG plot features.

svg_2d_plot my_plot; // Create a 2D plot (with all defaults).

svg_color_constant my_color = red; // Choose a color,
my_plot.background_border_color(my_color); // and apply to the border of the entire image.
my_plot.background_color(lightgray); // Or use directly, to set the color of the background.
```



Note

`svg_color` has a constructor for `svg_color_constant`, so you can use a `svg_color_constant` in place of a `svg_color` and it will be implicitly converted. However, there is **not** currently an `svg_color::operator=(svg_color_constant)` overload, so

```
svg_color my_color = red;
```

does not have the desired effect, and nor does `cout << red << endl;` output the expected "RGB(255,0,0)" but instead outputs "119", the value of the enum!

[svg_color interface](#)

You can define a `svg_color` using two different constructors:

```
svg_color(int, int, int); // The parameters are red, green, and blue respectively.
svg_color(svg_color_constant); // Use a pre-existing color constant, like red.
```

Example of using `svg_color`

```
using namespace boost::svg;

svg_color my_white(255, 255, 255); // Using RGB svg_color constructor.
svg_color my_const_white(white); // Using SVG-defined named constructor.

BOOST_ASSERT(my_white == my_const_white);
```



Note

You will find it convenient to use the namespace `boost::svg` (perhaps only at local scope) to avoid having to fully qualify **every** color that you want to use, for example, to avoid writing: using `boost::svg::azure`;



Note

`svg_color`'s constructor takes in three integer values. The [SVG 1.1 standard](#) allows any integer to represent an RGB value, with values less than 0 and greater than 255 being constrained to their respective min and max.

Colors Internals and Rationale

Color constants are defined in an enum, `svg_color_constant`, in alphabetical order at [color constants](#). This facilitates quick lookup of their RGB values from an array. Anywhere that a `svg_color` can be used, a `svg_color_constant` can be used, as the conversion is implicit.

All color information is stored in RGB format as three `unsigned char` in a `svg_color` struct. The rationale for storing information in RGB format is that it is precise, and is always representable the exactly the same way. Storing as a floating-point percent or fraction would introduce the possibility of undesirable rounding error.

Fonts

Examples of fonts are at [demo_2d_fonts.cpp](#).

Fine control of font family and size is provided, although the exact rendering may be limited by the browser used to view the .svg file.

An example is provided to demonstrate various fonts and fonts sizes.

The conventional wisdom of sticking to one or two fonts are deliberately broken to show various fonts and sizes that are available for SVG plots. The result is a graphic designers nightmare!

A font family may or may not be available for a particular internet browser, so it is inevitable that the exact appearance of a SVG plot may vary when viewed with different browsers. If a font family is not recognised, then a default (for that browser) will be used instead.

The font families available for browsers do not seem to be listed, but those used below are available on Mozilla Firefox 3.5. The example below looks very similar with the Adobe SVG Add-in with Microsoft Internet Explorer 8 (which does NOT render SVG natively at all), and with Inkscape and Opera.

Mozilla Firefox 3.5 allows one to add font(s), but how these are rendered with SVG is not so clear.

For most purposes the default Font family Verdana looks fine.

The following font families work with Firefox 3.5:

```
"arial", "impact", "courier", "lucida console", "Lucida sans unicode",
"verdana", "calibri", "century", "lucida calligraphy", "tahoma", "vivaldi"
"informal roman", "lucida handwriting", "lucida bright", "helvetica"
"arial narrow" is narrow, so may be useful to fit a long title or label.
"arial black" is black!
```

These do NOT work and are substituted:

```
"comic sans", "sans" "bauhaus" "brush script" "segeo condensed" = Serif
```

The narrow, wide, bold and italic features produce rather variable and unpredictable results - the rendering may be 'fuzzy' or ill-formed: so these are not recommended. For example,

```
"Times New Roman Bold" "Times New Roman Italic" are substituted by "Times New Roman"
```

But to get narrow characters "arial narrow" works well, squeezing in a longer title or label.

The font sizes are also changes from the defaults. This should change the positioning, but the calculations are complex and necessarily approximate. Collisions between labels, other value labels and axes are not impossible, especially when the tick value labels are not horizontal.

By default, the std::precision is reduced from the default 6 to 3, and unnecessary zeros and signs are stripped.

But it will still often be necessary to change the std::iosflags and std::precision, and/or the number of major ticks and/or font size and type to avoid tick value label collisions.

Unicode symbols can be found at [Unicode symbols](#). The 4 hex digit value needs to be wrapped with prefix &#x and suffix ; like �. Rendering of Unicode symbols is not entirely predictable, but usually works well to provide a wide range of greek and math symbols.

The code below shows plotting the sqrt function selecting the range of the axis by a user choice.

```
{
    svg_2d_plot my_plot; // Construct a 2D plot.

    my_plot.legend_on(true) // Note chaining.
        .title("Function ") // Unicode sqrt symbol.
        .title_font_size(35)
        .title_font_family("arial black")

        .legend_title("Legend title")
        .legend_header_font_size(15)
        .legend_font_family("lucida calligraphy")
        .legend_color(cyan)

        .x_range(0, +20.)
        .x_major_interval(2.)
        .x_num_minor_ticks(4) // MAJOR, minor, minor, minor, minor, MAJOR
        .x_label("abcd1234")
        .x_axis_label_color(green)
        .x_label_font_family("helvetica")
        .x_label_font_size(40)
        .x_ticks_values_color(red) //
        .x_ticks_values_font_family("Times New Roman")
        .x_ticks_values_font_size(14)
        .x_ticks_values_precision(0)
        .x_ticks_values_ioflags(ios_base::fixed)

        .y_label("sqrt(x) or (&#x221A;x)")
        .y_range(0., 5.)
        .y_ticks_values_color(magenta)
        .y_ticks_values_precision(1)
        .y_ticks_values_ioflags(ios_base::scientific | ios_base::showpos)
        .y_ticks_values_font_family("Lucida sans unicode")
        .y_ticks_values_font_size(20)
        // .y_label_font_family("informal roman")
        .y_label_font_family("Times New roman")
        .y_label_font_size(40)
        .y_axis_label_color(blue)
    ;

    // Add a container of data to the plot, choosing a color.
    my_plot.plot(data1, "Function (&#x221A;)").stroke_color(red).shape(circlet).size(3).bezier_on(true).line_col□
or(pink);

    my_plot.write("./demo_2d_fonts_1.svg"); // Write 1st plot to file.
    // Show some styling, for example for X ticks.
    cout << "my_plot.x_ticks_values_color() " << my_plot.x_ticks_values_color() << endl;
    cout << "my_plot.x_ticks_values_font_family() " << my_plot.x_ticks_values_font_family() << endl;
    cout << "my_plot.x_ticks_values_font_size() " << my_plot.x_ticks_values_font_size() << endl;
    cout << "my_plot.x_ticks_values_precision() " << my_plot.x_ticks_values_precision() << endl;
    cout << "my_plot.x_ticks_values_ioflags() 0x" << hex << my_plot.x_ticks_values_ioflags() << dec << endl;
}

// Axis label rotation default is horizontal.
{
    svg_2d_plot my_plot; // Construct another 2D plot to try other variations of font sizes.
    my_plot.x_range(0, +20.)
        .y_range(0., 5.)
        .x_label("X axis label font default size 14")
        .y_label("Y axis label font default size 14")
        // .x_label_font_size(10)
        // .y_label_font_size(10)
    ;
    my_plot.plot(data1, "Function (&#x221A;)").stroke_color(red).shape(circlet).size(10).line_on(false).line_col□
or(green);
}
```

```

    my_plot.write("./demo_2d_fonts_1.svg"); // Write another plot to file.
}
{
    svg_2d_plot my_plot; // Construct another 2D plot to try other variations of font sizes.
    my_plot.x_range(0, +20.)
        .y_range(0., 5.)
        .x_label("x (small X axis label font size 10)")
        .y_label("y (small X axis label font size 10)")
        .x_label_font_size(10)
        .y_label_font_size(10);
    my_plot.plot(data1, "Function (&#x221A;)").stroke_color(red).shape(circlet).size(10).line_on(false).line_col□
or(green);
    my_plot.write("./demo_2d_fonts_2.svg"); // Write another plot to file.
}
{
    svg_2d_plot my_plot; // Construct another 2D plot to try other variations of font sizes.
    my_plot.x_range(0, +20.)
        .y_range(0., 5.)
        .x_label("X axis label 30")
        .y_label("Y axis label 30")
        .x_label_font_size(30)
        .y_label_font_size(30);
    my_plot.plot(data1, "Function (&#x221A;)").stroke_color(red).shape(circlet).size(10).line_on(false).line_col□
or(green);
    my_plot.write("./demo_2d_fonts_3.svg"); // Write another plot to file.
}
{
    svg_2d_plot my_plot; // Construct another 2D plot to try other variations of font sizes.
    my_plot.x_range(0, +20.)
        .y_range(0., 5.)
        .x_label("x (large tick font size 20)")
        .y_label("y (large tick font size 20)")
        .x_label_font_size(10)
        .y_label_font_size(10)
        .x_ticks_values_font_size(20)
        .y_ticks_values_font_size(20)

    ;
    my_plot.plot(data1, "Function (&#x221A;)").stroke_color(red).shape(circlet).size(10).line_on(false).line_col□
or(green);
    my_plot.write("./demo_2d_fonts_4.svg"); // Write another plot to file.
}
// Now alter the rotation of the axis labels.
{
    svg_2d_plot my_plot; // Construct another 2D plot to try other variations of font sizes.
    my_plot.x_range(0, +20.)
        .y_range(0., 5.)
        .x_label("X axis label font default size 14")
        .y_label("Y axis label font default size 14")
        .x_major_label_rotation(uphill)
        .y_major_label_rotation(uphill)
        // .x_label_font_size(10)
        // .y_label_font_size(10)
        ;
    my_plot.plot(data1, "Function (&#x221A;)").stroke_color(red).shape(circlet).size(10).line_on(false).line_col□
or(green);
    my_plot.write("./demo_2d_fonts_5.svg"); // Write another plot to file.
}
{
    svg_2d_plot my_plot; // Construct another 2D plot to try other variations of font sizes.
    my_plot.x_range(0, +20.)
        .y_range(0., 5.)
        .x_label("x (small X axis label font size 10)")

```

```

.y_label("y (small X axis label font size 10)")
.x_label_font_size(10)
.y_label_font_size(10)
.x_major_label_rotation(uphill)
.y_major_label_rotation(uphill)
;

my_plot.plot(data1, "Function (&#x221A;)").stroke_color(red).shape(circlet).size(10).line_on(false).line_col□
or(green);
my_plot.write("./demo_2d_fonts_6.svg"); // Write another plot to file.
}

{
svg_2d_plot my_plot; // Construct another 2D plot to try other variations of font sizes.
my_plot.x_range(0, +20.)
.y_range(0., 5.)
.x_label("X axis label 30")
.y_label("Y axis label 30")
.x_label_font_size(30)
.y_label_font_size(30)
.x_major_label_rotation(uphill)
.y_major_label_rotation(uphill)
;
my_plot.plot(data1, "Function (&#x221A;)").stroke_color(red).shape(circlet).size(10).line_on(false).line_col□
or(green);
my_plot.write("./demo_2d_fonts_7.svg"); // Write another plot to file.
}

{
svg_2d_plot my_plot; // Construct another 2D plot to try other variations of font sizes.
my_plot.x_range(0, +20.)
.y_range(0., 5.)
.x_label("x tick size 12, label 14")
.y_label("y tick size 12, label 14")
.x_label_font_size(14)
.y_label_font_size(14)
.x_ticks_values_font_size(12)
.y_ticks_values_font_size(12)
.x_major_label_rotation(uphill)
.y_major_label_rotation(uphill)
;
my_plot.plot(data1, "Function (&#x221A;)").stroke_color(red).shape(circlet).size(10).line_on(false).line_col□
or(green);
my_plot.write("./demo_2d_fonts_8.svg"); // Write another plot to file.
}

{
svg_2d_plot my_plot; // Construct another 2D plot to try other variations of font sizes.
my_plot.x_range(0, +20.)
.y_range(0., 5.)
.x_label("X axis label font default size 14")
.y_label("Y axis label font default size 14")
.x_major_label_rotation(downward)
.y_major_label_rotation(upward)
;
my_plot.plot(data1, "Function (&#x221A;)").stroke_color(red).shape(circlet).size(10).line_on(false).line_col□
or(green);
my_plot.write("./demo_2d_fonts_9.svg"); // Write another plot to file.
}

{
svg_2d_plot my_plot; // Construct another 2D plot to try other variations of font sizes.
my_plot.x_range(0, +20.)
.y_range(0., 5.)
.x_label("x (small X axis label font size 10)")

```

```

.y_label("y (small X axis label font size 10)")
.x_label_font_size(10)
.y_label_font_size(10)
.x_major_label_rotation(stEEPDown)
.y_major_label_rotation(stEEPUp)
;

my_plot.plot(data1, "Function (\u221A)").stroke_color(red).shape(circlet).size(10).line_on(false).line_col□
or(green);
my_plot.write("./demo_2d_fonts_10.svg"); // Write another plot to file.
}
{
svg_2d_plot my_plot; // Construct another 2D plot to try other variations of font sizes.
my_plot.x_range(0, +20.)
.y_range(0., 5.)
.x_label("X axis label 30")
.y_label("Y axis label 30")
.x_label_font_size(30)
.y_label_font_size(30)
.x_major_label_rotation(downhill)
.y_major_label_rotation(uphill)
;
my_plot.plot(data1, "Function (\u221A)").stroke_color(red).shape(circlet).size(10).line_on(false).line_col□
or(green);
my_plot.write("./demo_2d_fonts_11.svg"); // Write another plot to file.
}
{
svg_2d_plot my_plot; // Construct another 2D plot to try other variations of font sizes.
my_plot.x_range(0, +20.)
.y_range(0., 5.)
.x_label("x tick size 12, label 14")
.y_label("y tick size 12, label 14")
.x_label_font_size(14)
.y_label_font_size(14)
.x_ticks_values_font_size(12)
.y_ticks_values_font_size(12)
.x_major_label_rotation(slopedownhill)
.y_major_label_rotation(slopeup)
;
my_plot.plot(data1, "Function (\u221A)").stroke_color(red).shape(circlet).size(10).line_on(false).line_col□
or(green);
my_plot.write("./demo_2d_fonts_12.svg"); // Write another plot to file.
}
}
catch(const std::exception& e)
{
std::cout <<
"\n""Message from thrown exception was:\n " << e.what() << std::endl;
}
return 0;
} // int main()

```

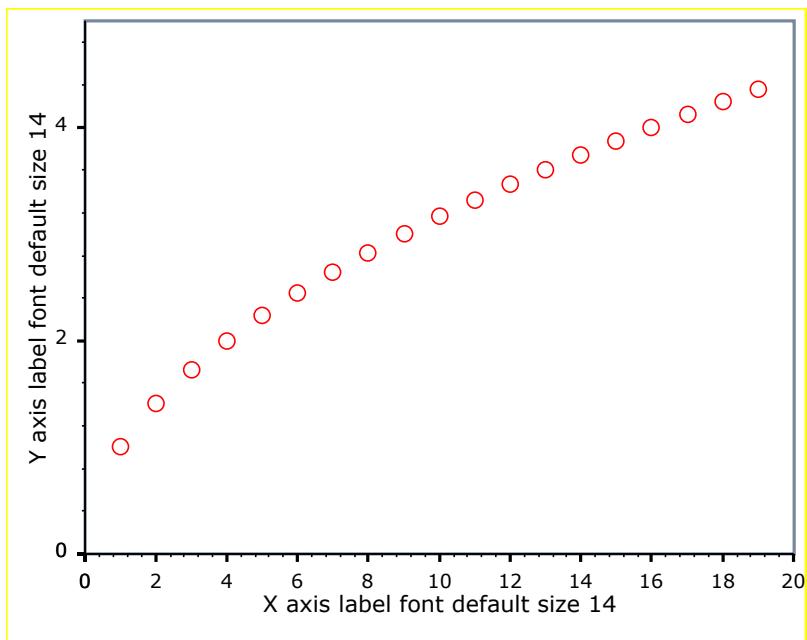
Output:

```

Autorun 'j:\Cpp\SVG\Debug\demo_2d_fonts.exe'
my_plot.x_ticks_values_color() RGB(255,0,0)
my_plot.x_ticks_values_font_family() Verdana
my_plot.x_ticks_values_font_size() 12
my_plot.x_ticks_values_precision() 0
my_plot.x_ticks_values_ioflags() 0x2000

```

Producing this ugly plot, as an example from the files demo_2d_fonts_1.svg to demo_2d_fonts_12.svg:



See [demo_2d_fonts.cpp](#) for full source code.

1D Tutorials

This section gives examples of plotting 1-dimensional data held in, for example, in an array or vector.

1-D Vector Example

First we need a few includes to use Boost.Plot:

```
#include <boost/svg_plot/svg_1d_plot.hpp>
using namespace boost::svg;
#include <vector>
using std::vector;
```

STL vector is used as the container for our two data series, and values are inserted using push_back. Since this is a 1-D plot the order of data values is not important.

```
vector<double> dan_times;
dan_times.push_back(3.1);
dan_times.push_back(4.2);

vector<double> elaine_times;
elaine_times.push_back(2.1);
elaine_times.push_back(7.8);
```

The constructor initializes a new 1D plot, called my_plot, and also sets all the very many defaults for axes, width, colors, etc.

```
svg_1d_plot my_plot;
```

A few (member) functions that set are fairly self-explanatory:

- title provides a title at the top for the whole plot,
- legend_on(true) will mean that titles of data series and markers will display in the legend box.
- x_range(-1, 11) sets the axis limits from -1 to +11 (instead of the default -10 to +10).
- background_border_color(blue) sets just one of the very many options.

```
my_plot.background_border_color(blue)
.legend_on(true)
.title("Race Times")
.x_range(-1, 11);

my_plot.legend_lines(true);
```

The syntax my_plot.title("Hello").legend_on(true)... may appear unfamiliar, but is a convenient way of specifying many parameters in any order. It is equivalent to:

```
my_plot.title("Race Times");
my_plot.legend_on(true);
my_plot.x_range(-1, 11);
my_plot.background_border_color(blue);
```

Chaining thus allows you to avoid repeatedly typing "myplot." and easily group related settings like plot window, axes ... together. A fixed order would clearly become impracticable with hundreds of possible arguments needed to set all the myriad plot options.

Within all of the plot classes, 'chaining' works the same way, by returning a reference to the calling object thus return `*this`;

Then we need to add our data series, and add optional (but very helpful) data series titles if we want them to show on the legend.

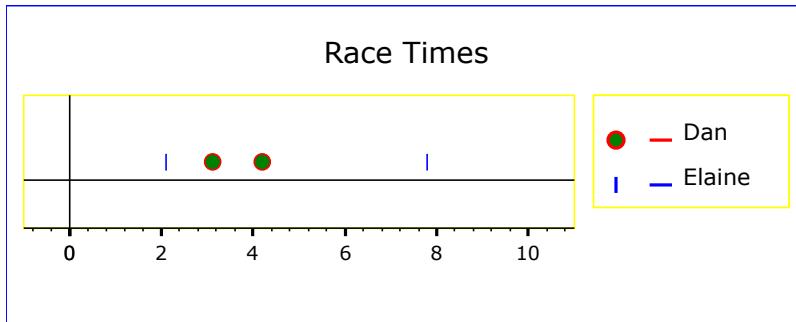
```
my_plot.plot(dan_times, "Dan").shape(circlet).size(10).stroke_color(red).fill_color(green);
my_plot.plot(elaine_times, "Elaine").shape(vertical_line).stroke_color(blue);
```

Finally, we can write the SVG to a file of our choice.

```
my_plot.write("./demo_1d_vector.svg");
```

The IDE output is not very exciting in this case

```
Compiling...
demo_1d_vector.cpp
Linking...
Embedding manifest...
Autorun "j:\Cpp\SVG\debug\demo_1d_vector.exe"
Build Time 0:04
```



but the plot is

(You can also view the SVG XML source using browsers (View, Source) or use your favorite text editor like Notepad, TextPad etc at [demo_1d_vector.svg](#)

See [demo_1d_vector.cpp](#) for full source code.

1-D STL Containers Examples

First a few includes to use Boost.Plot and various STL containers:

```
#include <boost/svg_plot/svg_1d_plot.hpp>
// using namespace boost::svg;
#include <boost/array.hpp>
using boost::array;
#include <vector>
using std::vector;
#include <set>
using std::set; // Automatically sorted - though this is not useful to the plot process.
using std::multiset; // At present using std::multiset, allowing duplicates, plot does not indicate duplicates.
// With 2_D data in std::multimap, duplicate values are usefully displayed.
#include <list>
using std::list;
#include <deque>
using std::deque;
```

STL vector is used as the container, and fictional values are inserted using `push_back`. Since this is a 1-D plot the order of data values is not important.

```
std::vector<float> values;
values.push_back(3.1f);
values.push_back(-5.5f);
values.push_back(8.7f);
values.push_back(0.5f);
```

The constructor initializes a new 1D plot, called `my_plot`, and also sets all the many default values.

```
using namespace boost::svg;
boost::svg::svg_1d_plot my_plot;
```

Setting (member) functions are fairly self-explanatory: `Title` provides a title at the top for the whole plot, and `plot` adds a (unnamed) data series (naming isn't very useful if there is only one data series).

```
my_plot.title("vector<float> example");
```



Note

One must insert the XML character entity equivalents of < for < and > for >).

```
my_plot.plot(values);
```

Write the SVG to a file.

```
my_plot.write("./demo_1d_vector_float.svg");
```

If the container is a static array, then it must be filled by assignment:

```
boost::array<long double, 4> values = {3.1L, -5.5L, 8.7L, 0.5L};

boost::svg::svg_1d_plot my_plot;
my_plot.title("array<long double> example");
my_plot.plot(values);
my_plot.write("./demo_1d_array_long_double.svg");
```

If the container type is a set, then it can be filled with `insert`:

```
std::set<double> values;
values.insert(-8.4);
values.insert(-2.3);
values.insert(0.1);
values.insert(5.6);
values.insert(7.8);

boost::svg::svg_1d_plot my_plot;
my_plot.title("set<double> example");
my_plot.plot(values);
my_plot.write("./demo_1d_set_double.svg");
```

If the container type is a list, then it can be filled with `push_back` or `push_front`:

```
std::list<double> values;
values.push_back(-8.4);
values.push_back(-2.3);
values.push_back(0.1);
values.push_back(5.6);
values.push_back(7.8);
boost::svg::svg_1d_plot my_plot;
my_plot.title("list<double> example");
my_plot.plot(values);
my_plot.write("./demo_1d_list_double.svg");
```

If the container type is a deque, then it can be filled with push_back or push_front:

```
std::deque<double> values;
values.push_front(-8.4);
values.push_front(-2.3);
values.push_front(0.1);
values.push_front(5.6);
values.push_front(7.8);

boost::svg::svg_1d_plot my_plot;
my_plot.title("deque<double> example");
my_plot.plot(values);
my_plot.x_label("X values as doubles");

my_plot.write("./demo_1d_deque_double.svg");
```



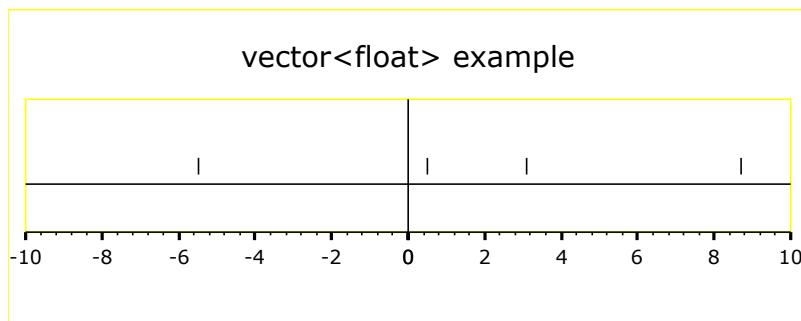
Note

For filling STL containers you may find the [Boost.Assign](#) library by Thorsten Ottosen useful.

The IDE output is not very exciting in this case:

```
Compiling...
demo_1d_containers.cpp
Linking...
Embedding manifest...
Autorun "j:\Cpp\SVG\debug\demo_1d_containers.exe"
Build Time 0:03
```

The plot is:



And you can view the other svg files (with most internet browsers, and other programs too) for example:

- [demo_1d_array_long_double.svg](#)

- [demo_1d_set_double.svg](#)
- [demo_1d_list_double.svg](#)
- [demo_1d_deque_double.svg](#)

See [demo_1d_containers.cpp](#) for full source code.

Tutorial: 1D Autoscale with Multiple Containers

This example demonstrates autoscaling with **multiple** STL containers.

First we need a few includes to use Boost.Plot (and some others only needed for this example).

```
#include <boost/svg_plot/svg_1d_plot.hpp>
using namespace boost::svg;
using boost::svg::svg_1d_plot;

#include <boost/svg_plot/detail/auto_axes.hpp>
using boost::svg::show; // A single STL container.
using boost::svg::show_all; // Multiple STL containers.
//using boost::svg::range; // Find min and max of a single STL container.
using boost::svg::range_all; // Find min and max of multiple STL containers.
// Note neither of these check for 'limits' (infinite, NaN) values.

#include <boost/svg_plot/detail/pair.hpp>
// using boost::svg::detail::operator<<; // Output pair as, for example: 1.23, 4.56

//#include <boost/svg_plot/show_1d_settings.hpp>
// Only needed for showing which settings in use.
//void boost::svg::show_1d_plot_settings(svg_1d_plot&);

#include <boost/algorithm/minmax.hpp>
using boost::minmax;
#include <boost/algorithm/minmax_element.hpp>
using boost::minmax_element;

#include <iostream>
using std::cout;
using std::endl;
using std::boolalpha;

#include <limits>
using std::numeric_limits;

#include <vector>
using std::vector;

#include <utility>
using std::pair;

#include <algorithm>
using std::multiplies;
using std::transform;
using std::copy;
#include <iterator>
using std::iterator;
using std::iterator_traits;
using std::ostream_iterator;
```

This example uses two containers to demonstrate autoscaling. It is common to plot more than one set of data series together. Autoscaling must probably inspect all the containers of these series in order to find axis ranges that will be **suitable for all of them**.

```
vector<double> my_data_1;
// Initialize STL container my_data_1 with some entirely fictional data.
my_data_1.push_back(0.2); // [0]
my_data_1.push_back(1.1); // [1]
my_data_1.push_back(4.2); // [2]
my_data_1.push_back(3.3); // [3]
my_data_1.push_back(5.4); // [4]
my_data_1.push_back(6.5); // [5]
```

We might use some convenient functions to list the container to cout.

```
show(my_data_1); // Entire container contents.
```

Others are easily written, often using std::copy, for example:

```
copy(my_data_1.begin(), my_data_1.end(), ostream_iterator<double>(cout, " "));
```

Now we concoct another equally fictional data series by a transform multiplying by 2.3.

```
vector<double> my_data_2; // Create a second data series.
copy(my_data_1.begin(), my_data_1.end(), back_inserter(my_data_2));
// Change the values in an entirely arbitrary way (each * 2.3).
transform(my_data_2.begin(), my_data_2.end(), my_data_2.begin(), bind1st(multiplies<double>(), 2.3));
//cout << endl << my_data.size() << " values in my_data_2. " << endl;
```

Next we need a new STL container, vector say, to hold our multiple containers of data series. They must all be the same STL container type, in this example, `vector<double>`. And we use pushback to add the containers.

```
vector<vector<double> > my_containers;

my_containers.push_back(my_data_1); // Add 1st data series.
my_containers.push_back(my_data_2); // Add another data series.
cout << my_containers.size() << " containers." << endl;
show_all(my_containers);
```

Finally we can use all the containers to find the minimum of mimimums and maximum of maximums ready to feed into the plot autoscale function.

```
pair<double, double> mm = range_all(my_containers);
cout << "Data range: " << mm << endl; // min & max of all data.
svg_1d_plot my_1d_plot; // Construct a plot with all the default constructor values.
```

We could feed the minimum and maximum values separately,

```
my_1d_plot.x_autoscale(mm.first, mm.second); // Use minimum and maximum to autoscale.
```

but usually feeding the pair is more convenient. (Perhaps we might want to impose some other minimum and maximum here).

```
my_1d_plot.x_autoscale(mm); // Use overall minimum and maximum to autoscale.
```

Finally, we add the data series containers to the plot, and write the SVG out to file.

```
my_1d_plot.plot(my_data_1, "data_1");
my_1d_plot.plot(my_data_2, "data_2").stroke_color(red);

my_1d_plot.write("auto_1d_containers.svg"); // Write the plot to file.
```

If we want, we can check the autoscale range used.

```
using boost::svg::detail::operator<<; // To avoid ambiguity.
cout << "x_range() " << my_1d_plot.x_range() << endl; // x_range() 0, 15
```

And even all the (hundreds of) plot settings (useful for diagnosis why your plot doesn't meet your expectations).

```
//show_1d_plot_settings(my_1d_plot);
```



Warning

The containers must be of the same type to use the function range_all. If different types of containers, for example some in a set and some in a vector, then the min and max for each container must be computed separately and the minimum of the minimums and the maximum of the maximums injected into the x_autoscale (and/or y_autoscale) call.

Typical output is:

```
Compiling...
auto_1d_containers.cpp
Linking...
Embedding manifest...
Autorun "J:\Cpp\SVG\debug\auto_1d_containers.exe"
6 values in container: 0.2 1.1 4.2 3.3 5.4 6.5
2 containers.
6 values in container: 0.2 1.1 4.2 3.3 5.4 6.5
6 values in container: 0.46 2.53 9.66 7.59 12.42 14.95
Data range: 0.2, 14.9
x_range() 0, 15
Build Time 0:03
```

with unc class

```
Description: Autorun "J:\Cpp\SVG\Debug\auto_1d_containers.exe"
6 values in container: 0.2 1.1 4.2 3.3 5.4 6.5
2 containers.
6 values in container: 0.2 1.1 4.2 3.3 5.4 6.5
6 values in container: 0.46 2.53 9.66 7.59 12.42 14.95
Data range: <0.2, 14.95>
x_range() 0, 15
```

See [auto_1d_containers.cpp](#) for full source code and sample output.

Tutorial: 1D More Layout Examples

This section shows a few more of the near-infinite layout options. See also the reference section and function index.

```

#include <boost/svg_plot/svg_1d_plot.hpp>
using namespace boost::svg;
#include <cmath> // for sqrt

#include <vector>
#include <deque>
#include <boost/array.hpp>

// Three different STL-style containers:
using std::vector;
using std::deque;
using boost::array;

// Three sample functions:
double f(double x)
{
    return sqrt(x);
}

double g(double x)
{
    return -2 + x*x;
}

double h(double x)
{
    return -1 + 2*x;
}

int main()
{
    vector<double> data1;
    deque<double> data2;
    boost::array<double, 10> data3;
    // Fill the containers with some data using the functions:
    int j=0;
    for(double i=0; i<9.5; i+=1.)
    {
        data1.push_back(f(i));
        data2.push_front(g(i));
        data3[j++] = h(i);
    }

    svg_1d_plot my_plot; // Construct my_plot.

    // Set many plot options,
    // using chaining to group similar aspects together.
    // Size/scale settings.
    my_plot.image_size(500, 350)
        .x_range(-3, 10);

    // Text settings.
    my_plot.title("Oh My!")
        .title_font_size(29)
        .x_label("Time in Months") // X- axis label
        .x_label_on(true); // Not needed, because although the default is no axis label,
                           // providing a label has this effect.
        // If we want later to switch *off* labels,
        // my_plot.x_label_on(false); // is needed.

    // Commands.
    my_plot.legend_on(true)
        .plot_window_on(true)

```

```

.x_major_labels_on(true);

// Color settings.
my_plot.background_color(svg_color(67, 111, 69))
    .legend_background_color(svg_color(207, 202, 167))
    .legend_border_color(svg_color(102, 102, 84))
    .plot_background_color(svg_color(136, 188, 126))
    .title_color(white);

// Axis settings.
my_plot.x_major_interval(2)
    .x_major_tick_length(14)
    .x_major_tick_width(1)
    .x_minor_tick_length(7)
    .x_minor_tick_width(1)
    .x_num_minor_ticks(3);

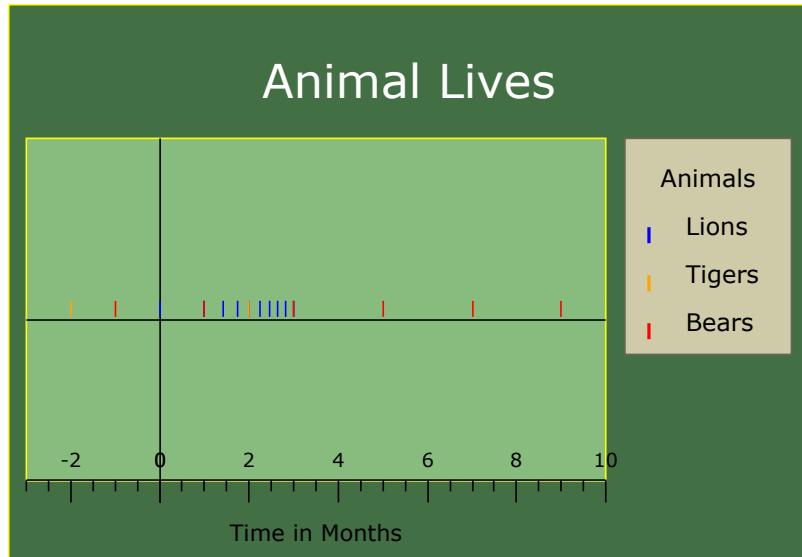
// Legend settings.
my_plot.legend_title_font_size(15);

// Put the three containers of data onto the plot.
my_plot.plot(data1, "Lions", blue);
my_plot.plot(data2, "Tigers", orange);
my_plot.plot(data3, "Bears", red);

my_plot.write("1d_full_layout.svg");

return 0;
} // int main()

```



This produces the following plot:

A little bit of color customization goes a **very** long way!

Tutorial: 1D Gridlines & Axes - more examples

It is possible to change the location of the intersection of the axes, and the labelling - at the top, bottom or in the middle on the Y axis. This is controlled using the enum `x_axis_intersect` and function `x_ticks_on_window_or_axis`.

Following previous examples, we set up two containers for two data series.

```

vector<double> dan_times;
vector<double> elaine_times;

dan_times.push_back(3.1);
dan_times.push_back(4.2);
elaine_times.push_back(2.1);
elaine_times.push_back(7.8);

svg_1d_plot my_plot;

// Adding some generic settings.
my_plot.background_border_color(black)
    .legend_on(true)
    .plot_window_on(true)
    .title("Race Times")
    .x_range(-1, 10);

```

We add tastelessly color the grids for both major and minor ticks, and switch both grids on.

```

my_plot.x_major_grid_color(pink)
    .x_minor_grid_color(lightgray);

my_plot.x_major_grid_on(true)
    .x_minor_grid_on(true);

```

Also we specify the position of the labelling of the X-axis. It can be controlled using values in the enum `x_axis_intersect`.

```

enum x_axis_intersect
{ //! \enum x_axis_intersect
    bottom = -1, // On the bottom of the plot window.
    x_intersects_y = 0, // On the Y axis (in the middle of the plot window).
    top = +1 // On the top of the plot window.
};

```

For this example, we choose to show the X axis and tick value labels at the top of the plot window.

```

my_plot.x_ticks_on_window_or_axis(top); // on top, not on axis.

// Write to plot.
my_plot.plot(dan_times, "Dan").stroke_color(blue);
my_plot.plot(elaine_times, "Elaine").stroke_color(orange);

// Write to file.
my_plot.write("./demo_1d_x_external.svg");
return 0;
} // int main()

```

X-Axis Grid Lines

If you would like vertical grid lines that go on the graph, you can make the following call to `svg_1d_plot`:

```

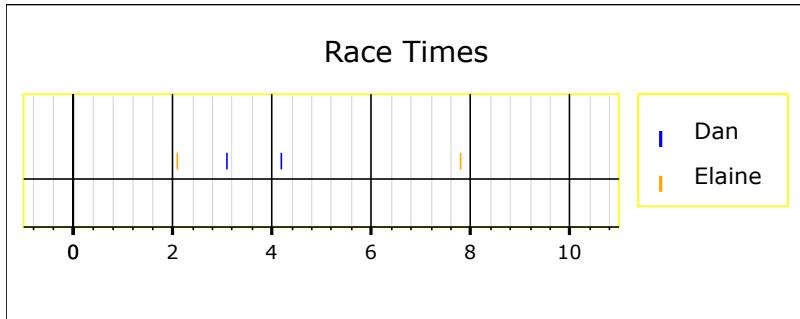
my_plot.x_major_grid_on(true)
    .x_minor_grid_on(true);

```

To color style it, you might add the following calls:

```
my_plot.x_major_grid_color(lightgray) // Darker color for major grid.  
.x_minor_grid_color(whitesmoke); // Lighter color for minor grid.
```

This will produce the following image:



X-Axis Tick value labels precision, iosflags, font family and size

An example to demonstrate some options for controlling the layout and color of tick values.

The code below shows plotting the sqrt function selecting the range of the axis by a user choice.



Note

Unicode symbols can be found at http://en.wikipedia.org/wiki/Unicode_symbols. The 4 hex digit value needs to be wrapped with prefix &#x and suffix ; like �

```

svg_2d_plot my_plot; // Construct a 2D plot.

my_plot.legend_on(true) // Set title and legend, and X and Y axis range.
    .title("√ Function") // Unicode sqrt symbol.
    .x_range(0, +20.)
    .x_major_interval(2.)

    .x_axis_label_color(green)
    .x_label_font_family("helvetica")
    .x_label_font_size(30)

    .x_num_minor_ticks(4) // MAJOR, minor, minor, minor, minor, MAJOR
    .x_ticks_values_color(red) //
    .x_ticks_values_font_family("Times New Roman")
    .x_ticks_values_font_size(20)
    .x_ticks_values_precision(0)
    .x_ticks_values_ioflags(ios_base::fixed)

    .y_range(0., 5.)
    .y_ticks_values_color(magenta)
    .y_ticks_values_precision(1)
    .y_ticks_values_ioflags(ios_base::scientific | ios_base::showpos)

    // "arial", "impact", "courier", "lucida console", "Lucida sans unicode", "verdana", "calibri", "century", "lucida calligraphy", "tahoma", "vivaldi"
    // "informal roman", "lucida handwriting", "lucida bright", "helvetica"
    // "arial narrow" is narrow, so may be useful.
    // "arial black" is black!
    // "Times New Roman Bold" "Times New Roman italic" = Times New Roman
    // "comic sans", "sans" "bauhaus" "brush script" "segeo condensed" = Serif

    .y_ticks_values_font_family("lucida console")
    .y_ticks_values_font_size(10)

    .y_label_font_family("Times New Roman")
    .y_label_font_size(30)
    .y_axis_label_color(blue)
;

my_plot.x_label("x abcd1234(√)").y_label("sqrt(x)"); // Note chaining.

// Add a container of data to the plot, choosing a color.
my_plot.plot(data1, "Function (√)").stroke_color(red).shape(circlet).size(3).bezier_on(true).line_color(pink);

my_plot.write("./demo_2d_tick_values.svg"); // Write the plot to another file.

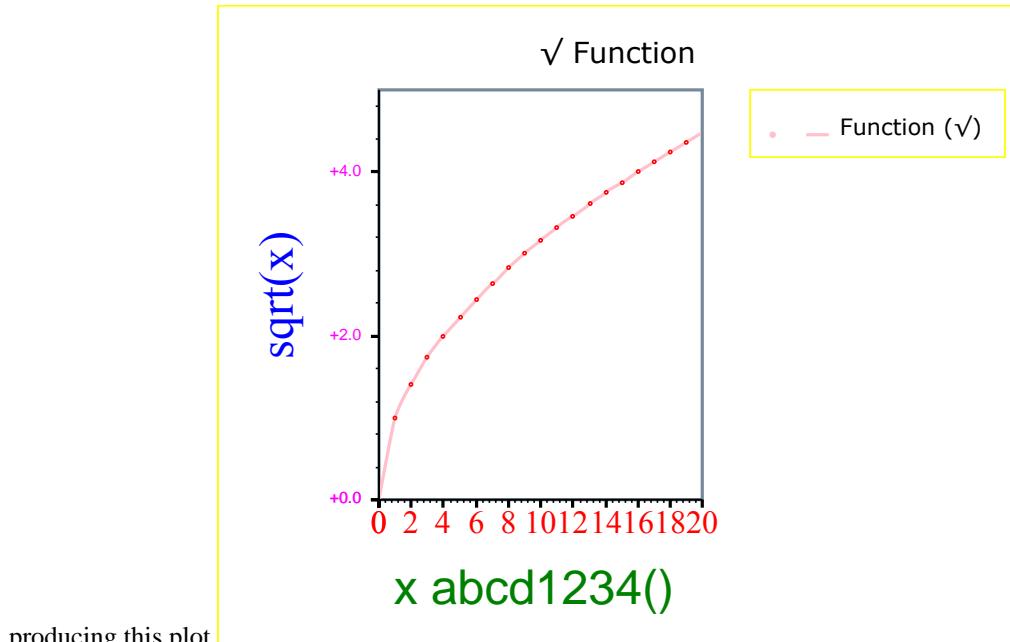
// Show the ticks styling:
// X ticks.
cout << "my_plot.x_ticks_values_color() " << my_plot.x_ticks_values_color() << endl;
cout << "my_plot.x_ticks_values_font_family() " << my_plot.x_ticks_values_font_family() << endl;
cout << "my_plot.x_ticks_values_font_size() " << my_plot.x_ticks_values_font_size() << endl;
cout << "my_plot.x_ticks_values_precision() " << my_plot.x_ticks_values_precision() << endl;
cout << "my_plot.x_ticks_values_ioflags() 0x" << hex << my_plot.x_ticks_values_ioflags() << dec << endl;
// Y ticks.
cout << "my_plot.y_ticks_values_color() " << my_plot.y_ticks_values_color() << endl;

```

```

cout << "my_plot.y_ticks_values_font_family() " << my_plot.y_ticks_values_font_family() << endl;
cout << "my_plot.y_ticks_values_font_size() " << my_plot.y_ticks_values_font_size() << endl;
cout << "my_plot.y_ticks_values_color() " << my_plot.y_ticks_values_color() << endl;
cout << "my_plot.y_ticks_values_precision() " << my_plot.y_ticks_values_precision() << endl;
cout << "my_plot.y_ticks_values_ioflags() 0x" << hex << my_plot.y_ticks_values_ioflags() << dec << endl;
}

```



See [demo_2d_tick_values.cpp](#) for full source code.

1-D Axis Scaling

Axis scaling with function scale_axis

This example shows the use of functions scale_axis to find suitable axis limits. Normally one would use autoscaling, but there are conceivable circumstances when one would want to check on the scale_axis algorithm's choice of axis, and perhaps intervene in the process.

First some includes to use Boost.Plot (and some others only needed for this example).

```

#include <boost/svg_plot/svg_1d_plot.hpp>
using namespace boost::svg;
using boost::svg::svg_1d_plot;

#include <boost/svg_plot/show_1d_settings.hpp>
// Only needed for showing which settings in use.
// void boost::svg::show_1d_plot_settings(svg_1d_plot&);
// (Also provides operator<< for std::pair).

#include <boost/algorithm/minmax.hpp>
using boost::minmax;
#include <boost/algorithm/minmax_element.hpp>
using boost::minmax_element;

#include <iostream> // for debugging.
using std::cout;
using std::endl;
using std::boolalpha;

#include <limits>
using std::numeric_limits;

#include <vector>
using std::vector;
#include <set>
using std::multiset;

#include <utility>
using std::pair;

#include <boost/svg_plot/detail/auto_axes.hpp>
using boost::svg::show; // A single STL container.
using boost::svg::show_all; // Multiple STL containers.
// using boost::svg::range; // Find min and max of a STL container.
using boost::svg::range_all; // Find min and max of multiple STL containers.

```

This example uses a few types of containers to demonstrate axis_scaling. axis_scaling must inspect the container in order to find axis ranges that will be suitable. First we create a container and fill with some fictional data.

```
vector<double> my_data;
// Initialize my_data with some entirely fictional data.
my_data.push_back(0.2);
my_data.push_back(1.1); // [1]
my_data.push_back(4.2); // [2]
my_data.push_back(3.3); // [3]
my_data.push_back(5.4); // [4]
my_data.push_back(6.5); // [5]
show(my_data); // Show entire container contents,
// 6 values in container: 0.2 1.1 4.2 3.3 5.4 6.5
```

```
multiset<double> my_set;
// Initialize my_set with some entirely fictional data.
my_set.insert(1.2);
my_set.insert(2.3);
my_set.insert(3.4);
my_set.insert(4.5);
my_set.insert(5.6);
my_set.insert(6.7);
my_set.insert(7.8);
my_set.insert(8.9);
// Show the set.
multiset<double>::const_iterator si;
show(my_set); // for two different types of container.
// 8 values in container: 1.2 2.3 3.4 4.5 5.6 6.7 7.8 8.9
```

Show can also display just a part of the container contents.

```
// show(&my_data[0], &my_data[my_data.size()]); // pointers - wrong! > all data ;-
show(&my_data[0], &my_data[my_data.size()-1]); // pointers, all data.
show(&my_data[1], &my_data[5]); // pointers, part data.
show(my_data.begin(), my_data.end()); // iterators.
show(++(my_data.begin()), --(my_data.end())); // Just the 4 middle values.

vector<double>::const_iterator idb = my_data.begin();
vector<double>::const_iterator ide = my_data.end();
show(idb, ide); // All
++idb; // Move to 2nd value.
--ide; // move back from last value.
show(idb, ide); // Just the 4 middle values.
```

scale_axis Function Examples

Is is possible to find the minimum and maximum values in a container using the STL functions min & max or more conveniently and efficiently with boost::minmax_element:

```
typedef vector<double>::const_iterator vector_iterator;
pair<vector_iterator, vector_iterator> result = boost::minmax_element(my_data.begin(), my_data.end());
cout << "The smallest element is " << *(result.first) << endl; // 0.2
cout << "The largest element is " << *(result.second) << endl; // 6.5

// axis_scaling using two double min and max values.
double min_value = *(my_data.begin());
double max_value = *(--my_data.end());
cout << "axis_scaling 1 min " << min_value << ", max = " << max_value << endl;
```

and to apply these values to the axis_scaling algorithm using by plot to choose the axes limits and ticks.

```

double axis_min_value; // Values to be updated by function `scale_axis`.
double axis_max_value;
double axis_tick_increment;
int axis_ticks;

scale_axis(min_value, max_value,
    &axis_min_value, &axis_max_value, &axis_tick_increment, &axis_ticks,
    false, tol100eps, 6); // Display range.
cout << "Axis_scaled 2 min " << axis_min_value << ", max = " << axis_max_value << ", increment " << axis_ticks << endl;

```

It is also possible to use this with containers that use iterators and whose contents are ordered in ascending value, axis_scaling using first and last in container, for example, set, map, multimap, or a sorted vector or array. A number of variations are shown below, mainly by way of testing.

```

scale_axis(*my_data.begin(),*(--my_data.end())),
    &axis_min_value, &axis_max_value, &axis_tick_increment, &axis_ticks,
    false, tol100eps, 6); // Display range.
cout << "Axis_scaled 3 min " << axis_min_value << ", max = " << axis_max_value << ", increment " << axis_ticks << endl;

// axis_scaling using two begin & end iterators into STL container,
// scale_axis does finding min and max.
scale_axis(my_data.begin(), my_data.end(), // Input range
    &axis_min_value, &axis_max_value, &axis_tick_increment, &axis_ticks, // to update.
    true, // check for non-finite
    3., // autoscale_plusminus = 3 sd
    false, // Do not include origin
    tol100eps, // tight
    6, // steps at default.
    0); // Display range.
cout << "Axis_scaled 4 min " << axis_min_value << ", max = " << axis_max_value << ", increment " << axis_ticks << endl;

// axis_scaling using two begin & end iterators into STL container,
// scale_axis does finding min and max.
scale_axis(my_data[1], my_data[4], // Only middle part of the container used, ignoring 1st and last values.
    &axis_min_value, &axis_max_value, &axis_tick_increment, &axis_ticks,
    true, tol100eps, 6); // Display range.
cout << "Axis_scaled 5 min " << axis_min_value << ", max = " << axis_max_value << ", increment " << axis_ticks << endl;

// axis_scaling using whole STL container,
// scale_axis does finding min and max.
scale_axis(my_data, &axis_min_value, &axis_max_value, &axis_tick_increment, &axis_ticks,
    true, 3., false, tol100eps, 6); // Display range.
cout << "Axis_scaled 6 min " << axis_min_value << ", max = " << axis_max_value << ", increment " << axis_ticks << endl;

svg_1d_plot my_1d_plot; // Construct a plot with all the default constructor values.

// One could intercept and change any values calculated by scale_axis here?
// Set the plot to use range and interval from the scale_axis values.

```

The axis range thus computed can be inserted directly into the plot using the range and x_major_interval functions.

```

my_1d_plot.x_range(axis_min_value, axis_max_value)
    .x_major_interval(axis_tick_increment);

//my_1d_plot.x_autoscale(false); // Ensure autoscale values are *not* recalculated for the plot.

// Set some axis_scaling parameters:
my_1d_plot.x_with_zero(false);
my_1d_plot.x_min_ticks(10);
my_1d_plot.x_steps(0);
my_1d_plot.x_tight(0.001);

// Show the flags just set.
cout << (my_1d_plot.x_with_zero() ? "x_with_zero, " : "not x_with_zero, ")
    << my_1d_plot.x_min_ticks() << "x_min_ticks, "
    << my_1d_plot.x_steps() << "x_steps, "
    << my_1d_plot.x_tight() << "tightness." << endl;

my_1d_plot.x_autoscale(my_data); // Use all my_data to autoscale.
cout << "Axis_scaled " // Show the results of autoscale:
    "min " << my_1d_plot.x_auto_min_value()
    << ", max " << my_1d_plot.x_auto_max_value()
    << ", interval " << my_1d_plot.x_auto_tick_interval() << endl; // Autoscaled min 0, max 6.5, interval 0.5

my_1d_plot.x_autoscale(my_data.begin(), my_data.end()); // Use all my_data to autoscale.

cout << "Axis_scaled " // Show the results of autoscale:
    "min " << my_1d_plot.x_auto_min_value()
    << ", max " << my_1d_plot.x_auto_max_value()
    << ", interval " << my_1d_plot.x_auto_tick_interval() << endl; // Autoscaled min 0, max 6.5, interval 0.5

my_1d_plot.x_autoscale(my_data[1], my_data[4]); // Use only part of my_data to autoscale.

cout << "Axis_scaled " // Show the results of autoscale:
    "min " << my_1d_plot.x_auto_min_value()
    << ", max " << my_1d_plot.x_auto_max_value()
    << ", interval " << my_1d_plot.x_auto_tick_interval() << endl; // Autoscaled min 1, max 5.5, interval 0.5

my_1d_plot.x_autoscale(true); // Ensure autoscale values are used for the plot.

my_1d_plot.plot(my_data, "Auto 1D"); // Add the one data series.
cout << "Axis_scaled " // Show the results of autoscale:
    "min " << my_1d_plot.x_auto_min_value()
    << ", max " << my_1d_plot.x_auto_max_value()
    << ", interval " << my_1d_plot.x_auto_tick_interval() << endl; // Autoscaled min 1, max 5.5, interval 0.5

my_1d_plot.plot(my_set.begin(), my_set.end(), "Auto 1D"); // Add another data series from my_set.
my_1d_plot.plot(my_set, "Auto 1D"); // Add another whole data series from my_set.
my_1d_plot.plot(&my_data[1], &my_data[4], "Auto 1D"); // Add part (1,2 3 but *not* 4) of the one data series.
//my_1d_plot.plot(&my_set[1], &my_set[4], "Auto 1D"); // operator[] is not defined for set container!

my_1d_plot.write("demo_1d_axis_scaling.svg"); // Write the plot to file.

using boost::svg::detail::operator<<; // Needed for output a pair.
cout << "x_range() " << my_1d_plot.x_range() << endl; // x_range() 1, 5.5

//show_1d_plot_settings(my_1d_plot); // For *all* settings.

```

The output is:

```

Autorun "j:\Cpp\SVG\Debug\demo_1d_axis_scaling.exe"
6 values in container: 0.2 1.1 4.2 3.3 5.4 6.5
8 values in container: 1.2 2.3 3.4 4.5 5.6 6.7 7.8 8.9
0.2 1.1 4.2 3.3 5.4 : 5 values used.
1.1 4.2 3.3 5.4 : 4 values used.
0.2 1.1 4.2 3.3 5.4 6.5 : 6 values used.
1.1 4.2 3.3 5.4 : 4 values used.
0.2 1.1 4.2 3.3 5.4 6.5 : 6 values used.
1.1 4.2 3.3 5.4 : 4 values used.
The smallest element is 0.2
The largest element is 6.5
axis_scaling 1 min 0.2, max = 6.5
Axis_scaled 2 min 0, max = 7, increment 1
Axis_scaled 3 min 0, max = 7, increment 1
Axis_scaled 4 min 0, max = 7, increment 1
Axis_scaled 5 min 0, max = 6, increment 1
Axis_scaled 6 min 0, max = 7, increment 1
not x_with_zero, 10 x_min_ticks, 0 x_steps, 0.001 tightness.
Axis_scaled min 0, max 6.5, interval 0.5
Axis_scaled min 0, max 6.5, interval 0.5
Axis_scaled min 1, max 5.5, interval 0.5
Axis_scaled min 1, max 5.5, interval 0.5
x_range() 1, 5.5

```

See [demo_1d_axis_scaling.cpp](#) for full source code.

1-D Auto scaling Examples

First we need a few includes to use Boost.Plot (and some others only needed for this example).

```
#include <boost/svg_plot/svg_1d_plot.hpp>
using namespace boost::svg;
using boost::svg::svg_1d_plot;

#include <boost/svg_plot/show_1d_settings.hpp>
// Only needed for showing which settings in use.
// void boost::svg::show_1d_plot_settings(svg_1d_plot&);
// (Also provides operator<< for std::pair).

#include <boost/algorithms/minmax.hpp>
using boost::minmax;
#include <boost/algorithms/minmax_element.hpp>
using boost::minmax_element;

#include <iostream> // for debugging.
using std::cout;
using std::endl;
using std::boolalpha;

#include <limits>
using std::numeric_limits;

#include <vector>
using std::vector;
#include <set>
using std::multiset;

#include <utility>
using std::pair;

#include <boost/svg_plot/detail/auto_axes.hpp>
using boost::svg::show; // A single STL container.
using boost::svg::show_all; // Multiple STL containers.
// using boost::svg::range; // Find min and max of a STL container.
using boost::svg::range_all; // Find min and max of multiple STL containers.
```

This example uses containers to demonstrate autoscaling. Autoscaling must inspect the container in order to find axis ranges that will be suitable. First we create a container and fill with some fictional data.

```
vector<double> my_data;
// Initialize my_data with some entirely fictional data.
my_data.push_back(0.2);
my_data.push_back(1.1); // [1]
my_data.push_back(4.2); // [2]
my_data.push_back(3.3); // [3]
my_data.push_back(5.4); // [4]
my_data.push_back(6.5); // [5]
```

Also included is an 'at limit' value that could confuse autoscaling. Obviously, we do **not** want the plot range to include infinity.

```

my_data.push_back(numeric_limits<double>::infinity()); // [6]

try
{ // Ensure error, warning and information messages are displayed by the catch block.
    double mn;
    double mx;
    int good = mnmx(my_data.begin(), my_data.end(), &mn, &mx);
    cout << good << " good values, " << my_data.size() - good << " limit values." << endl;

    using boost::svg::detail::operator<<; // For displaying std::pair.
    svg_1d_plot my_1d_plot; // Construct a plot with all the default constructor values.
    my_1d_plot.x_autoscale(my_data); // Compute autoscale values for the plot.
    my_1d_plot.limit_color(blue).limit_fill_color(green); // Add limit value styling.

    cout << "my_1d_plot.limit_color() " << my_1d_plot.limit_color() << endl;
    cout << "my_1d_plot.limit_fill_color() " << my_1d_plot.limit_fill_color() << endl;

    my_1d_plot.plot(my_data, "Default 1D"); // Add the one data series, and give it a title.
    my_1d_plot.write("auto_1d_plot_1.svg"); // Write the plot to file.

```

It may be useful to display that range chosen by autoscaling.

```

cout << "x_range() " << my_1d_plot.x_range() << endl; // x_range()

```

Other STL containers can also be used, for example set, or multiset (allowing duplicates):

```

try
{ // Ensure error, warning and information messages are displayed by the catch block.

multiset<double> my_set;
// Initialize my_set with some entirely fictional data.
my_set.insert(1.2);
my_set.insert(2.3);
my_set.insert(3.4);
my_set.insert(4.5);
my_set.insert(5.6);
my_set.insert(6.7);
my_set.insert(7.8);
my_set.insert(8.9);
// Some examples of showing the set using various iterators:
multiset<double>::const_iterator si;

show(my_data); // Entire container contents,
show(my_set); // for two different types of container.

// show(&my_data[0], &my_data[my_data.size()]); // pointers - wrong! uses more all data ;-
show(&my_data[0], &my_data[my_data.size()-1]); // pointers, all data.
show(&my_data[1], &my_data[5]); // pointers, part data.
show(my_data.begin(), my_data.end()); // iterators.
show(++(my_data.begin()), --(my_data.end())); // Just the 4 middle values.

vector<double>::const_iterator idb = my_data.begin();
vector<double>::const_iterator ide = my_data.end();
show(idb, ide); // All
++idb; // Move to 2nd value.
--ide; // move back from last value.
show(idb, ide); // Just the 4 middle values.

typedef vector<double>::const_iterator vector_iterator;
pair<vector_iterator, vector_iterator> result = boost::minmax_element(my_data.begin(), my_data.end());
cout << "The smallest element is " << *(result.first) << endl; // 0.2
cout << "The largest element is " << *(result.second) << endl; // 6.5

```

Autoscaling can also use two double min and max values provided by the user program. Using `x_autoscale(my_set)` effectively uses the first and last items in the STL container. If the container is sorted, then these are the minimum and maximum values.

```

double min_value = *(my_data.begin());
double max_value = *(--my_data.end());
cout << "my_set min " << min_value << ", max = " << max_value << endl;

```

Function `scale_axis` is used by `autoscale`, but is also available for use direct by the user. It accepts parameters controlling the scaling and updates 4 items. Its signature is:

```
void scale_axis(
    double min_value, // Input range min
    double max_value, // Input range max
    double* axis_min_value, double* axis_max_value, double* axis_tick_increment, int* auto_ticks, // All 4 updated.
    bool check_limits, // Whether to check all values for infinity, NaN etc.
    bool origin, // If true, ensures that zero is a tick value.
    double tight, // Allows user to avoid a small fraction over a tick using another tick.
    int min_ticks, // Minimum number of ticks.
```

```
double axis_min_value; // Values to be updated by autoscale.
double axis_max_value;
double axis_tick_increment;
int axis_ticks;
```

The values of `min_value` and `max_value` could be provided by the user program: but usually these values are derived in some way from the user data. Several examples follow:

```
scale_axis(1., 9., // User chosen min and max,
           &axis_min_value, &axis_max_value, &axis_tick_increment, &axis_ticks,
           false, tol100eps, 6, 0); // Display range.
cout << "scaled min " << axis_min_value << ", max = " << axis_max_value
     << ", increment " << axis_tick_increment << ", axis ticks " << axis_ticks << endl;

// Scaling using only the first and last values in a container,
// assuming the data are already ordered in ascending value,
// for example, set, map, multimap, or a sorted vector or array.
// scale_axis(*my_data.begin(),*(--my_data.end()));

// Scaling using two begin & end iterators into STL container,
// scale_axis does finding min and max.
scale_axis(my_data.begin(), my_data.end(),
           &axis_min_value, &axis_max_value, &axis_tick_increment, &axis_ticks,
           true, 3., false, tol100eps, 6); // Display range.
cout << "scaled min " << axis_min_value << ", max = " << axis_max_value
     << ", increment " << axis_tick_increment << ", axis ticks " << axis_ticks << endl;

// Scaling using two begin & end iterators into STL container,
// scale_axis does finding min and max.
scale_axis(my_data[1], my_data[4], // Only middle part of the container used, ignoring 1st and last values.
           &axis_min_value, &axis_max_value, &axis_tick_increment, &axis_ticks,
           false, tol100eps, 6); // Display range.
cout << "scaled min " << axis_min_value << ", max = " << axis_max_value
     << ", increment " << axis_tick_increment << ", axis ticks " << axis_ticks << endl;

// Scaling using whole STL vector container,
// scale_axis does finding min and max.
scale_axis(my_data, &axis_min_value, &axis_max_value, &axis_tick_increment, &axis_ticks,
           true, 3., false, tol100eps, 6); // Display range.
cout << "scaled min " << axis_min_value << ", max = " << axis_max_value
     << ", increment " << axis_tick_increment << ", axis ticks " << axis_ticks << endl;

// Scaling using whole STL set container,
// scale_axis does finding min and max.
scale_axis(my_set, &axis_min_value, &axis_max_value, &axis_tick_increment, &axis_ticks,
           true, 3., false, tol100eps, 6); // Display range.
cout << "scaled min " << axis_min_value << ", max = " << axis_max_value
     << ", increment " << axis_tick_increment << ", axis ticks " << axis_ticks << endl;
```

However autoscaling may go wrong if the data could contain values that are outside normal limits. Infinity (+ and -), and maximum value, and NaN (Not A Number), are separated by the plot program to allow them to be shown but separate from 'normal' values. These values similarly can distort automatic scaling: a single infinity would result in useless scaling! When the plot range is set, the

maximum and minimum values are checked, and if not normal then an exception will be thrown, and no plot will be produced. However, when autoscaling, it is more useful to ignore 'limit' values. But this means checking all values individually. If it known that all values are normal, for example because they come from some measuring equipment that is known only to produce normal values, it will be much quicker to use `std::min_max_element` which can take advantage of knowledge of the container.

The function `autoscale_check_limits(bool)` is provided to control this. If set true, all values will be checked, and those 'at limits' will be ignored in autoscaling. The default is true, to check all values.

If we had many very known normal values to plot and want to autoscale, we might instead opt to ignore these checks, and write:

```
svg_1d_plot my_1d_plot; // Construct a plot with all the default constructor values.

//my_1d_plot.autoscale_check_limits(false);
// This *will throw an exception* if checks are avoided and any values are at 'limits'.

// One could also intercept and change any values calculated by scale_axis here.
// Set the plot to use range and interval from the scale_axis values.
my_1d_plot.x_range(axis_min_value, axis_max_value)
    .x_major_interval(axis_tick_increment);

my_1d_plot.x_autoscale(false); // Ensure autoscale values are *not* recalculated for the plot.
```

There are also some parameters which can fine-tune the autoscaling to produce more aesthetically pleasing ranges. One can:

1. Enforce the inclusion of zero on the plot.
2. Specify a minimum number of major ticks.
3. Specify the steps between major ticks, default 0, or 2 for 2, 4, 6, 8, 10, 5 for 1, 5, 10, or 10 (2, 5, 10).
4. Avoid values that are a tiny amount over the minimum or maximum from causing an apparently unnecessary tick at the minimum or maximum.

```
// Set some autoscaling parameters:
my_1d_plot.x_with_zero(false);
my_1d_plot.x_min_ticks(10);
my_1d_plot.x_steps(0);
my_1d_plot.x_tight(0.001);

// Show the flags just set.
cout << (my_1d_plot.x_with_zero() ? "x_with_zero, " : "not x_with_zero, ")
    << my_1d_plot.x_min_ticks() << " x_min_ticks, "
    << my_1d_plot.x_steps() << " x_steps, "
    << my_1d_plot.x_tight() << " tightness." << endl;
```

Finally here are some examples of using autoscaling using all or part of containers.

```

my_1d_plot.x_autoscale(my_data); // Use all my_data to autoscale.
cout << "Autoscaled" // Show the results of autoscale:
"min" << my_1d_plot.x_auto_min_value()
<< ", max" << my_1d_plot.x_auto_max_value()
<< ", interval" << my_1d_plot.x_auto_tick_interval() << endl; // Autoscaled min 0, max 6.5, interval 0.5

my_1d_plot.x_autoscale(my_data.begin(), my_data.end()); // Use all my_data to autoscale.

cout << "Autoscaled" // Show the results of autoscale:
"min" << my_1d_plot.x_auto_min_value()
<< ", max" << my_1d_plot.x_auto_max_value()
<< ", interval" << my_1d_plot.x_auto_tick_interval() << endl; // Autoscaled min 0, max 6.5, interval 0.5

my_1d_plot.x_autoscale(my_data[1], my_data[4]); // Use only part of my_data to autoscale.

cout << "Autoscaled" // Show the results of autoscale:
"min" << my_1d_plot.x_auto_min_value()
<< ", max" << my_1d_plot.x_auto_max_value()
<< ", interval" << my_1d_plot.x_auto_tick_interval() << endl; // Autoscaled min 1, max 5.5, interval 0.5

//my_1d_plot.x_autoscale(true); // Ensure autoscale values are used for the plot.
// This is also automatically set true by any call of x_autoscale(some_data);

```

The actual addition of data values to the plot is, of course, quite separate from any autoscaling.

```

my_1d_plot.plot(my_data, "Auto 1D"); // Add the one data series.
cout << "Autoscaled" // Show the results of autoscale:
" min" << my_1d_plot.x_auto_min_value()
<< ", max" << my_1d_plot.x_auto_max_value()
<< ", interval" << my_1d_plot.x_auto_tick_interval() << endl; // Autoscaled min 1, max 5.5, interval 0.5

my_1d_plot.plot(my_set.begin(), my_set.end(), "Auto 1D"); // Add another data series from my_set.
my_1d_plot.plot(my_set, "Auto 1D"); // Add another whole data series from my_set.
my_1d_plot.plot(&my_data[1], &my_data[4], "Auto 1D"); // Add part (1,2 3 but *not* 4) of the one data series.
//my_1d_plot.plot(&my_set[1], &my_set[4], "Auto 1D"); // operator[] is not defined for set container!

my_1d_plot.write("auto_1d_plot_2.svg"); // Write the plot to file.

using boost::svg::detail::operator<<;
cout << "x_range()" << my_1d_plot.x_range() << endl; // x_range() 1, 5.5
show_1d_plot_settings(my_1d_plot);
}
catch(const std::exception& e)
{ // Error, warning and information messages are displayed by the catch block.
std::cout <<
"\n""Message from thrown exception was:\n" << e.what() << std::endl;
}

```

If necessary, one can obtain a complete listing of all the settings used. This is often useful when the plot does not meet expectations.

The output is:

```

----- Rebuild All started: Project: auto_1d_plot, Configuration: Debug Win32 -----
Deleting intermediate and output files for project 'auto_1d_plot', configuration 'Debug|Win32'
Compiling...
auto_1d_plot.cpp
Linking...
Embedding manifest...
Autorun "j:\Cpp\SVG\debug\auto_1d_plot.exe"
limit value: 1.#INF
6 good values, 1 limit values.
limit value: 1.#INF
x_range() 0, 7
7 values in container: 0.2 1.1 4.2 3.3 5.4 6.5 1.#INF
8 values in container: 1.2 2.3 3.4 4.5 5.6 6.7 7.8 8.9
0.2 1.1 4.2 3.3 5.4 6.5 : 6 values used.
1.1 4.2 3.3 5.4 : 4 values used.
0.2 1.1 4.2 3.3 5.4 6.5 1.#INF : 7 values used.
1.1 4.2 3.3 5.4 6.5 : 5 values used.
0.2 1.1 4.2 3.3 5.4 6.5 1.#INF : 7 values used.
1.1 4.2 3.3 5.4 6.5 : 5 values used.
The smallest element is 0.2
The largest element is 1.#INF
my_set min 0.2, max = 1.#INF
scaled min 1, max = 9, increment 1, axis ticks 9
limit value: 1.#INF
scaled min 0, max = 7, increment 1, axis ticks 8
scaled min 1, max = 6, increment 1, axis ticks 6
limit value: 1.#INF
scaled min 0, max = 7, increment 1, axis ticks 8
scaled min 1, max = 9, increment 1, axis ticks 9
not x_with_zero, 10 x_min_ticks, 0 x_steps, 0.001 tightness.
limit value: 1.#INF
Autoscaled min 0, max 6.5, interval 0.5
limit value: 1.#INF
Autoscaled min 0, max 6.5, interval 0.5
Autoscaled min 1, max 5.5, interval 0.5
Autoscaled min 1, max 5.5, interval 0.5
x_range() 1, 5.5

axes_on false
background_border_width 1
background_border_color RGB(255,255,0)
background_color RGB(255,255,255)
image_border_margin() 10
image_border_width() 1
coord_precision 3
copyright_date
copyright_holder
description
document_title
image_x_size 500
image_y_size 200
legend_on false
legend_place 2
legend_top_left -1, -1, legend_bottom_right -1, -1
legend_background_color blank
legend_border_color RGB(255,255,0)
legend_color blank
legend_title
legend_title_font_size 14
legend_width 0
legend_lines true
limit points stroke color RGB(119,136,153)
limit points fill color RGB(250,235,215)

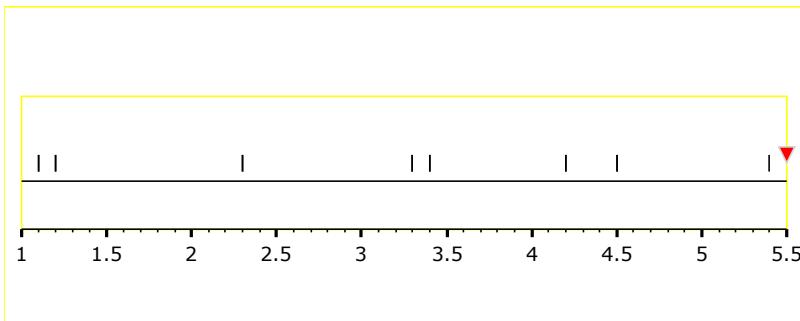
```

```

license_on false
license_reproduction permits
license_distribution permits
license_attribution requires
license_commercialuse permits
plot_background_color RGB(255,255,255)
plot_border_color RGB(255,255,0)
plot_border_width 1
plot_window_on true
plot_window_x 11, 489
plot_window_x_left 11
plot_window_x_right 489
plot_window_y 56, 139
plot_window_y_top 56
plot_window_y_bottom 139.2
title_on true
title ""
title_color blank
title_font_alignment 2
title_font_decoration
title_font_family Verdana
title_font_rotation 0
title_font_size 18
title_font_stretch
title_font_style
x_value_precision 3
x_value_ioflags 200 IOS format flags (0x200) dec.
x_max 5.5
x_min 1
x_axis_on true
x_axis_color() RGB(0,0,0)
x_axis_label_color RGB(0,0,0)
x_axis_value_color RGB(0,0,0)
x_axis_width 1
x_label_on true
x_label
x_label_color blank
x_label_font_family Verdana
x_label_font_size 14
x_label_units
x_label_units_on false
x_major_labels_side left
x_values_font_size 10
x_values_color blank
x_values_precision 3
x_values_ioflags 512
x_major_label_rotation 0
x_major_grid_color RGB(200,220,255)
x_major_grid_on false
x_major_grid_width 1
x_major_interval 0.5
x_major_tick 0.5
x_major_tick_color RGB(0,0,0)
x_major_tick_length 5
x_major_tick_width 2
x_minor_interval 0
x_minor_tick_color RGB(0,0,0)
x_minor_tick_length 2
x_minor_tick_width 1
x_minor_grid_on false
x_minor_grid_color RGB(200,220,255)
x_minor_grid_width 0.5
x_range() 1, 5.5

```

```
x_num_minor_ticks 4
x_ticks_down_on true
x_ticks_up_on false
x_ticks_on_window_or_axis bottom
x_axis_position x_axis_position intersects Y axis (Y range includes zero)
x_autoscale true
x_autoscale_check_limits true
data_lines width 0
```



See [auto_1d_plot.cpp](#) for full source code.

1-D Autoscaling Various Containers Examples

First some includes to use Boost.Plot (and some others only needed for this example).

```
#include <boost/svg_plot/svg_1d_plot.hpp>
using namespace boost::svg;
using boost::svg::svg_1d_plot;

#include <boost/svg_plot/show_1d_settings.hpp>
// Only needed for showing which settings in use.
// void boost::svg::show_1d_plot_settings(svg_1d_plot&);
// (Also provides operator<< for std::pair).

#include <boost/algorithm/minmax.hpp>
using boost::minmax;
#include <boost/algorithm/minmax_element.hpp>
using boost::minmax_element;

#include <iostream> // for debugging.
using std::cout;
using std::endl;
using std::boolalpha;

#include <limits>
using std::numeric_limits;

#include <vector>
using std::vector;
#include <set>
using std::multiset;

#include <utility>
using std::pair;

#include <boost/svg_plot/detail/auto_axes.hpp>
using boost::svg::show; //!< A single STL container.
using boost::svg::show_all; //!< Multiple STL containers.
using boost::svg::range_mx; //!< Find min and max of a single STL container.
using boost::svg::range_all; //!< Find min and max of multiple STL containers.
```

This example uses a few types of containers to demonstrate autoscaling. Autoscaling can inspect the container in order to find axis ranges that will be suitable. First we create a container and fill with some fictional data, and display the values.

```
std::vector<double> my_data;
// Initialize my_data with some entirely fictional data.
my_data.push_back(0.2);
my_data.push_back(1.1); // [1]
my_data.push_back(4.2); // [2]
my_data.push_back(3.3); // [3]
my_data.push_back(5.4); // [4]
my_data.push_back(6.5); // [5]
show(my_data); // Show entire container contents,
// 6 values in container: 0.2 1.1 4.2 3.3 5.4 6.5
```

Construct a plot , and add some data to the plot.

```
svg_1d_plot my_1d_plot; // Construct a plot with all the default constructor values.
my_1d_plot.title("Demo 1D autoscaling").x_label("X values"); // Add a title and X axis label.
my_1d_plot.plot(my_data, "Auto 1D my_data"); // Add whole data series from my_data.
```

Use `x_autoscale` to scale the axis, in this most common and simplest case, using all the values.

```
my_1d_plot.x_autoscale(my_data);
```

and finally write the SVG to a file.

```
my_1d_plot.write("demo_1d_autoscaling_1.svg"); // Write the plot to file.
```

In a second example, we use a different type of container, a set, and use autoscale in a more advanced way.

```
std::multiset<double> my_set;
// Initialize my_set with some entirely fictional data.
my_set.insert(1.2);
my_set.insert(2.3);
my_set.insert(3.4);
my_set.insert(4.5);
my_set.insert(5.6);
my_set.insert(6.7);
my_set.insert(7.8);
my_set.insert(8.9);
// Show the set.
multiset<double>::const_iterator si;
show(my_set); // for two different types of container.
// 8 values in container: 1.2 2.3 3.4 4.5 5.6 6.7 7.8 8.9
svg_1d_plot my_1d_plot; // Construct a plot with all the default constructor values.
```

and also override the default controls of scale_axis function used by autoscaling.

```
// Set some autoscaling parameters:
my_1d_plot.x_with_zero(true); // Always include zero in the axis range, even if the data values don't.
my_1d_plot.x_min_ticks(10); // Ensure more than the default minimum number of ticks.
my_1d_plot.x_steps(0); // Permits axis ticks in steps of 1,2,4,6,8 or 1, 5, 10, or 1, 2, 5, 10.
my_1d_plot.x_tight(0.01); // Prevent values that are only 1% outside the last tick requiring another tick.

// Show the flags just set.
cout << (my_1d_plot.x_with_zero() ? "x_with_zero, " : "not x_with_zero, ") // x_with_zero
    << my_1d_plot.x_min_ticks() << "x_min_ticks, " // 10
    << my_1d_plot.x_steps() << "x_steps, " // 0
    << my_1d_plot.x_tight() << "tightness." << endl; // 0.001
```

Purely to show the possibilities with autoscale, we don't use the whole container, but use the two-iterators version of autoscale to **not use the first nor the last values** for autoscaling. (Remember values in set are sorted).

```
my_1d_plot.x_autoscale(++my_set.begin(),--my_set.end());
```

This also sets autoscale(true), but note that x_range() is still not updated. If we want, we can display the ranges chosen by autoscale:

```
cout << "x_auto_min_value" << my_1d_plot.x_auto_min_value()
    << ", x_auto_max_value" << my_1d_plot.x_auto_max_value()
    << ", x_auto_tick_interval" << my_1d_plot.x_auto_tick_interval() << endl;
```

As before, we add the data set to the plot, and write to SVG XML to file. If you inspect the plot, you will see that the lowest data value 1.2 and highest data value 8.9 are no longer shown, because it is now outside the plot window. This is as if we had only written part of the data series.

```

my_1d_plot.plot(++my_set.begin(),--my_set.end(), "Auto 1D my_set"); // Add 'top and tailed' data series from my_set.
//my_1d_plot.plot(my_set, "Auto 1D my_set"); // Add whole data series from my_set.
my_1d_plot.write("demo_1d_autoscaling_2.svg"); // Write the plot to file.

```

If we want, we can check the autoscale range used, noting that zero **is** included because we demanded it.

```

using boost::svg::detail::operator<<;
cout << "x_range() " << my_1d_plot.x_range() << endl; // x_range() 0, 8
//show_1d_plot_settings(my_1d_plot); // If required.
}

```

try'n'catch blocks are very useful to display any plot error messages. Otherwise any exceptions thrown will just terminate - silently and most unhelpfully.

```

catch(const std::exception& e)
{
    cout << "\n""Message from thrown exception was:\n " << e.what() << endl;
}

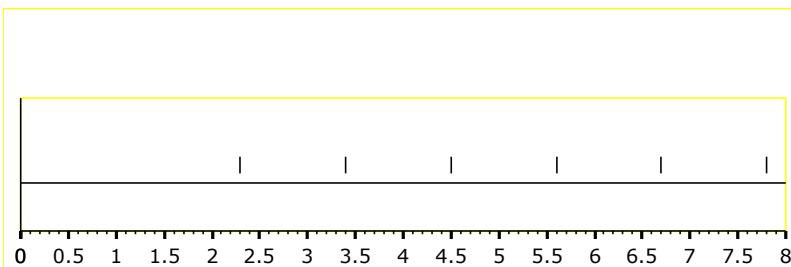
```

The output is:

```

demo_1d_autoscaling.cpp
Linking...
Embedding manifest...
Autorun "j:\Cpp\SVG\debug\demo_1d_autoscaling.exe"
6 values in container: 0.2 1.1 4.2 3.3 5.4 6.5
8 values in container: 1.2 2.3 3.4 4.5 5.6 6.7 7.8 8.9
x_with_zero, 10 x_min_ticks, 0 x_steps, 0.01 tightness.
x_auto_min_value 0, x_auto_max_value 8, x_auto_tick_interval 0.5
x_range() 0, 8
Build Time 0:04

```



and the plot:

See [demo_1d_autoscaling.cpp](#) for full source code.

1-D Data Values Examples

Showing 1D Data Values Examples

Showing the 1D values of items from the data set. Some of the many possible formatting options are demonstrated.

As ever, we need a few includes to use Boost.Plot

```
#include <boost/svg_plot/svg_1d_plot.hpp>
using namespace boost::svg;
using boost::svg::svg_1d_plot;

#include <boost/svg_plot/show_1d_settings.hpp>
// using boost::svg::show_1d_plot_settings - Only needed for showing which settings in use.

#include <iostream>
using std::cout;
using std::endl;
using std::hex;

#include <vector>
using std::vector;
```

Some fictional data is pushed into an STL container, here `vector<double>`:

```
vector<double> my_data;
my_data.push_back(-1.6);
my_data.push_back(2.0);
my_data.push_back(4.2563);
my_data.push_back(0.00333974);
my_data.push_back(5.4);
my_data.push_back(6.556);

try
{ // try'n'catch blocks are needed to ensure error messages from any exceptions are shown.
    svg_1d_plot my_1d_plot; // Construct a plot with all the default constructor values.

    my_1d_plot.title("Default 1D Values Demo") // Add a string title of the plot.
        .x_range(-5, 10) // Add a range for the X-axis.
        .x_label("length (m)"); // Add a label for the X-axis.
```

Add the one data series, `my_data` and a description, and how the data points are to marked, here a circle with a diameter of 5 pixels.

```
my_1d_plot.plot(my_data, "1D Values").shape(circlet).size(5);
```

To put a value label against each data point, switch on the option:

```
my_1d_plot.x_values_on(true); // Add a label for the X-axis.
```

If the default size and color are not to your taste, set more options, like:

```
my_1d_plot.size(500, 350) // Change from default size
    .x_values_font_size(14) // Change font size for the X-axis value labels.
    .x_values_font_family("Times New Roman") // Change font for the X-axis value labels.
    .x_values_color(red); // Change color from default black to red.
```

The format of the values may also not be ideal, so we can use the normal `iostream` precision and `ioflags` to change, here to reduce the number of digits used from default precision 6 down to a more readable 2, reducing the risk of collisions between adjacent values. (Obviously the most suitable precision depends on the range of the data points. If values are very close to each other, a higher precision will be needed to differentiate them).

```
my_1d_plot.x_values_precision(2); // precision label for the X-axis value label.
```

We can also prescribe the use of scientific format and force a positive sign:

```
my_1d_plot.x_values_ioflags(std::ios::scientific | std::ios::showpos);
```

By default, any unnecessary spacing-wasting zeros in the exponent field are removed. (If, perversely, the full 1.123456e+012 format is required, the stripping can be switched off with: `my_1d_plot.x_labels_strip_e0s(false);`)

In general, sticking to the defaults usually produces the neatest presentation of the values.

The default value label is horizontal, centered above the data point marker, but, depending on the type and density of data points, and the length of the values (controlled in turn by the precision and ioflags in use), it is often clearer to use a different orientation. This can be controlled in steps of 45 degrees, using an 'enum rotate_style`.

- **uphill** - writing up at 45 degree slope is often a good choice,
- **upward** - writing vertically up and
- **backup** are also useful.

(For 1-D plots other directions are less attractive, placing the values below the horizontal Y-axis line, but for 2-D plots all writing orientations can be useful).

```
my_1d_plot.x_values_rotation(steeplup); // Orientation for the X-axis value labels.
```

```
my_1d_plot.x_decor("[ x = ", "", "\u00A0;sec]"); // Note the need for a Unicode space A0.
```

To use all these settings, finally write the plot to file.

```
my_1d_plot.write("demo_1d_values.svg");
```

If chosen settings do not have the effect that you expect, it may be helpful to display some of them! (All the myriad settings can be displayed with `show_1d_plot_settings(my_1d_plot)`.)

```
//show_1d_plot_settings(my_1d_plot);
using boost::svg::detail::operator<<;
cout << "my_1d_plot.image_size()" << my_1d_plot.size() << endl;
cout << "my_1d_plot.image_x_size()" << my_1d_plot.x_size() << endl;
cout << "my_1d_plot.image_y_size()" << my_1d_plot.y_size() << endl;
cout << "my_1d_plot.x_values_font_size()" << my_1d_plot.x_values_font_size() << endl;
cout << "my_1d_plot.x_values_font_family()" << my_1d_plot.x_values_font_family() << endl;
cout << "my_1d_plot.x_values_color()" << my_1d_plot.x_values_color() << endl;
cout << "my_1d_plot.x_values_precision()" << my_1d_plot.x_values_precision() << endl;
cout << "my_1d_plot.x_values_ioflags()" << hex << my_1d_plot.x_values_ioflags() << endl;
```

Output:

```
demo_1d_values.cpp
Linking...
Embedding manifest...
Autorun "j:\Cpp\SVG\debug\demo_1d_values.exe"
my_1d_plot.x_values_font_size() 14
my_1d_plot.x_values_font_family() Times New Roman
my_1d_plot.x_values_color() RGB(255,0,0)
my_1d_plot.x_values_precision() 2
my_1d_plot.x_values_ioflags() 1020
Build Time 0:03
```

Showing 1D Data 'at limit' Values Examples

This example demonstrates showing values that are too small or too large, or NotANumber.

As ever, we need a few includes to use Boost.Plot

```
#include <boost/svg_plot/svg_1d_plot.hpp>
using namespace boost::svg;
using boost::svg::svg_1d_plot;

#include <iostream>
using std::cout;
using std::endl;
using std::hex;

#include <vector>
using std::vector;

#include <limits>
using std::numeric_limits;
```

Some fictional data is pushed into an STL container, here `vector<double>`, including a NaN and + and - infinity:

```

vector<double> my_data;
my_data.push_back(-1.6);
my_data.push_back(2.0);
my_data.push_back(4.2563);
my_data.push_back(-4.0);
my_data.push_back(numeric_limits<double>::infinity());
my_data.push_back(-numeric_limits<double>::infinity());
my_data.push_back(numeric_limits<double>::quiet_NaN());

try
{ // try'n'catch blocks are needed to ensure error messages from any exceptions are shown.
    svg_1d_plot my_1d_plot; // Construct a plot with all the default constructor values.

    my_1d_plot.title("Default 1D NaN and infinities Demo") // Add a string title of the plot.
        .x_range(-5, 5) // Add a range for the X-axis.
        .x_label("length (m)"); // Add a label for the X-axis.

```

Add the one data series, `my_data` and a description, and how the data points are to marked, here a circle with a diameter of 5 pixels.

```
my_1d_plot.plot(my_data, "1D limits").shape(circlet).size(5);
```

To put a value label against each data point, switch on the option:

```
my_1d_plot.x_values_on(true); // Add data point value labels for the X-axis.
```

To change the default colors (lightgray and whitesmoke) for the 'at limit' point marker to something more conspicuous for this demonstration:

```
my_1d_plot.limit_color(blue);
my_1d_plot.limit_fill_color(pink);
```

To use all these settings, finally write the plot to file.

```
my_1d_plot.write("demo_1d_limits.svg");
```



Note

the +infinity point is marked on the far right of the plot, the -infinity on the far left, but the NaN (Not A Number) is at zero.

To echo the new marker colors chosen:

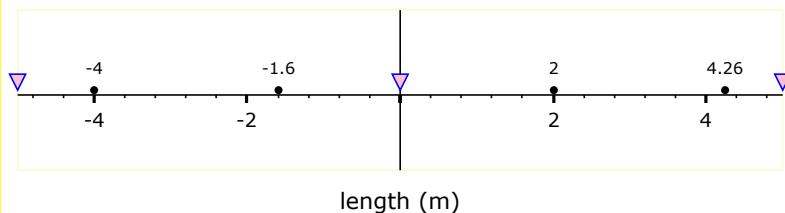
```
cout << "limit points stroke color " << my_1d_plot.limit_color() << endl;
cout << "limit points fill color " << my_1d_plot.limit_fill_color() << endl;
```

Typical output is:

Output:

```
demo_1d_limits.cpp
Linking...
Embedding manifest...
Autorun "j:\Cpp\SVG\debug\demo_1d_limits.exe"
limit points stroke color RGB(0,0,255)
limit points fill color RGB(255,192,203)
Build Time 0:04
```

Default 1D NaN and infinities Demo



See [demo_1d_limits.cpp](#) for full source code and sample output.

Real-life Heat flow data

This example shows some real-life data plotted in 1D and as a boxplot.

```

// An example to show Heat Flow data from
// http://www.itl.nist.gov/div898/handbook/eda/section4/eda4281.htm
// This data set was collected by Bob Zarr of NIST in January, 1990
// from a heat flow meter calibration and stability analysis.

#ifndef _MSC_VER
// Path submitted to cure this.
## pragma warning (disable: 4510) // boost::array<T,N>' : default constructor could not be generated
## pragma warning (disable: 4610) // warning C4610: class 'boost::array<T,N>' can never be instantiated - □
user defined constructor required
#endif

#include <boost/svg_plot/svg_1d_plot.hpp>
using namespace boost::svg;
using boost::svg::svg_1d_plot;
#include <boost/svg_plot/svg_boxplot.hpp>
using boost::svg::svg_boxplot;

#include <boost/svg_plot/show_1d_settings.hpp>
// void boost::svg::show_1d_plot_settings(svg_1d_plot&);

#include <iostream> // for debugging.
using std::cout;
using std::endl;

#include <vector>
using std::vector;

#include <limits>
using std::numeric_limits;

int main()
{
    boost::array<const double, 195> heat_flows = {
        9.206343,
        9.299992,
        9.277895,
        9.305795,
        9.275351,
        9.288729,
        9.287239,
        9.260973,
        9.303111,
        9.275674,
        9.272561,
        9.288454,
        9.255672,
        9.252141,
        9.297670,
        9.266534,
        9.256689,
        9.277542,
        9.248205,
        9.252107,
        9.276345,
        9.278694,
        9.267144,
        9.246132,
        9.238479,
        9.269058,
        9.248239,
        9.257439,
        9.268481,
    };
}

```

```
9.288454,  
9.258452,  
9.286130,  
9.251479,  
9.257405,  
9.268343,  
9.291302,  
9.219460,  
9.270386,  
9.218808,  
9.241185,  
9.269989,  
9.226585,  
9.258556,  
9.286184,  
9.320067,  
9.327973,  
9.262963,  
9.248181,  
9.238644,  
9.225073,  
9.220878,  
9.271318,  
9.252072,  
9.281186,  
9.270624,  
9.294771,  
9.301821,  
9.278849,  
9.236680,  
9.233988,  
9.244687,  
9.221601,  
9.207325,  
9.258776,  
9.275708,  
9.268955,  
9.257269,  
9.264979,  
9.295500,  
9.292883,  
9.264188,  
9.280731,  
9.267336,  
9.300566,  
9.253089,  
9.261376,  
9.238409,  
9.225073,  
9.235526,  
9.239510,  
9.264487,  
9.244242,  
9.277542,  
9.310506,  
9.261594,  
9.259791,  
9.253089,  
9.245735,  
9.284058,  
9.251122,  
9.275385,  
9.254619,
```

```
9.279526,  
9.275065,  
9.261952,  
9.275351,  
9.252433,  
9.230263,  
9.255150,  
9.268780,  
9.290389,  
9.274161,  
9.255707,  
9.261663,  
9.250455,  
9.261952,  
9.264041,  
9.264509,  
9.242114,  
9.239674,  
9.221553,  
9.241935,  
9.215265,  
9.285930,  
9.271559,  
9.266046,  
9.285299,  
9.268989,  
9.267987,  
9.246166,  
9.231304,  
9.240768,  
9.260506,  
9.274355,  
9.292376,  
9.271170,  
9.267018,  
9.308838,  
9.264153,  
9.278822,  
9.255244,  
9.229221,  
9.253158,  
9.256292,  
9.262602,  
9.219793,  
9.258452,  
9.267987,  
9.267987,  
9.248903,  
9.235153,  
9.242933,  
9.253453,  
9.262671,  
9.242536,  
9.260803,  
9.259825,  
9.253123,  
9.240803,  
9.238712,  
9.263676,  
9.243002,  
9.246826,  
9.252107,  
9.261663,
```

```
9.247311,  
9.306055,  
9.237646,  
9.248937,  
9.256689,  
9.265777,  
9.299047,  
9.244814,  
9.287205,  
9.300566,  
9.256621,  
9.271318,  
9.275154,  
9.281834,  
9.253158,  
9.269024,  
9.282077,  
9.277507,  
9.284910,  
9.239840,  
9.268344,  
9.247778,  
9.225039,  
9.230750,  
9.270024,  
9.265095,  
9.284308,  
9.280697,  
9.263032,  
9.291851,  
9.252072,  
9.244031,  
9.283269,  
9.196848,  
9.231372,  
9.232963,  
9.234956,  
9.216746,  
9.274107,  
9.273776  
};  
  
try  
{
```

try'n'catch block Very useful to ensure you see any error messages!

```

vector<double> heat_flow_data(heat_flows.begin(), heat_flows.end()); // Copy from Boost.array to vector.
// This might be useful if data is in a C array.

svg_1d_plot heat_flow_1d_plot; // Construct with all the default constructor values.
heat_flow_1d_plot.document_title("NIST Heat_Flow Data") // This text shows on the browser tab.
    .description("NIST Heat_Flow Data")
    .copyright_date("2008-06-19")
    .copyright_holder("Paul A. Bristow, Bob Zarr, NIST")
    .license("permits", "permits", "requires", "permits", "permits"); // Require notice only.
    //see http://creativecommons.org/licenses/ for details.
// This will generate an XML comment and an author and rights entries thus:
// <!-- SVG Plot Copyright Paul A. Bristow, Bob Zarr, NIST 2008-06-19 -->
// <dc:author><cc:Agent><dc:title>Paul A. Bristow, Bob Zarr, NIST </dc:title> </cc:Agent> </dc:author>
// <dc:rights><cc:Agent><dc:title>Paul A. Bristow, Bob Zarr, NIST</dc:title></cc:Agent></dc:rights>

heat_flow_1d_plot.coord_precision(6); // Some rather precise data real-life,
// so use high precision (even at the expense of slightly longer svg files).

heat_flow_1d_plot
    .title("NIST Heat flow data") // Add title for the plot.
    .x_label("heat flow") // Add a label for the X-axis.
    .x_autoscale(heat_flow_data); // Use autoscale from this data.

heat_flow_1d_plot.plot(heat_flows, "NIST heat flows"); // Add a dataset as an array.
//heat_flow_1d_plot.plot(heat_flow_data, "NIST heat flows"); // Add a dataset.
heat_flow_1d_plot.write("./heat_flow_data.svg");

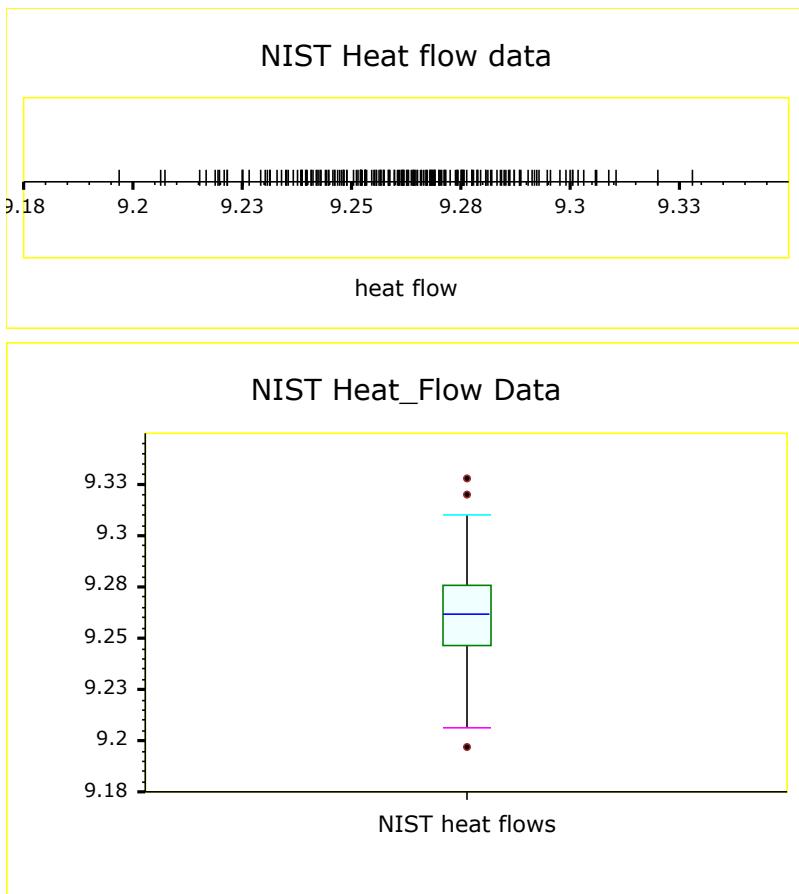
```

Now we use a boxplot to show the same data, including its quartiles and any outliers.

```

svg_boxplot heat_flow_boxplot; // Construct with all the default constructor values.
heat_flow_boxplot.y_yscale(heat_flow_data); // Autoscale using this heat_flow data.
heat_flow_boxplot.title("NIST Heat_Flow Data"); // Give a title.
heat_flow_boxplot.plot(heat_flow_data, "NIST heat flows"); // Add a dataset.
heat_flow_boxplot.write("./heat_flow_data_boxplot.svg");
}
catch(const std::exception& e)
{
    std::cout <<
        "\n""Message from thrown exception was:\n" << e.what() << std::endl;
}

```



See [demo_1d_heat_flow_data.cpp](#) for full source code and sample output.

Demonstration of using 1D data that includes information about its Uncertainty

First we need a few includes to use Boost.Plot:

```
#include <boost/quan/unc.hpp>

// using boost::quan::unc; // Holds value and uncertainty formation.
#include <boost/svg_plot/detail/functors.hpp>
// using boost::svg::detail::unc_1d_convert;
#include <boost/svg_plot/svg_1d_plot.hpp>
using namespace boost::svg;

#include <boost/svg_plot/show_1d_settings.hpp>
//void boost::svg::show_1d_plot_settings(svg_1d_plot&);

#include <iostream>
// using std::cout;
// using std::endl;
#include <vector>
// using std::vector;
#include <iterator>
// using std::ostream_iterator;
#include <algorithm>
// using std::copy;
```

STL vector is used as the container for our two data series, and values are inserted using push_back. Since this is a 1-D plot the order of data values is not important.

```
setUncDefaults(std::cout);
typedef unc<false> uncurl; // Uncertain Uncorrelated (the normal case).
float NaN = std::numeric_limits<float>::quiet_NaN();
std::vector<uncurl> A_times;
A_times.push_back(unc<false>(3.1, 0.02F, 8));
A_times.push_back(unc<false>(4.2, 0.01F, 14, 0U));

short unsigned int t = UNC_KNOWN | UNC_EXPLICIT| DEG_FREE_EXACT | DEG_FREE_KNOWN;

std::vector<unc<false> > B_times;
B_times.push_back(unc<false>(2.1, 0.001F, 30, t)); // Value, uncertainty, degrees of freedom and type known.
// (But use of type is not yet implemented.)
B_times.push_back(unc<false>(5.1, 0.025F, 20, 0U)); // Value, uncertainty, and degrees of freedom known - the usual case.
B_times.push_back(unc<false>(7.8, 0.0025F, 1, 0U)); // Value and uncertainty known, but not degrees of freedom.
B_times.push_back(unc<false>(3.4, 0.03F, 1, 0U)); // Value and uncertainty known, but not degrees of freedom.
//B_times.push_back(unc<false>(6.9, 0.0F, 0, 0U)); // Only value known - no information available about uncertainty but treated as exact.
B_times.push_back(unc<false>(5.9, NaN, 1, 0U)); // Only value known - uncertainty explicit NaN meaning no information available about uncertainty.
// So in both cases show all possibly significant digits (usually 15).
// This is ugly on a graph, so best to be explicit about uncertainty.

std::vector<unc<false> > C_times;
C_times.push_back(unc<false>(2.6, 0.1F, 5, 0U));
C_times.push_back(unc<false>(5.4, 0.2F, 11, 0U));
```

Echo the values input:

```
std::copy(A_times.begin(), A_times.end(), std::ostream_iterator<uncurl>(std::cout, " "));
std::cout << std::endl;
std::copy(B_times.begin(), B_times.end(), std::ostream_iterator<uncurl>(std::cout, " "));
std::cout << std::endl;
```

The constructor initializes a new 1D plot, called my_plot, and also sets all the very many defaults for axes, width, colors, etc.

```
svg_1d_plot my_plot;
```

A few (member) functions that are set (using concatenation or chaining) should be fairly self-explanatory:

- * `.title()` provides a title at the top **for** the whole plot,
- * `.legend_on(true)` will mean that titles of data series **and** markers will display in the legend box.
- * `.x_range(-1, 11)` sets the axis limits from **-1** to **+11** (instead of the **default** **-10** to **+10**).
- * `.background_border_color(blue)` sets just one of the very many other options.

```
my_plot.autoscale_check_limits(false); // Default is true.
my_plot.autoscale_plusminus(2); // default is 3.
my_plot.confidence(0.01); // Change alpha from default 0.05 == 95% to 0.01 == 99%.

my_plot
.image_x_size(600)
.image_y_size(300)
.plot_window_on(true)
.background_border_color(blue)
.plot_border_color(yellow)
.plot_border_width(1)
//.x_ticks_on_window_or_axis(0) // now the default.
.legend_on(false)
.title("A, B and C Times")
.x_range(0, 10)
.x_label("times (sec)")
.x_values_on(true)
// .x_values_precision(0) // Automatic number of digits of precision.
.x_values_precision(2) // User chosen precision.
.x_values_rotation(slopeup)
.x_plusminus_on(true)
.x_plusminus_color(blue)
.x_addlimits_on(true)
.x_addlimits_color(purple)
.x_df_on(true)
.x_df_color(green)
.x_autoscale(B_times) // Note that this might not be right scaling for A_times and/or C_times.
;
```

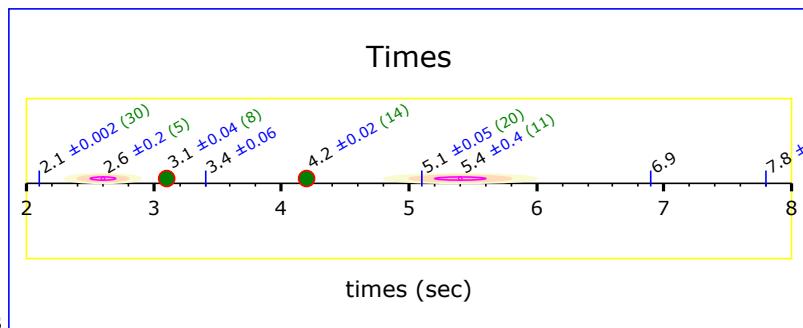
Then we add our data series, and add optional data series titles (very helpful if we want them to show on the legend).

The A_times mark data points with a red border circle with a green fill, The B_times use a blue vertical line, while C_times use an ellipse whose width (x radius) is from the uncertainty, 1st standard deviation shows as ellipse in magenta, and 2nd as yellow. All the data points are also labeled with their value, and uncertainty and degrees of freedom if known.

```
my_plot.plot(A_times, "A").shape(circlet).size(10).stroke_color(red).fill_color(green);
my_plot.plot(B_times, "B").shape(vertical_line).stroke_color(blue);
my_plot.plot(C_times, "C").shape(unc_ellipse).fill_color(lightyellow).stroke_color(magenta);
```

Finally, we can write the SVG to a file of our choice.

```
std::string svg_file = (my_plot.legend_on() == true) ?
    "./demo_1d_uncertainty_legend.svg" : "./demo_1d_uncertainty.svg";
my_plot.write(svg_file);
std::cout << "Plot written to file " << svg_file << std::endl;
show_1d_plot_settings(my_plot);
```



The resulting plot is

See [demo_1d_uncertainty.cpp](#) for full source code and sample output.

2D Tutorial

Simple Code Example

```
#include <boost/svg_plot/svg_2d_plot.hpp>
using namespace boost::svg;

#include <map>
using std::multimap; // A more complicated STL sorted container.

int main()
{
    multimap<double, double> map1; // 1st data series.
    multimap<double, double> map2; // 2nd data series.

    // Random data used purely for this example.
    map1[1.] = 3.2; // 1st data series.
    map1[1.] = 5.4;
    map1[7.3] = 9.1;

    map2[3.1] = 6.1; // 2nd data series.
    map2[5.4] = 7.;

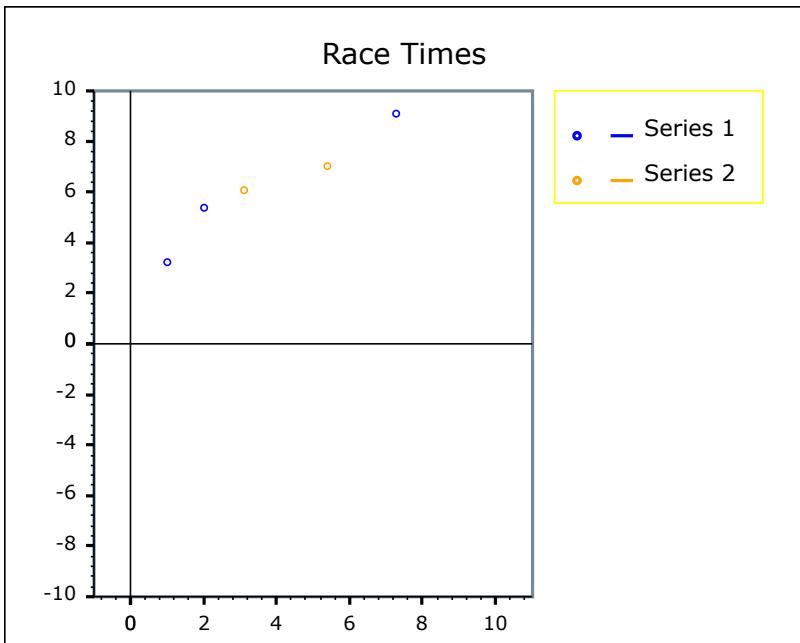
    svg_1d_plot my_plot;

    my_plot.title("Race Times")
        .legend_on(true)
        .x_range(-1, 11)
        .background_border_color(black);

    my_plot.plot(map1, "Series 1", blue); // Data point marker color is blue.
    my_plot.plot(map2, "Series 2", orange); // Data point marker color is orange.

    my_plot.write("simple_2d.svg");
    return 0;
}
```

Resulting Simple_2D Example Image



Simple_2D Example Breakdown

Let's examine what this does.

```
svg_2d_plot my_plot;
```

This constructor initializes a new 2D plot. This also sets the very many default values.

We could accept all of these and just plot one series of data with

```
my_plot.plot(map1, "Series 1"); // Data point marker color is default black.
```

but we can also add a few optional details:

```
my_plot.title("Race Times")
    .legend_on(true)
    .x_range(-1, 11)
    .background_border_color(black);
```

All of the setter methods are fairly self-explanatory. To walk through it once,

- The title, which will appear at the top of the graph, will say "Race Times".
- `legend_on(true)` means that the legend box will show up (at top right by default).
- `x_range(-1, 11)` means that the axis displayed will be between -1 and 11, as you can see in the above images.
- `background_border_color(black)` sets the border around the image to black. Ordinarily it is left to be the color of the whole image background (and so no border of the plot area will be visible).

```
my_plot.plot(map1, "Series 1", blue); my_plot.plot(map2, "Series 2", orange);
```

This draws `map1` and `map2` to `my_plot`.

The name of the serieses are "Series 1" and "Series 2", and that text will show in the legend box.

(As many containers as you want can be drawn to `my_plot`. Eventually the plot will become cluttered and confusing, when creating other plot(s) becomes more sensible!)

Finally

```
my_plot.write("simple_2d.svg");
```

writes plot `my_plot` to the file "simple_2d.svg".

You can view this file with most internet browsers, and other programs too.

Tutorial: Fuller Layout Example

```
#include <boost/svg_plot/svg_2d_plot.hpp>
using namespace boost::svg;
#include <map>
using std::multimap;
#include <cmath> // for sqrt

// Some example functions:
double f(double x)
{
    return sqrt(x); // Note: negative values of x will all return NaN.
}

double g(double x)
{
    return -2 + x * x;
}

double h(double x)
{
    return -1 + 2 * x;
}

int main()
{
    multimap<double, double> data1, data2, data3;

    for(double i = 0; i <= 10.; i += 1.)
    { // Evaluate the functions as demonstration data.
        data1[i] = f(i);
        data2[i] = g(i);
        data3[i] = h(i);
    }

    svg_2d_plot my_plot; // Construct a new plot.

    // Override a few default settings with our choices:

    // Size of SVG image and X and Y range settings.
    my_plot.image_size(700, 500)
        .x_range(-1, 10)
        .y_range(-5, 100)

    // Text settings.
    my_plot.title("Plot of Mathematical Functions")
        .title_font_size(25)
        .x_label("Time in Months");

    // Commands.
```

```

my_plot.legend_on(true)
    .plot_window_on(true)
    .x_label_on(true)
    .x_major_labels_on(true);

// Color settings.
my_plot.background_color(svg_color(67, 111, 69))
    .legend_background_color(svg_color(207, 202, 167))
    .legend_border_color(svg_color(102, 102, 84))
    .plot_background_color(svg_color(136, 188, 126))
    .title_color(white);

// X axis settings.
my_plot.x_major_interval(2)
    .x_major_tick_length(14)
    .x_major_tick_width(1)
    .x_minor_tick_length(7)
    .x_minor_tick_width(1)
    .x_num_minor_ticks(3)

// Y axis settings.
.y_major_tick(10)
.y_num_minor_ticks(2);

// Legend settings.
my_plot.legend_title_font_size(15);

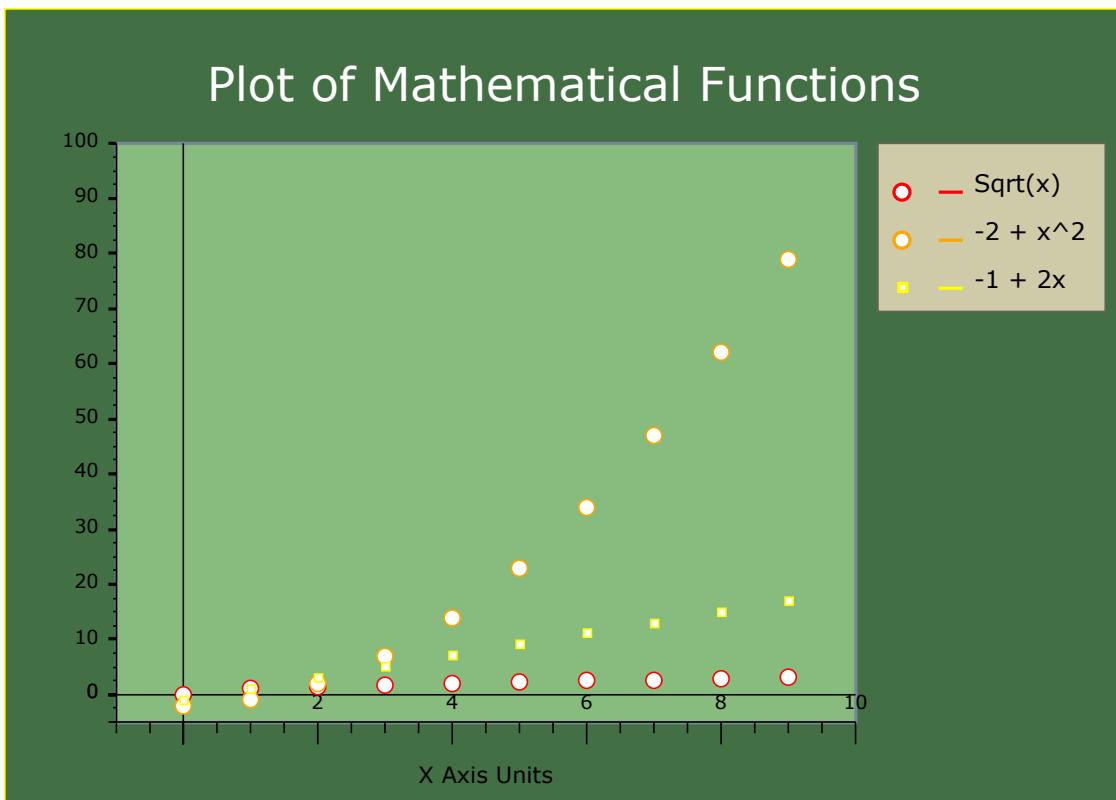
// Add the 3 data series to the plot, using different markers and line colors.
my_plot.plot(data1, "Sqrt(x)").fill_color(red);
my_plot.plot(data2, "-2 + x^2").fill_color(orange).size(5);
my_plot.plot(data3, "-1 + 2x").fill_color(yellow).bezier_on(true).line_color(blue).shape(square);
// Note how the options can be chained.

my_plot.write("2d_full.svg");

return 0;
}

```

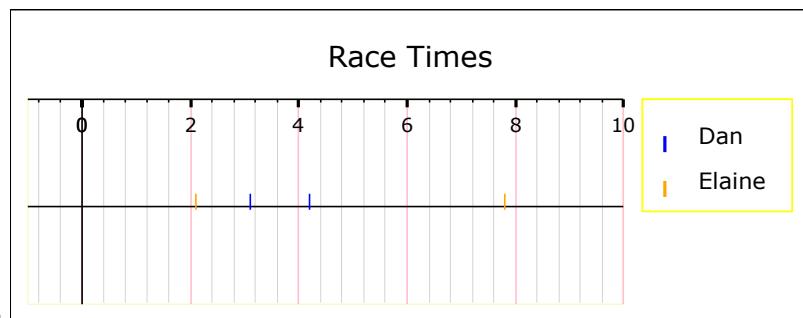
This produces the following image:



A little bit of color customization goes a **very** long way!

Tutorial: 2D Special Features

Y-Axis ticks and tick value label position



[demo_1d_x_external_1] producing this plot

See [demo_1d_x_external.cpp](#) for full source code.

X-Axis Tick value labels precision, iosflags, font family and size

As ever, we need a few includes to use Boost.Plot and an STL container.

Some fictional data is pushed into an STL container, here `vector<double>`:

```

vector<double> my_data;
my_data.push_back(-1.6);
my_data.push_back(4.2563);
my_data.push_back(0.00333974);
my_data.push_back(5.4);
my_data.push_back(6.556);

try
{ // try'n'catch blocks are needed to ensure error messages from any exceptions are shown.
    svg_1d_plot my_1d_plot; // Construct a plot with all the default constructor values.

    my_1d_plot.title("Demo 1D Tick Values") // Add a string title of the plot.
    .x_range(-5, 10) // Add a range for the X-axis.
    .x_label("temp (\u00B0C)") // Add a label for the X-axis, using Unicode degree symbol.

```

Add the one data series, `my_data` and a description, and how the data points are to be marked, a circle with a diameter of 7 pixels.

```
my_1d_plot.plot(my_data, "1D Values").shape(circlet).size(7);
```

If the default size and color are not to your taste, set more options, like:

```

my_1d_plot.size(500, 150) // Change plot window from the default image size.
// Change the X-axis label style:
.x_axis_label_color(green)
.x_label_font_family("Arial")
.x_label_font_size(18)

// Change the style of the X (major) ticks:
.x_ticks_values_color(magenta)
//.x_ticks_values_font_family("Times New Roman")
.x_ticks_values_font_family("arial")
.x_ticks_values_font_size(15)

```

The format of the tick value labels may not suit your data and its range, so we can use the normal iostream precision and ioflags to change, here to reduce the number of digits used from default precision 6 down to a more readable 2, reducing the risk of collisions between adjacent values. If values are very close to each other (a small range on the axis), a higher precision will be needed to differentiate them. We could also prescribe the use of scientific format and force a positive sign: By default, any unnecessary spacing-wasting zeros in the exponent field are removed.

If, perversely, the full 1.123456e+012 format is required, the stripping can be switched off with: `my_1d_plot.x_ticks_values_strip_e0s(false);`

```

// Change the format from the default "-4", "-2", "0" "2", "4" ...
// (which makes a 'best guess' at the format)
// to "-4.00", "-2.00", ..." +2.00", "4.00"
// showing trailing zeros and a leading positive sign.
.x_ticks_values_ioflags(ios_base::fixed | std::ios::showpos)
.x_ticks_values_precision(1) //

// One could use ios_base::scientific for e format.
//.x_ticks_values_ioflags(ios_base::fixed | std::ios::showpos )
//.x_ticks_values_ioflags(std::ios::showpoint | std::ios::showpos)
//.x_ticks_values_ioflags(std::ios::scientific)
;

```

To use all these settings, finally write the plot to file.

```
my_1d_plot.write("demo_1d_tick_values.svg");
```

If chosen settings do not have the effect that you expect, it may be helpful to display them.

(All the myriad settings can be displayed with `show_1d_plot_settings(my_1d_plot.)`)

```
//show_1d_plot_settings(my_1d_plot);
using boost::svg::detail::operator<<;
cout << "my_1d_plot.size() " << my_1d_plot.size() << endl;
cout << "my_1d_plot.x_size() " << my_1d_plot.x_size() << endl;
cout << "my_1d_plot.y_size() " << my_1d_plot.y_size() << endl;

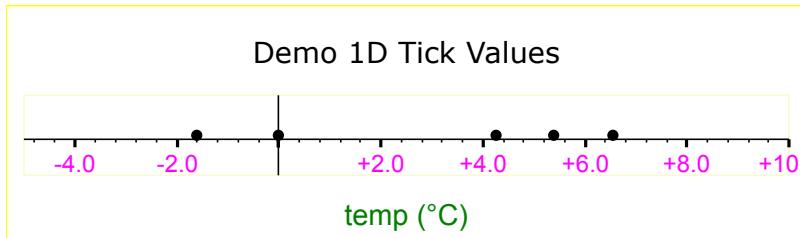
cout << "my_1d_plot.x_axis_label_color() " << my_1d_plot.x_axis_label_color() << endl;
cout << "my_1d_plot.x_label_font_family() " << my_1d_plot.x_label_font_family() << endl;
cout << "my_1d_plot.x_label_font_size() " << my_1d_plot.x_label_font_size() << endl;

cout << "my_1d_plot.x_ticks_values_font_family() " << my_1d_plot.x_ticks_values_font_family() << endl;
cout << "my_1d_plot.x_ticks_values_font_size() " << my_1d_plot.x_ticks_values_font_size() << endl;
cout << "my_1d_plot.x_ticks_values_color() " << my_1d_plot.x_ticks_values_color() << endl;

cout << "my_1d_plot.x_ticks_values_precision() " << my_1d_plot.x_ticks_values_precision() << endl;
cout << "my_1d_plot.x_ticks_values_ioflags() " << hex << my_1d_plot.x_ticks_values_ioflags() << endl;
```

See `demo_1d_ticks_values.cpp` for full source code.

producing this plot



See `demo_1d_tick_values.cpp` for full source code.

Y-Axis Grid Lines

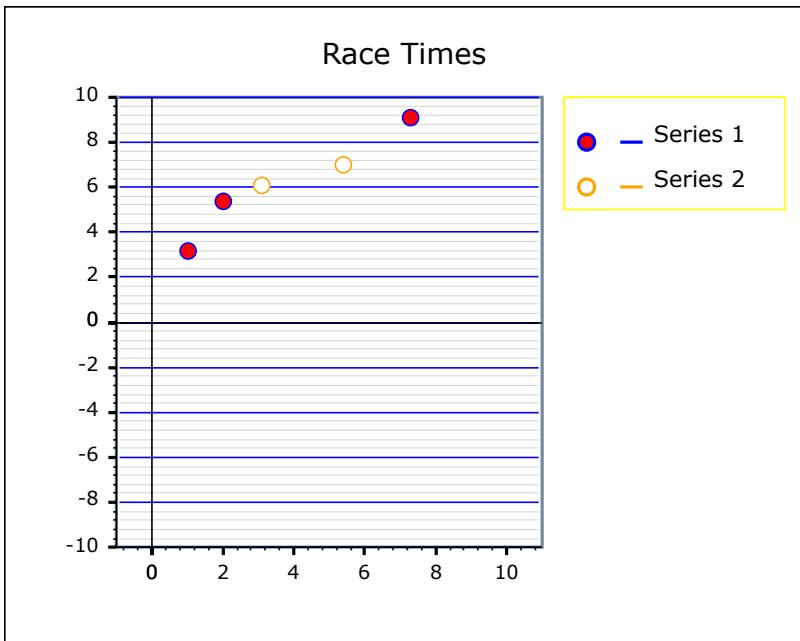
If you would like horizontal Y grid lines that go across the graph, you can make the following call to `svg_2d_plot`:

```
my_plot.y_major_grid_on(true)
.y_minor_grid_on(true);
```

To style it, you could use the following calls:

```
my_plot.y_major_grid_color(lightgray) // Darker color for major grid.
.y_minor_grid_color(whitesmoke); // Lighter color for minor grid.
```

This will produce the following plot image:



The source of this examples is at [..../example/2d_y_grid.cpp](#)

Similarly you can have horizontal and/or vertical lines:

```
my_plot.x_major_grid_on(true)
.x_minor_grid_on(true);
```

and color it similarly for faint blues like old fashioned graph paper:

```
my_plot.y_major_grid_color(svg_color(200, 220, 255)) // Darker color for major grid.
.y_minor_grid_color(svg_color(240, 240, 255)); // lighter color for minor grid.
```

and to make the major grid much wider than the minor grid:

```
my_plot.y_major_grid_width(4)
.y_minor_grid_width(1)
```

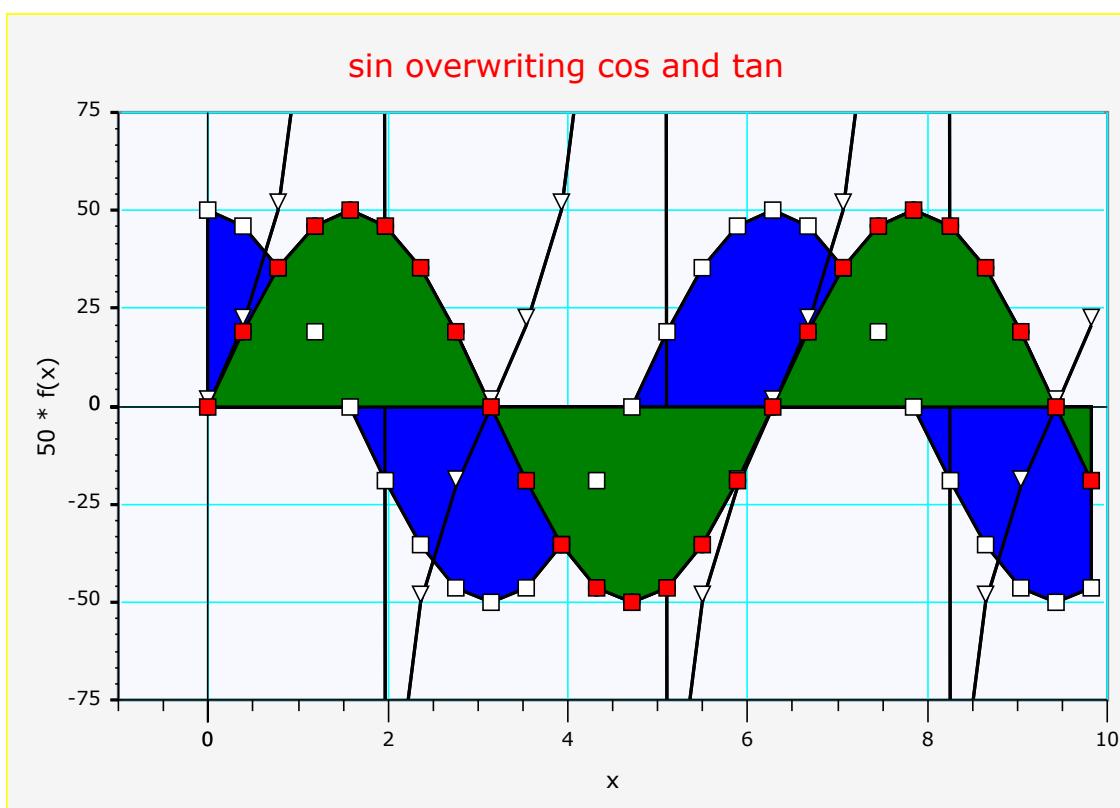
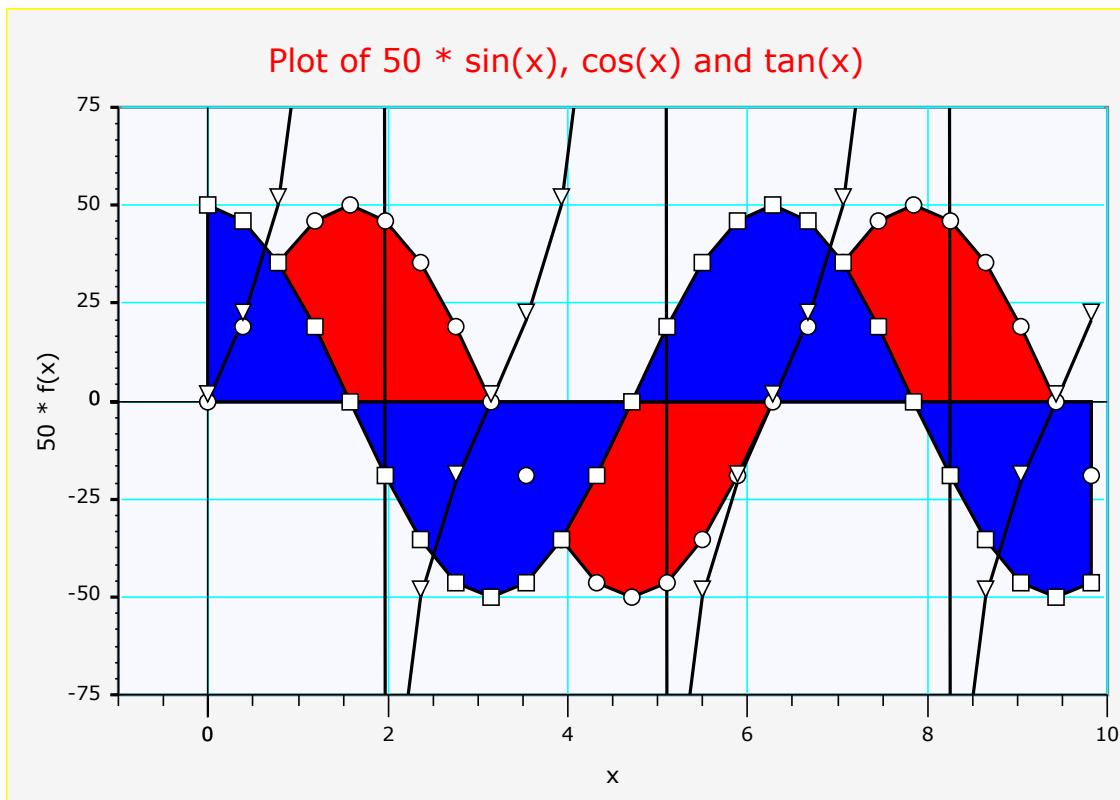
Fill the area between the plot and the axis

When there is a call to the plot() method, define `area_fill_color`

```
multimap<double, double> my_data;
svg_2d_plot my_plot;

my_plot.plot(my_data, "Data", area_fill_color(red));
```

This produces the following images:



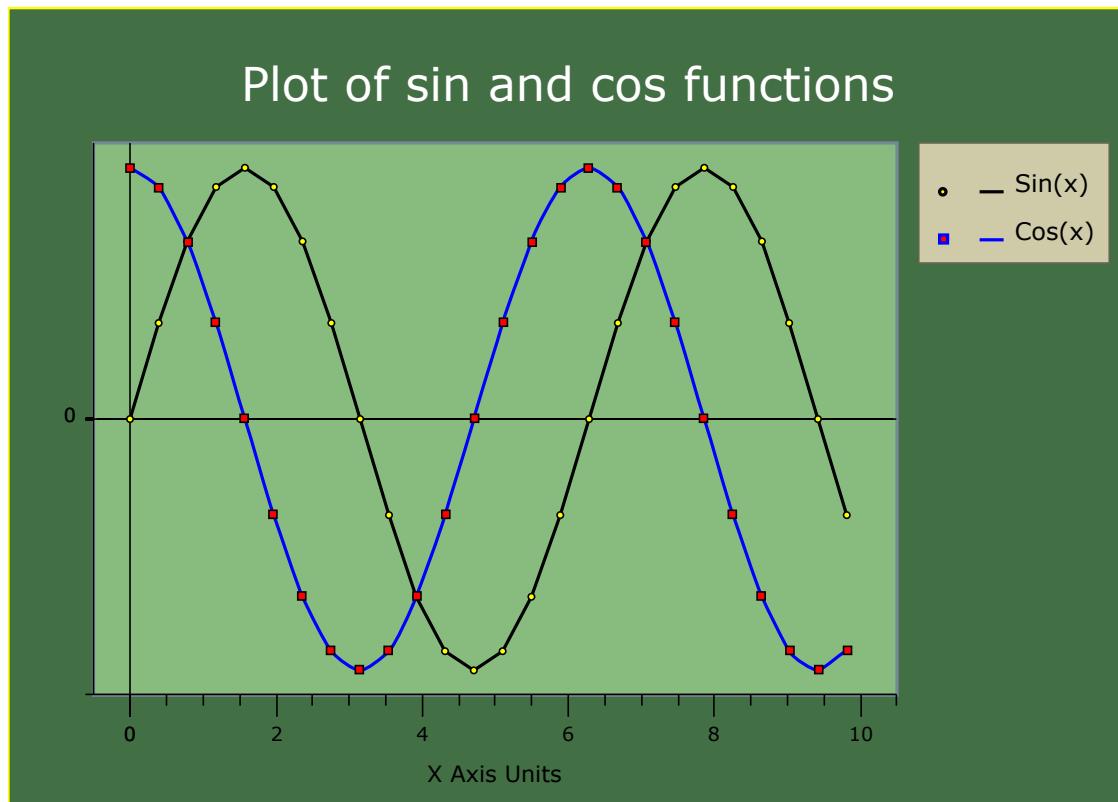
from source at [../../example/2d_area_fill.cpp](#)

Curve Interpolation

If you would like an **interpolated curve** shown over your data (rather than the default straight lines joining points), simply use the following command:

```
my_plot.plot(data, "Series 1", bezier_on(true);
```

This produces something like the following images:



Warning

The `_bezier_curve` feature still displays undesired behavior in extreme circumstances. Do not use this feature with curves that have a numeric_limit value (`-NaN`, for example), or with data that has high irregularity in X-Axis spacing (for example, a clump of points between (0, 1) on the X axis, with the next one at 100 on the X axis.) In general, curves must be reasonably well-behaved to be smoothable. In general, using more data points will make smoothing work better, but will increase svg file sizes.

2-D Data Values Examples

Showing 2d Data Values Examples

As always, we need a few includes to use Boost.Plot

```
#include <boost/svg_plot/svg_2d_plot.hpp>
using namespace boost::svg;
using boost::svg::svg_2d_plot;

#include <iostream>
using std::cout;
using std::endl;
using std::dec;
using std::hex;

#include <map>
using std::map;
```

Some fictional data is pushed into an STL container, here map: This example uses a single map to demonstrate autoscaling. We create a map to hold our data series.

```
map<const double, double> my_data;
```

Inserting some fictional values also sorts the data. The index value in [] is the x value.

```
//my_data[1.1] = 3.2;
//my_data[7.3] = 9.1;
my_data[2.12] = 2.4394;
my_data[5.47] = 5.3861;

try
{ // try'n'catch blocks are needed to ensure error messages from any exceptions are shown.
    svg_2d_plot my_2d_plot; // Construct a plot with all the default constructor values.

    my_2d_plot.title("Default 2d Values Demo") // Add a string title of the plot.
        .x_range(-5, 10) // Add a range for the X-axis.
        .x_label("length (m)"); // Add a label for the X-axis.
```

Add the one data series, my_data and a description, and how the data points are to be marked, here a circle with a diameter of 5 pixels.

```
my_2d_plot.plot(my_data, "2d Values").shape(circlet).size(5).line_on(false);
```

To put a value label against each data point, switch on the option:

```
//my_2d_plot.x_values_on(true); // Add a label for the X-axis.
//my_2d_plot.y_values_on(true); // Add a label for the X-axis.
my_2d_plot.xy_values_on(true); // Add a label for the X-axis.
```

If the default size and color are not to your taste, set more options, like:

```
my_2d_plot.x_values_font_size(16) // Change font size for the X-axis value labels.
    .x_values_font_family("Times New Roman") // Change font for the X-axis value labels.
    .x_values_color(red); // Change X values color from default black to red.

my_2d_plot.y_values_font_size(14) // Change font size for the Y-axis value labels.
    .y_values_font_family("Arial") // Change font for the Y-axis value labels.
    .y_values_color(blue); // Change Y color from default black to blue.
```

The format of the values may also not be ideal, so we can use the normal iostream precision and ioflags to change, here to reduce the number of digits used from default precision 6 down to a more readable 2, reducing the risk of collisions between adjacent values.

(Obviously the most suitable precision depends on the range of the data points. If values are very close to each other, a higher precision will be needed to differentiate them). For measurement of typical precision, 2 or 3 decimal places will suffice.

```
my_2d_plot.x_values_precision(3); // precision label for the X-axis value label.  
my_2d_plot.y_values_precision(5); // precision label for the Y-axis value label.
```

We can also prescribe the use of scientific, fixed format and/or force a positive sign:

```
//my_2d_plot.x_values_ioflags(std::ios::scientific | std::ios::showpos);  
//my_2d_plot.x_values_ioflags(std::ios::scientific);  
//my_2d_plot.y_values_ioflags(std::ios::fixed);
```

By default, any unnecessary spacing-wasting zeros in the exponent field are removed. Stripping "e+000" may appear to mean that the normal effect of scientific is not working. (If, probably perversely, the full 1.123456e+012 format is required, the stripping can be switched off with: my_2d_plot.x_labels_strip_e0s(false);)

In general, sticking to the default ioflags usually produces the neatest presentation of values.

```
my_2d_plot.x_plusminus_on(true); // unc label for the X-axis value label.  
my_2d_plot.x_df_on(true); // df label for the X-axis value label.  
  
my_2d_plot.y_plusminus_on(true); // unc label for the X-axis value label.  
my_2d_plot.y_df_on(true); // df label for the X-axis value label.
```

The default value label is horizontal, centered above the data point marker, but, depending on the type and density of data points, and the length of the values (controlled in turn by the precision and ioflags in use), it is often clearer to use a different orientation. This can be controlled in steps of 45 degrees, using an 'enum rotate_style`.

- **leftward** - writing level with the data point but to the left.
- **rightward** - writing level with the data point but to the right.
- **uphill** - writing up at 45 degree slope is often a good choice,
- **upward** - writing vertically up and
- **backup** - writing to the left are also useful.

(For 1-D plots other directions are less attractive, placing the values below the horizontal Y-axis line, but for 2-D plots all writing orientations can be useful).

```
my_2d_plot.x_values_rotation(rightward); // Orientation for the Y-axis value labels.  
//my_2d_plot.x_values_rotation(horizontal); // Orientation for the X-axis value labels.  
//my_2d_plot.x_values_rotation(uphill); // Orientation for the X-axis value labels.  
// my_2d_plot.y_values_rotation(downhill); // Orientation for the Y-axis value labels.  
//my_2d_plot.x_values_rotation(leftward); // Orientation for the X-axis value labels.  
//my_2d_plot.y_values_rotation(rightward); // Orientation for the Y-axis value labels.  
  
// my_2d_plot.x_plusminus_on(true); // Show Uncertainty for X-axis value labels.  
// my_2d_plot.x_df_on(true); // Show Degrees of freedom (n-1) for X-axis value labels.  
//my_2d_plot.y_plusminus_on(true); // Uncertainty for X-axis value labels.
```

To use all these settings, finally write the plot to file.

```
my_2d_plot.write("demo_2d_values.svg");
```

If chosen settings do not have the expected effect, is may be helpful to display them.

(All settings can be displayed with `show_2d_plot_settings(my_2d_plot)`.)

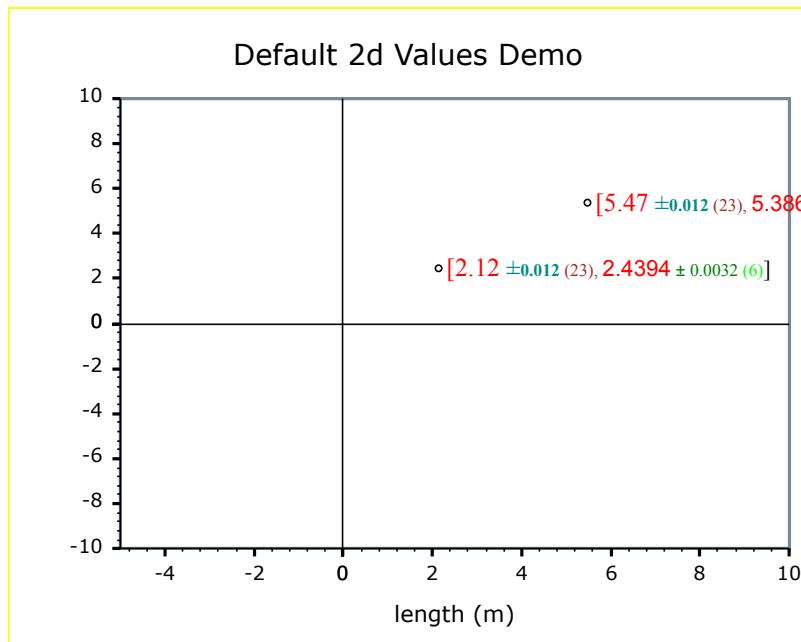
```
cout << "my_2d_plot.x_values_font_size() " << my_2d_plot.x_values_font_size() << endl;
cout << "my_2d_plot.x_values_font_family() " << my_2d_plot.x_values_font_family() << endl;
cout << "my_2d_plot.x_values_color() " << my_2d_plot.x_values_color() << endl;
cout << "my_2d_plot.x_values_precision() " << my_2d_plot.x_values_precision() << endl;
cout << "my_2d_plot.x_values_ioflags() " << hex << my_2d_plot.x_values_ioflags() << dec << endl;

cout << "my_2d_plot.y_values_font_size() " << my_2d_plot.y_values_font_size() << endl;
cout << "my_2d_plot.y_values_font_family() " << my_2d_plot.y_values_font_family() << endl;
cout << "my_2d_plot.y_values_color() " << my_2d_plot.y_values_color() << endl;
cout << "my_2d_plot.y_values_precision() " << my_2d_plot.y_values_precision() << endl;
cout << "my_2d_plot.y_values_ioflags() " << hex << my_2d_plot.y_values_ioflags() << dec << endl;
```

Output:

```
demo_2d_values.cpp
Linking...
Embedding manifest...
Autorun "j:\Cpp\SVG\debug\demo_2d_values.exe"
my_2d_plot.x_values_font_size() 16
my_2d_plot.x_values_font_family() Times New Roman
my_2d_plot.x_values_color() RGB(255,0,0)
my_2d_plot.x_values_precision() 3
my_2d_plot.x_values_ioflags() 200
my_2d_plot.y_values_font_size() 14
my_2d_plot.y_values_font_family() Arial
my_2d_plot.y_values_color() RGB(0,0,255)
my_2d_plot.y_values_precision() 5
my_2d_plot.y_values_ioflags() 200
```

And the plot:



Showing 2d Data 'at limit' Values Examples

Some values are 'at limit' - infinite, NotANumber (NaN) or near the maximum possible for the floating-point type. These values are displayed differently (and not used for autoscaling).

An example to demonstrate plotting 2D 'at limits' values including NaN and + and - infinity.

As ever, we need the usual includes to use Boost.Plot.

Some fictional data is pushed into an STL container, here map:

```
map<double, double> my_data;
```

Inserting some fictional values also sorts the data. The map index value in [] is the x value, so mydata[x] = y.

First some normal valued points, not 'at limits'.

```
my_data[1.1] = 3.2;
my_data[4.3] = 3.1;
my_data[0.25] = 1.4;
```

Now some values including + and - infinity:

```
my_data[3] = numeric_limits<double>::quiet_NaN(); // marker at x = 3, y = 0
my_data[0.] = numeric_limits<double>::quiet_NaN(); // Marker at 0,0
my_data[1.] = numeric_limits<double>::infinity(); // Marker at 1, top
my_data[-1] = -numeric_limits<double>::infinity(); // Marker at -1, bottom
my_data[+numeric_limits<double>::infinity()] = +1.; // Marker at right, 1
my_data[-numeric_limits<double>::infinity()] = -1.; // Marker at left, -1
my_data[+(numeric_limits<double>::max)()] = +2.; // Marker at right, 2
my_data[-(numeric_limits<double>::max)()] = +2.; // Marker at left, 2
my_data[-(numeric_limits<double>::max)/2] = +3.; // Value near to max, marker left, 3
my_data[numeric_limits<double>::infinity()] = numeric_limits<double>::infinity(); // Top right.
my_data[-numeric_limits<double>::infinity()] = -numeric_limits<double>::infinity(); // Bottom left.
```

Caution



Using map (rather than multimap that allows duplicates) some assignments values overwrite, and so not all display as they do individually. In particular, an X value of quiet_NaN() causes a overwrite of the lowerest value (because NaNs never compare equal). So avoid NaN as an X value.

```
try
{ // try'n'catch blocks are needed to ensure error messages from any exceptions are shown.
svg_2d_plot my_2d_plot; // Construct a plot with all the default constructor values.

my_2d_plot.title("Default 2D 'at limits' NaN and infinities Demo") // Add a string title of the plot.
.x_range(-5, 5) // Add a range for the X-axis.
.y_range(-5, 5) // Add a range for the Y-axis
.x_label("time (s)") // Add a label for the X-axis.
```

Add the one data series, my_data and a description, and how the data points are to be marked, here a circle with a diameter of 5 pixels.

```
svg_2d_plot_series& my_series = my_2d_plot.plot(my_data, "2D limits").shape(circlet).size(5);
```

We can also keep note of the plot series and use this to interrogate how many normal and how many 'at limit' values.

```
cout << my_series.values_count() << " normal data values in series." << endl;
cout << my_series.limits_count() << " 'at limits' data values in series." << endl;
```

To put a value label against each data point, switch on the option:

```
// my_2d_plot.x_values_on(true).y_values_on(true).x_values_font_size(12).y_values_font_size(12); // Add the X-axis and Y-axis values.  
// This displays x horizontally and Y downward.  
my_2d_plot.xy_values_on(true).x_values_font_size(12).y_values_font_size(12); // Add the X-axis and Y-axis values.  
// This displays X above and Y below.
```

To change the default colors (lightgray and whitesmoke) for the 'at limit' point marker to something more conspicuous for this demonstration:

```
my_2d_plot.limit_color(blue);  
my_2d_plot.limit_fill_color(pink);
```

To use all these settings, finally write the plot to file.

```
my_2d_plot.write("demo_2d_limits.svg");
```

Note the +infinity point is marked on the far right of the plot, the -infinity on the far left, but the NaN (Not A Number is at zero).

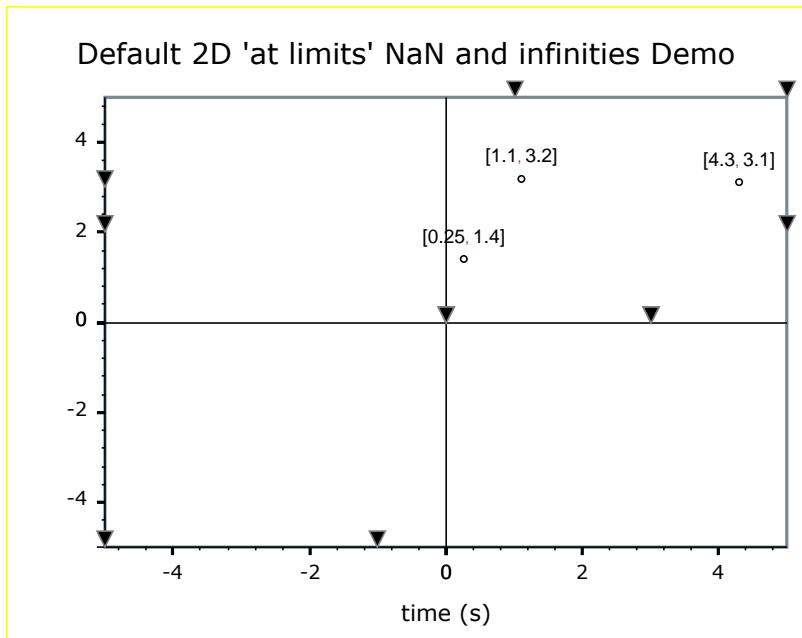
To echo the new marker colors chosen:

```
cout << "limit points stroke color " << my_2d_plot.limit_color() << endl;  
cout << "limit points fill color " << my_2d_plot.limit_fill_color() << endl;
```

Output:

```
Autorun "j:\Cpp\SVG\Debug\demo_2d_limits.exe"  
3 normal data values in series.  
9 'at limits' data values in series.  
limit points stroke color RGB(0,0,255)  
limit points fill color RGB(255,192,203)  
*/
```

And the plot:



2-D Autoscaling Examples

Autoscale 2D Examples

First we need a few includes to use Boost.Plot

```

#include <boost/svg_plot/svg_2d_plot.hpp>
using namespace boost::svg;
#include <utility>
using std::pair;
#include <map>
using std::map;
#include <set>
using std::multiset;
#include <iostream>
using std::cout;
using std::endl;

#include <limits>
using std::numeric_limits;

#include <boost/math/special_functions.hpp>
using boost::math::isfinite;

// Getting the max and min of x and y data points.
template <typename T> // T an STL container: array, vector ...
void s(T& container, // Container Data series to plot - entire container.
       // (not necessarily ordered, so will find min and max).
       double* x_min, double* x_max,
       double* y_min, double* y_max
)
{
    typedef typename T::const_iterator iter;
    std::pair<iter, iter> result = boost::minmax_element(container.begin(), container.end());
    // minmax_element is efficient for maps because can use knowledge of being sorted,
    // BUT only if it can be assumed that no values are 'at limits',
    // infinity, NaN, max_value, min_value, denorm_min.
    // Otherwise it is necessary to inspect all values individually.
    std::pair<const double, double> px = *result.first;
    std::pair<const double, double> py = *result.second;
    *x_min = px.first;
    *x_max = py.first;
    *y_min = px.second;
    *y_max = py.second;

    cout << "s x_min " << *x_min << ", x_max " << *x_max << endl; // x_min 1, x_max 7.3
    cout << "s y_min " << *y_min << ", y_max " << *y_max << endl; // y_min 3.2, y_max 9.1
} // template <class T> int scale_axis T an STL container: array, vector ...

```

This example uses a single map to demonstrate autoscaling. We create a map to hold our data series.

```
std::map<const double, double> my_map;
```

Inserting some fictional values also sorts the data. The index value in [] is the x value.

```
my_map[1.1] = 3.2;
my_map[7.3] = 9.1;
my_map[2.1] = 5.4;
```

Also include some 'at limits' values that might confuse autoscaling.

```
my_map[99.99] = std::numeric_limits<double>::quiet_NaN();
my_map[999.9] = std::numeric_limits<double>::infinity();
my_map[999.] = +std::numeric_limits<double>::infinity();
```

Next a 2D plot is created using defaults for the very many possible settings.

```
try
{ // try'n'catch clocks are needed to ensure error messages from any exceptions are shown.
```

Construct myplot and add at least a title, specify the both x and y axes are to use autoscaling, and add the one data series to be plotted.

```
svg_2d_plot my_plot;
my_plot.title("Autoscale example"); // Add a title.
my_plot.xy_autoscale(my_map); // Specify that both x and y axes are to use autoscaling,
my_plot.plot(my_map); // Add the one data series to be plotted
my_plot.write("./auto_2d_plot.svg"); // And write the SVG image to a file.
```

We can show the ranges chosen by autoscaling;

```
cout << "X min " << my_plot.x_range().first << ", X max " << my_plot.x_range().second << endl;
cout << "Y min " << my_plot.y_range().first << ", Y max " << my_plot.y_range().second << endl;
```

If we know that there were no 'at limits' values, we could have chosen to skip the checks. This might be important for speed if there were thousands of data values.

```
my_plot.autoscale_check_limits(false); // Skip checks for speed.
```

The possible cost is that it will fail at run-time if there are any infinite or NaNs. We could also choose to autoscale either of the axes separately, for example:

```
my_plot.y_autoscale(0.4, 9.3); // autoscale using two doubles.
```

which will chose a neater scale from 0 to 10 for the Y axis.

```
my_plot.write("./auto_2d_plot2.svg"); // And write another SVG image to a file.
```

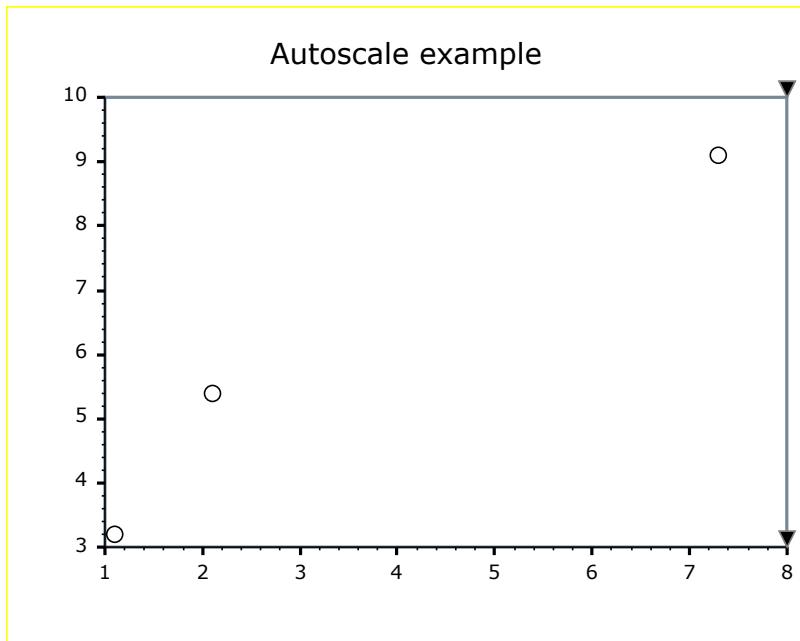
We can show the ranges chosen by autoscaling;

```
cout << "X min " << my_plot.x_range().first << ", X max " << my_plot.x_range().second << endl;
cout << "Y min " << my_plot.y_range().first << ", Y max " << my_plot.y_range().second << endl;
```

Output:

```
Autorun "j:\Cpp\SVG\Debug\auto_2d_plot.exe"
Checked: x_min 1.1, x_max 7.3, y_min 3.2, y_max 9.1, 3 'good' values, 3 values at limits.
X min 1, X max 8
Y min 3, Y max 10
X min 1, X max 8
Y min 0, Y max 10
```

The plot is:



Demonstration of using 2D data that includes information about its uncertainty

First we need some includes to use Boost.Plot and C++ Standard Library:

```

#include <boost/svg_plot/svg_2d_plot.hpp>
using namespace boost::svg;

#include <boost/svg_plot/show_2d_settings.hpp> // Only needed for showing which settings in use.
//using boost::svg::show_2d_plot_settings;
//void boost::svg::show_2d_plot_settings(svg_2d_plot&);

#include <boost/svg_plot/detail/pair.hpp>
// using boost::svg::detail::pair; operator<<

#include <boost/quan/unc.hpp>
//#include <boost/quan/unc_init.hpp>

#include <boost/quan/meas.hpp>

//#include <algorithm>
//#include <functional>

#include <map>
using std::map;
using std::multimap;

#include <utility>
using std::pair;
using std::make_pair;

#include <vector>
using std::vector;

#include <cmath>
using std::sqrt;

#include <iostream>
using std::cout;
using std::endl;
using std::scientific;
using std::hex;
using std::ios;
using std::boolalpha;

#include <iterator>
using std::ostream_iterator;

```

STL map is used as the container for our two data series, and pairs of values and their uncertainty information (approximately standard deviation and degrees of freedom) are inserted using push_back. Since this is a 2-D plot the order of data values is important.

```

typedef unc<false> uncurl; // Uncertain Uncorrelated (the normal case).

using boost::svg::detail::operator<<;

// Check pair for double.
pair<double, double> double_pair; // double X and Y
double_pair = make_pair(double(-2.234), double(-8.76));
cout << "make_pair(double(-2.234), double(-8.76)) = " << double_pair << endl;

uncun u1(1.23, 0.56F, 7); // For an X value.
cout << "u1 = " << u1 << endl; // u1 = 1.23+-0.056 (7)
uncun u2(3.45, 0.67F, 9); // For a Y value.
pair<uncun, uncurl> mp1 = make_pair(u1, u2); // XY pair of values.
cout << mp1 << endl; // 1.23+-0.056 (7), 2.345+-0.067 (9)

map<uncun, uncurl> data1; // Container for XY pair of points.
data1.insert(mp1); // u1, u2 = 1.23+-0.056 (7), 2.345+-0.067 (9)
data1.insert(make_pair(uncun(4.1, 0.4F, 7), uncurl(3.1, 0.3F, 18))); //
data1.insert(make_pair(uncun(-2.234, 0.03F, 7), uncurl(-8.76, 0.9F, 9)));

/*
`Make very sure you don't forget either uncurl(...) like this
`data1.insert(make_pair((-2.234, 0.12F, 7),(-8.76, 0.56F, 9)));
because, by the bizarre operation of the comma operator, the result will be an integer!
So you will be astonished to find that the values will be the *pair of degrees of freedom, (7, 9)*
and the other parts of uncurl will be undefined!

```

Echo the values input:

```

*/
cout << data1.size() << " XY data pairs:" << endl;
std::copy(data1.begin(), data1.end(), ostream_iterator<pair<uncun, uncurl>>(cout, "\n"));
cout << endl;

svg_2d_plot my_plot;

```

If you can be confident that the data set(s) only contains normal, valid data, so none are 'at limits' - too big or too small to be meaningful, infinite or NaN (NotANumber), then these checks can be skipped (for speed). An instrument or operator input might be known to provide only normal data. For this example, we know this is true, so override the default autoscale_check_limits(true).

```
my_plot.autoscale_check_limits(false);
```

The default is autoscale_plusminus(3.) so that confidence ellipses at 1, 2 and 3 (uncertainty nominally standard deviations) are all within the plot window, but if you are less interested in seeing the 2 and 3 ellipses, you could risk the outer edges spilling over the borders by reducing autoscale_plusminus, for example, to 1.5, down to zero.

```
my_plot.autoscale_plusminus(1.5); // default is 3.
my_plot.confidence(0.01); // Change from default 0.05 to 0.01 for 99% confidence.
```

Use data set **data** to autoscale (you can use a different data set to scale from the one you chose to plot).

```

my_plot.xy_autoscale(data1);

my_plot
.x_label("times (sec)")
.x_range(-3, +10)
.xy_values_on(true) // Show X and Y values next to each point.

//! \note Essential use of Unicode space in all strings - ANSI space has no effect!
//.x_decor("", ",\u00A0") // Keep all on one line using separator NOT starting with a newline.
.x_decor("", "\n") // Split onto two lines because X separator does start with newline.
.y_decor("\u00A0;\u00A0;\u00A0;", "\u00A0;time =", "\u00A0;sec")
// Note: a few padding spaces are used to get Y values to lie more nearly under X values.
// This is only necessary when label are not horizontal.
.x_values_rotation(slopeup)
.x_values_font_size(16)
.x_plusminus_on(true)
.x_plusminus_color(cyan)

.x_addlimits_on(true)
.x_addlimits_color(purple)

.x_df_on(true)
.x_df_color(magenta)
.x_values_font_family("Times New Roman")

.y_label("distance (km)")
.y_range(-10., +10.)
.y_values_rotation(uphill)
.y_values_font_family("Arial") // Different from X just to show effect.
.y_plusminus_on(true)
.y_plusminus_color(red)

.y_addlimits_on(true)
.y_addlimits_color(darkgreen)

.y_df_on(true)
.y_df_color(green)

```

The default uncertainty ellipse colors (that apply to both X and Y axes) can be changed thus:

```

.one_sd_color(lightblue)
.two_sd_color(svg_color(200, 230, 255))
.three_sd_color(svg_color(230, 240, 255))
; // my_plot

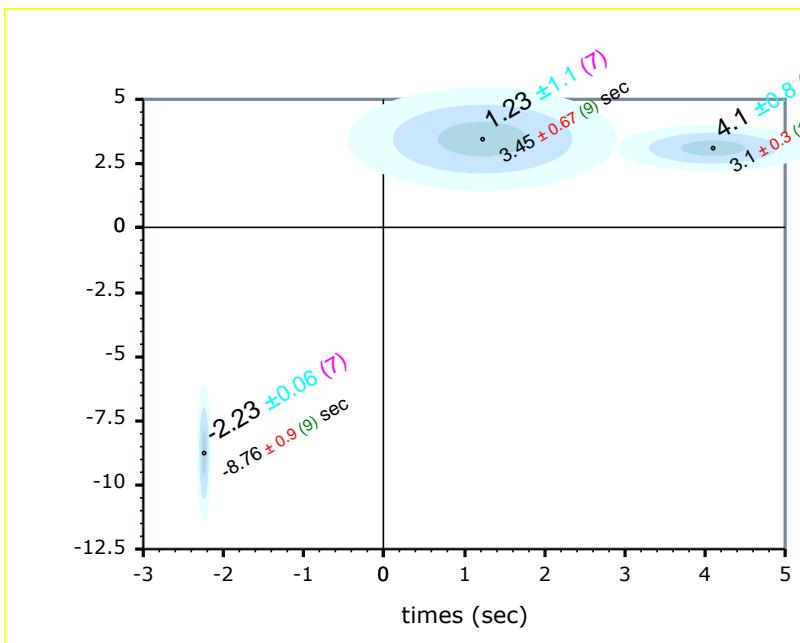
my_plot.plot(data1, "data1").shape(unc_ellipse);

my_plot.write("./demo_2d_uncertainty");

show_2d_plot_settings(my_plot);

```

The resulting plot is



See [demo_2d_uncertainty.cpp](#) for full source code and sample output.

Demonstration of adding lines and curves, typically a least squares fit

First we need some includes to use Boost.Plot and C++ Standard Library:

```
#include <boost/svg_plot/svg_2d_plot.hpp>
using namespace boost::svg;

#include <boost/svg_plot/show_2d_settings.hpp>
// using boost::svg::show_2d_plot_settings - Only needed for showing which settings in use.

#include <map>
using std::map;
using std::multimap;

#include <iostream>
using std::cout;
using std::endl;
using std::boolalpha;
```

This shows how to add lines to a plot, for example, for example a least-squares 'function' fit line. The data is roughly a straight line through the origin.

```
map<double, double> my_data;
my_data[-1.1] = -1.2;
my_data[-2.3] = -2.1;
my_data[-2.9] = -3.3;
my_data[-4.1] = -4.3;
my_data[-5.0] = -5.3;
my_data[-6.1] = -5.9;
my_data[0.] = 0.;
my_data[1.1] = 1.2;
my_data[2.3] = 2.1;
my_data[2.9] = 3.3;
my_data[4.1] = 4.3;
my_data[5.0] = 5.3;
my_data[6.1] = 5.9;
```

First construct, size and draw a simple plot ready to add some sample lines.

```
svg_2d_plot my_plot;
my_plot.size(400, 400);
my_plot.plot(my_data, "my_data").fill_color(red);
```

First draw a line using SVG coordinates (rather than the Cartesian coordinates used for user's data - see below). Note that for SVG coordinates, Y increases **down** the page, so Y = 0 is the top and Y = 300 is the bottom. Default black is provided for color.

```
my_plot.draw_line(10, 390, 390, 10);
```

This line almost reaches the corners of the SVG image, but for plotting XY graphs, you probably don't want SVG coordinates, but want to use Cartesian coordinates for user's data. Now draw a blue line using the Cartesian coordinates for user's data, from the bottom left through the origin of the plot to the top right of the plot.

```
my_plot.draw_plot_line(-10, -10, +10, +10, blue);
```

If you have calculated a confidence interval, you might want to add curved line(s) showing it (still using the Cartesian coordinates). For example, you can draw a curve (quadratic) through two X, Y pairs using a Bezier curve with the middle point as control point.

```
my_plot.draw_plot_curve(-6, -8, 0, +1, +8, +6, red);
```

Finally write the SVG image file.

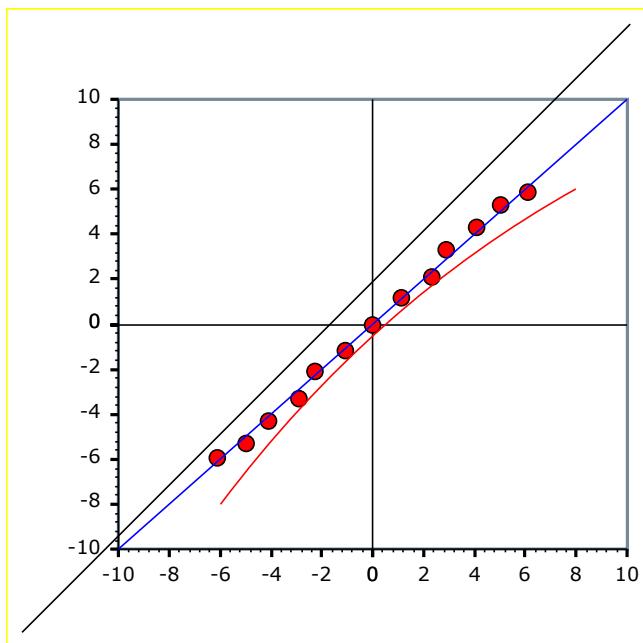


Note

At present, notes and lines must come after all plot commands to be put in the correct place.

```
my_plot.write("./demo_2d_lines");
//show_2d_plot_settings(my_plot);
```

The resulting plot is

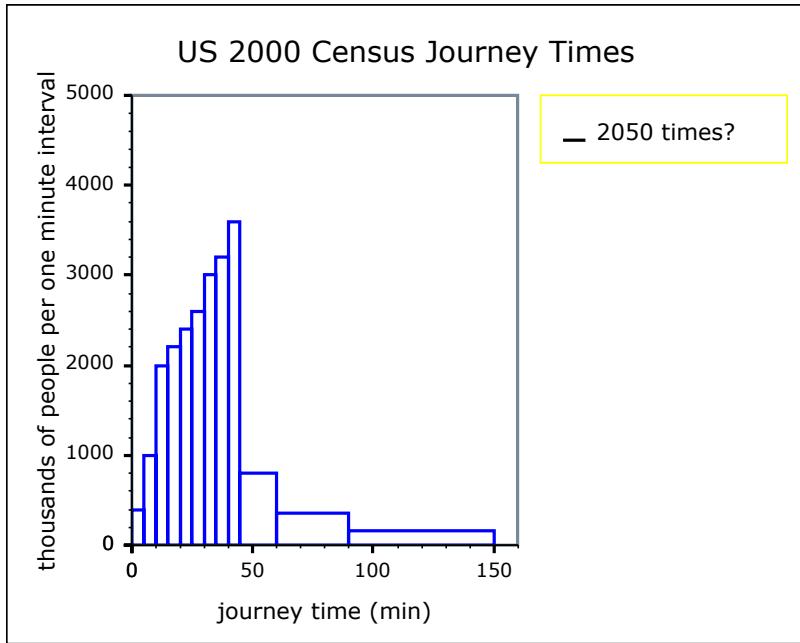
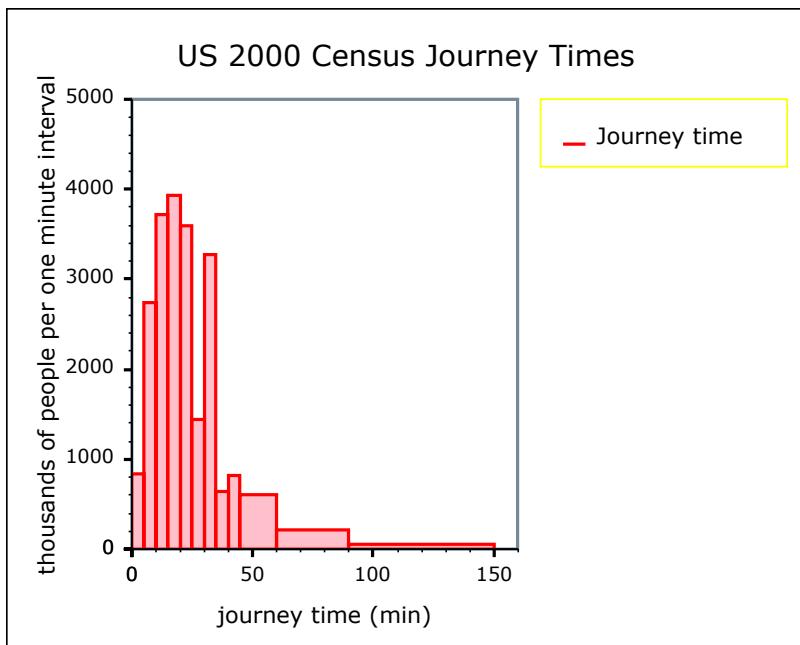


See [demo_2d_lines.cpp](#) for full source code and sample output.

Histograms of 2D data

See [demo_2d_histogram.cpp](#) for full source code and sample output.

The resulting histograms are



Tutorial: Boxplot

Simple Example

```
#include <boost/svg_plot/svg_2d_plot.hpp>
using namespace boost::svg;
#include <vector>
using std::vector;

// Functions to simulate distributions.
double f(double x)
{
    return 50 / x;
}

double g(double x)
{
    return 40 + 25 * sin(x * 50);
}

int main()
{
    std::vector<double> data1, data2;

    // Fill our vectors with values from the functions f(x) and g(x)
    for(double i = .1; i < 10; i+=.1)
    {
        data1.push_back(f(i));
        data2.push_back(g(i));
    }

    svg_boxplot my_plot; // Initialize a new box plot.

    my_plot.background_border_color(black); // Color information.

    my_plot.title("Boxplots of Common Functions") // Title and
.x_label("Functions") // labels.
.y_label("Population Size");

    my_plot.y_range(0, 100) // Axis information.
.y_minor_tick_length(20)
.y_major_interval(20);

    // Write two data series.
}
```

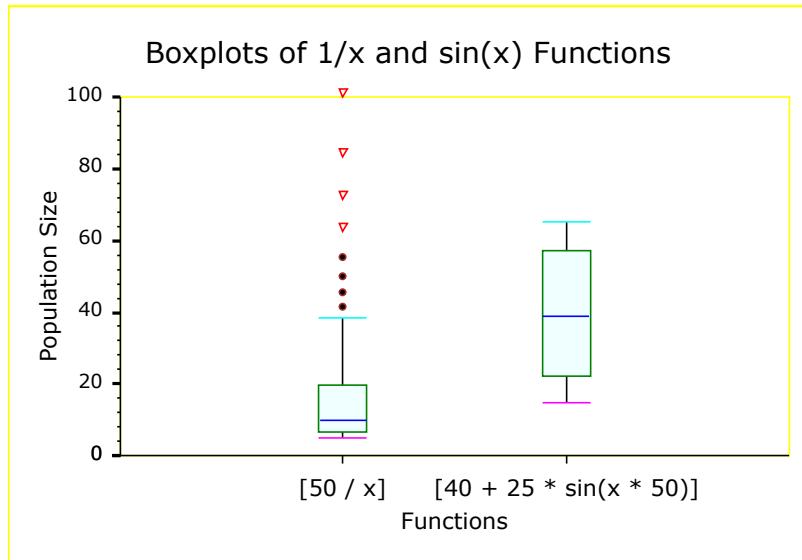
```

my_plot.plot(data1, "[50 / x]");
my_plot.plot(data2, "[40 + 25 * sin(x * 50)]");

my_plot.write("boxplot_simple.svg"); // Write final file.
return 0;
}

```

Image boxplot_simple.svg



Initializing a new boxplot

`svg_boxplot my_plot;` initializes a new boxplot. This also sets the very many default values.

Setting a color

This sets the border color of the entire image to black (not just the plot area).

```
my_plot.background_border_color(black);
```

Setting strings in the plot

The following code sets the title that appears at the top of the graph, the label that appears at the bottom to name the X-axis, and the label that appears to the left to name the Y-axis.

```

my_plot.title("Boxplots of Common Functions")
.x_label("Functions")
.y_label("Population Size");

```

Setting axis information

Axis label information for the X-axis is very limited, as the coordinate system for the X-axis is discrete, based only on the number of box-and-whisker plots that appear in the same boxplot.

Axis information for the Y-axis, however, is customizable much in the same way that that axis information is customizable for the 1 and 2 dimensional graphs.

```
// Axis information.
my_plot.y_range(0, 100)
    .y_minor_tick_length(20)
    .y_major_interval(20);
```

Writing to a file

This finally writes our plot to the file "boxplot_simple.svg".

```
my_plot.write("boxplot_simple.svg");
```

Definitions of the Quartiles

This example demonstrates the dramatic effect of the choice of definition of the quartiles.

"Some Implementations of the Boxplot" Michael Frigge, David C. Hoaglin and Boris Iglewicz The American Statistician, Vol. 43, No. 1 (Feb., 1989), pp. 50-54 discusses the design of the boxplot.

However the plot of their example data shown below shows the considerable variation in the appearance of the same data, using different definitions of quartiles used in various popular statistics packages.

One obvious conclusion is that you should not expect boxplots to look the same when using more than one program.

Boost.Plot provides 5 popular definitions for the quartiles. This should allow the user to produce plots that look similar to boxplots from most statistics plotting program. To confuse matter further, most have their own default definition **and** options to chose other definitions: these options are shown below as type, method, PCTLDEF.

The interquartile range is calculated using the 1st & 3rd sample quartiles, but there are various ways to calculate those quartiles, summarised in Rob J. Hyndman and Yenan Fan, 1996, "Sample Quantiles in Statistical Packages", The American Statistician 50(4):361-365, (1996).

The interquartile range, often called IQR is quartile 3 ($p = 3/4$) - quartile 1 ($1/4$). The median is the 2nd quartile ($p = 2/4 = 1/2$).

Five of Hyndman and Fan's sample quantile definitions have a particularly simple common form selected according to which definition of m is chosen in function quantiles. This is implemented in function quantiles by parameter HF_definition:

```
double quantile(vector<double>& data, double p, int HF_definition = 8);
```

The default definition is that recommended by Hyndman and Fan, or users can select which definition is used for all boxplots, or individual data series as shown in the example below.

```
my_boxplot.quartile_definition(5); // All plots
my_boxplot.plot.quartile_definition(7); // Just this data series plot.
```

Hyndman and Fan definitions 4 to 8 are used by the following packages:

- #4 SAS (PCTLDEF=1), R (type=4), Maple (method=3)
- #5 R (type=5), Maple (method=4), Wolfram Mathematica quartiles.
- #6 Minitab, SPSS, BMDP, JMP, SAS (PCTLDEF=4), R(type=6), Maple (method=5).
- #7 Excel, S-Plus, R (type=7[default]), Maxima, Maple (method=6).
- #8 H&F 8: R (type=8), Maple (method=7[default]).

Some observations on the various options are:

- #4 Often a moderate interquartile range.
- #5 Symmetric linear interpolation: a common choice when the data represent a sample from a continuous distribution and you want an unbiased estimate of the quartiles of that distribution.
- #6 This "half" sample excludes the sample median (k observations) for odd n ($=2*k+1$). This will tend to be a better estimate for the population quartiles, but will tend to give quartile estimates that are a bit too far from the center of the whole sample (too wide an interquartile range).
- #7 Smallest interquartile range, so flags most outliers. For a continuous distribution, this will tend to give too narrow an interquartile range, since there will tend to be a small fraction of the population beyond the extreme sample observations. In particular, for odd n ($=2*k+1$), Excel calculates the 1st (3rd) quartile as the median of the lower (upper) "half" of the sample including the sample median ($k+1$ observations).
- #8 recommended by H&F because it is approximately median-unbiased estimate regardless of distribution and thus suitable for continuous and discrete distributions. which gives quartiles between those reported by Minitab and Excel. This approach is approximately median unbiased for continuous distributions. Slightly higher interquartile range than definition 7.

The 'fences' beyond which points are regarded as outliers, or extreme outliers, are a multiplying factor, usually called k , and usually $1.5 * \text{interquartile range}$, and $3 * \text{interquartile range}$ as recommended by Hoaglin et al.

```
#include <vector>
using std::vector;
#include <cmath>
using ::sin;
//#include <boost/assert.hpp> // for BOOST_ASSERT
#include <boost/svg_plot/svg_boxplot.hpp>

#include <boost/svg_plot/quantile.hpp>
using boost::svg::quantile;

// double boost::svg::quantile(vector<double>& data, double p, int HF_definition);
// Estimate pth quantile of data using one of 5 definitions.
// Default HF_definition is the recommendation of Hyndman and Fan, definition #8.

#include <boost/array.hpp>
using boost::array;

#include <iostream>
using std::cout;
using std::endl;
```

```
// 11 values from Hoaglin et al page 50.
const boost::array<double, 11> Hoaglin_data = {53., 56., 75., 81., 82., 85., 87., 89., 95., 99., 100.};
//                                     q1      median      q3

vector<double> Hoaglin(Hoaglin_data.begin(), Hoaglin_data.end());
for (int def = 4; def <= 8; def++)
{ // All the F&Y definitions of quartiles.
    double q1 = quantile(Hoaglin, 0.25, def); // 75
    double q2 = quantile(Hoaglin, 0.5, def); // 85
    double q3 = quantile(Hoaglin, 0.75, def); // 95
    cout << "Hoaglin definition #" << def << ", q1 " << q1
        << ", q2 " << q2 << ", q3 " << q3 << ", IQR " << q3 - q1 << endl;
} // for

// Same data copied for different data series.
vector<double> Hoaglin4(Hoaglin_data.begin(), Hoaglin_data.end());
vector<double> Hoaglin5(Hoaglin_data.begin(), Hoaglin_data.end());
vector<double> Hoaglin6(Hoaglin_data.begin(), Hoaglin_data.end());
vector<double> Hoaglin7(Hoaglin_data.begin(), Hoaglin_data.end());
vector<double> Hoaglin8(Hoaglin_data.begin(), Hoaglin_data.end());

svg_boxplot H_boxplot;
```

Show the quartile definition default.

```
cout << "Default boxplot.quartile_definition() = " << H_boxplot.quartile_definition() << endl; // 8
```

Add title, labels, range etc to the whole boxplot:

```
H_boxplot // Title and axes labels.
.title("Hoaglin Example Data")
.x_label("Boxplot")
.y_label("Value")
.y_range(45, 115) // Y-Axis range.
.y_minor_tick_length(2)
.y_major_interval(10);
```

Add a few setting to the plot including setting quartile definition (though is actually same as the default 8), and show that the value is stored.

```
svg_boxplot& b = H_boxplot.median_values_on(true)
.outlier_values_on(true)
.extreme_outlier_values_on(true)
.quartile_definition(8);
```

Show the quartile definition just assigned:

```
cout << "boxplot.quartile_definition() = " << b.quartile_definition() << endl; // 8
```

Add a data series container, and labels, to the plot using the whole boxplot quartile definition set.

```
H_boxplot.plot(Hoaglin_data, "default_8");
```

Add another data series container, and the labels, to the plot, and select a **different** quartile definition.

```
svg_boxplot_series& d4 =
H_boxplot.plot(Hoaglin4, "def #4")
.whisker_length(4.)
.quartile_definition(4.);
```

Show the quartile definition just assigned to the this data series.

```
cout << "boxplot_series.quartile_definition() = " << d4.quartile_definition() << endl; // 4
```

Add yet more data series container, and the labels, to the plot, and select a **different** quartile definition for each.

```
H_boxplot.plot(Hoaglin5, "def #5")
.whisker_length(5.)
.quartile_definition(5.);

H_boxplot.plot(Hoaglin6, "def #6")
.whisker_length(6.)
.quartile_definition(6.);

H_boxplot.plot(Hoaglin6, "def #7")
.whisker_length(7.)
.quartile_definition(7.);

H_boxplot.plot(Hoaglin6, "def #8")
.whisker_length(8.)
.quartile_definition(8.);
```

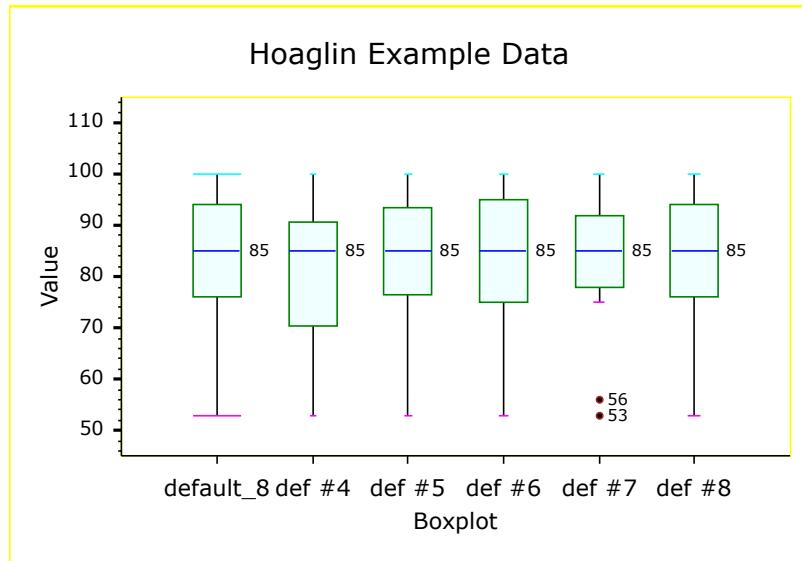
Write the entire SVG plot to a file.

```
H_boxplot.write("demo_Hoaglin.svg");
```

Typical output is:

```
Autorun "j:\Cpp\SVG\debug\demo_Hoaglin.exe"
Hoaglin definition #4, q1 70.25, q2 83.5, q3 90.5, IQR 20.25
Hoaglin definition #5, q1 76.5, q2 85, q3 93.5, IQR 17
Hoaglin definition #6, q1 75, q2 85, q3 95, IQR 20
Hoaglin definition #7, q1 78, q2 85, q3 92, IQR 14
Hoaglin definition #8, q1 76, q2 85, q3 94, IQR 18
Default boxplot.quartile_definition() = 8
boxplot.quartile_definition() = 8
boxplot_series.quartile_definition() = 4
```

Plot showing the appearance for Hoaglin's eight definitions is at:



See [demo_Hoaglin.cpp](#) for full source code and sample output.

SVG tutorial

The SVG interface is only documented in the reference section.

Most users will want to use the plot interfaces.

A few very rudimentary examples of use (mainly historically used to testbed various features used in the plot interfaces) can be seen at

[demo_svg.cpp](#)

Showing data values at Numerical Limits

All limits that are dealt with are double precision floating-point limits. Data values that are 'at limits' are detected when data is added to the plot by the call to the `plot()` method.

Currently, the line interpolation algorithms do not take limits into account, so behavior may be incorrect for such plots as $1 / x$.

Data at the limits are drawn, by default, as a cone with `stroke_color` is `lightgray` and `fill_color` is `whitesmoke`.

This is customizable using, for example:

```
my_plot.limit_color(red).limit_fill_color(green);
```

showing an inverted down-pointing cone with a red outline and a green fill.

NaN

Any double precision floating-point numbers that return a nonzero value for the function `isnan(double)` is considered to be a *NaN* or *NotANumber* value. When plotted, the number will appear in the user-defined coordinates as 0 or 0,0.

Infinity

Any double precision floating point number that is equal to either of the following is considered to be *infinity*:

```
std::numeric_limits<double>::max()
std::numeric_limits<double>::infinity()
```

When plotted, these values will appear at the **top** of your plot window. If the plot window is not turned on, these points will appear at the top of the image.

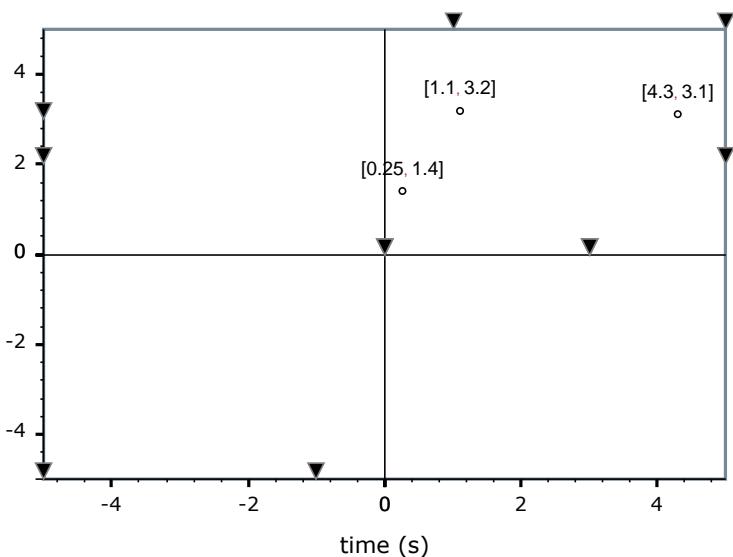
Negative Infinity

Any double precision floating point number that is equal to either of the following is considered to be *negative infinity*:

```
std::numeric_limits<double>::min()
-std::numeric_limits<double>::infinity()
std::numeric_limits<double>::denorm_min()
```

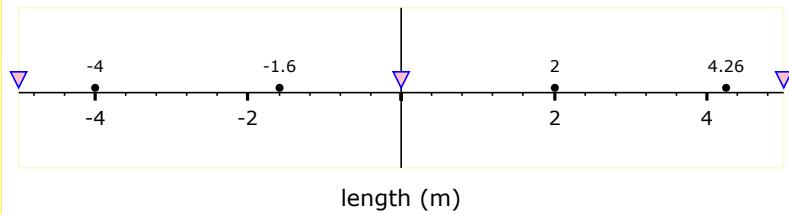
When plotted, these values will appear at the **bottom** of your plot window. If the plot window is not turned on, these points will appear at the bottom of the image.

Default 2D 'at limits' NaN and infinities Demo



Here are examples of showing values at numeric limits:

Default 1D NaN and infinities Demo



See also examples of limits in 1-D and 2-D plots. [demo_1d_limits.cpp](#) [demo_2d_limits.cpp](#)

Boost.SVG plot C++ Reference

Header <boost/svg_plot/detail/auto_axes.hpp>

Scalable Vector Graphic (SVG) autoscaling of axes.

Inspect container or data values to find minimum and maximum, avoiding values that are NaN and/or 'at limit'. Scale axis using max and min values (calculated or user provided), optionally to include the origin, and to set the ticks. Provide fine control over any overlap at the edges of the axes to avoid a tiny amount over the limit resulting in an ugly extra major tick. Also allow optional forcing of the ticks to be multiples of 1, 2, 5, 10.

```
namespace boost {
    namespace math {
        template<typename FPT> bool isfinite(FPT);
    }
    namespace svg {
        template<typename Iter> int mnmx(Iter, Iter, double *, double *);
        template<typename T> std::pair<double, double> range_all(const T &);
        template<typename T> std::pair<double, double> range_mx(const T &);
        double rounddown10(double);
        double rounddown2(double);
        double rounddown5(double);
        double roundup10(double);
        double roundup2(double);
        double roundup5(double);
        void scale_axis(double, double, double *, double *, double *, int *,
                      bool = false, double = 0., int = 6, int = 0);
        void scale_axis(double, double, double *, double *, double *, int *,
                      bool = true, double = 2., bool = false, double = 0.,
                      int = 6, int = 0);
        template<typename Iter>
        void scale_axis(Iter, Iter, double *, double *, double *, int *, bool,
                      double, bool = false, double = 0., int = 6, int = 0);
        template<typename C>
        void scale_axis(const C &, double *, double *, double *, int *, bool,
                      double = 3., bool = false, double = 0., int = 6,
                      int = 0);
        template<typename C>
        void scale_axis(const C &, double *, double *, double *, int *,
                      double *, double *, double *, int *, bool = true,
                      double = 3., bool = false, double = 0., int = 6,
                      int = 0, bool = false, double = 0., int = 6, int = 0);
        template<typename T> size_t show(const T &);
        template<typename iter> size_t show(iter, iter);
        template<typename T> size_t show_all(const T &);
    }
}
```

Function template isfinite

boost::math::isfinite — If a floating-point value is finite, return true.

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>

template<typename FPT> bool isfinite(FPT t);
```

Description

Parameters: `t` floating-point value to test if finite.
 Template Parameters: `FPT` floating-point type (float, double, long double, or user-defined).
 Returns: true if is finite, false if + or - infinite, or any NaN.

Function template mnmx

`boost::svg::mnmx` — Inspect values to find min and max.

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>

template<typename Iter>
int mnmx(Iter begin, Iter end, double * min, double * max);
```

Description

Inspect all values between begin and (one before) end to work out and update min and max. Similar to `boost::minmax_element`, but ignoring at 'limit': non-finite, +-infinity, max & min, & NaN). If can't find a max and a min, then throw a `runtime_error` exception.

Parameters: `begin` Iterator to chosen first item in container.
`end` Iterator to chosen last item in container.
`max` Updated with Maximum value found (not 'at limit').
`min` Updated with Minimum value found (not 'at limit').
 Returns: number of normal values (not 'at limit' neither too big, NaN nor infinite).

Function template range_all

`boost::svg::range_all`

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>

template<typename T>
std::pair<double, double> range_all(const T & containers);
```

Description

Returns: minimum and maximum of a container containing STL containers.

Function template range_mx

boost::svg::range_mx

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>

template<typename T> std::pair< double, double > range_mx(const T & container);
```

Description

Calculate minimum and maximum from data in a container.

Parameters: container Container Data series.
 Template Parameters: T an STL container: array, vector, set, map ...
 Returns: minimum and maximum of an STL container as a std::pair.

Function rounddown10

boost::svg::rounddown10

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>

double rounddown10(double value);
```

Description

Round down to nearest multiple of 10. Decimal scaling steps, so value is 0.1, 0.2, 0.5, 1., 2., 5. or 1., 10., 20., 100. ...

Returns: Rounded down value.

Function rounddown2

boost::svg::rounddown2

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>

double rounddown2(double value);
```

Description

Binary scaling steps, so return 0.1, 0.2, 0.4, 0.6, 0.8, 1.0, 2, 4, 6, 8, 10, 20, 40 60, 80, 100..

Returns: Rounded down value.

Function rounddown5

boost::svg::rounddown5

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>

double rounddown5(double value);
```

Description

Semi-decimal scaling, so return 0.1, 0.5, 1, 5, 10, 50, 100 ...

Returns: Rounded down value.

Function roundup10

boost::svg::roundup10

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>

double roundup10(double value);
```

Description

Round up to nearest multiple of 10. Decimal scaling steps, so value is 0.1, 0.2, 0.5, 1., 2., 5. or 1., 10., 20., 100. ...

Returns: Rounded up value.

Function roundup2

boost::svg::roundup2

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>

double roundup2(double value);
```

Description

Binary scaling steps, so return 0.1, 0.2, 0.4, 0.6, 0.8, 1.0, 2, 4, 6, 8, 10, 20, 40 60, 80, 100..

Returns: Rounded up value.

Function roundup5

boost::svg::roundup5

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>

double roundup5(double value);
```

Description

Semi-decimal scaling, so return 0.1, 0.5, 1, 5, 10, 50, 100 ...

Returns: rounded up value.

Function scale_axis

boost::svg::scale_axis

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>

void scale_axis(double min_value, double max_value, double * axis_min_value,
                double * axis_max_value, double * axis_tick_increment,
                int * auto_ticks, bool origin = false, double tight = 0.,
                int min_ticks = 6, int steps = 0);
```

Description

Scale axis and update min and max axis values, and tick increment and number of ticks.

Parameters:	auto_ticks	Computed number of ticks, updated by scale_axis.
	axis_max_value	Computed maximum value for the axis, updated by scale_axis.
	axis_min_value	Computed minimum value for the axis, updated by scale_axis.
	axis_tick_increment	Computed tick increment for the axis, updated by scale_axis.
	max_value	Scale axis from explicit input range maximum.
	min_ticks	Minimum number of major ticks.
	min_value	Scale axis from explicit input range minimum.
	origin	If false, do not include the origin unless the range min_value to max_value includes zero.
	steps	0, or 2 for 2, 4, 6, 8, 10, 5 for 1, 5, 10, or 10 (2, 5, 10).
	tight	Fraction of overrun allowed before another tick used. For a good visual effect, up to about 0.001 might suit a 1000 pixel wide image, allowing values just 1 pixel over the tick to be shown.

Function scale_axis

boost::svg::scale_axis

Synopsis

// In header: <boost/svg_plot/detail/auto_axes.hpp>

```
void scale_axis(double min_value, double max_value, double * axis_min_value,
               double * axis_max_value, double * axis_tick_increment,
               int * auto_ticks, bool check_limits = true,
               double autoscale_plusminus = 2., bool origin = false,
               double tight = 0., int min_ticks = 6, int steps = 0);
```

Description

Scale axis function to define axis marker ticks based on min & max parameters values (handling uncertainty).

Parameters:	auto_ticks	Computed number of ticks, updated by scale_axis.
	autoscale_plusminus	Multiplier of uncertainty or standard deviations to allow for confidence ellipses.
	axis_max_value	Computed maximum value for the axis, updated by scale_axis.
	axis_min_value	Computed minimum value for the axis, updated by scale_axis.
	axis_tick_increment	Computed tick increment for the axis, updated by scale_axis.
	check_limits	If true then check all values for infinity, NaN etc.
	max_value	Scale axis from explicit input maximum.
	min_ticks	Minimum number of major ticks.
	min_value	Scale axis from explicit input minimum.
	origin	If true, ensures that zero is a tick value.
	steps	Round up and down to 2, 4, 6, 8, 10, or 5, 10 or 2, 5, 10 systems.
	tight	Allows user to avoid a small fraction over a tick using another tick.

Function template scale_axis

boost::svg::scale_axis

Synopsis

// In header: <boost/svg_plot/detail/auto_axes.hpp>

```
template<typename Iter>
void scale_axis(Iter begin, Iter end, double * axis_min_value,
                double * axis_max_value, double * axis_tick_increment,
                int * auto_ticks, bool check_limits,
                double autoscale_plusminus, bool origin = false,
                double tight = 0., int min_ticks = 6, int steps = 0);
```

Description

Scale axis from data series (usually to plot), perhaps only part of container.

Parameters:	auto_ticks	Computed number of ticks, updated by scale_axis.
	autoscale_plusminus	Multiplier of uncertainty or standard deviations to allow for confidence ellipses.
	axis_max_value	Computed maximum value for the axis, updated by scale_axis.
	axis_min_value	Computed minimum value for the axis, updated by scale_axis.
	axis_tick_increment	Computed tick increment for the axis, updated by scale_axis.
	begin	First item in container to use to calculate autoscale minimum or maximum.

check_limits	Whether to check all values for infinity, NaN etc.
end	Last item in container to use to calculate autoscale minimum or maximum.
min_ticks	Minimum number of major ticks.
origin	If false, do not include the origin unless the range $\text{min_value} \leq 0 \leq \text{max_value}$.
steps	0, or 2 for 2, 4, 6, 8, 10, 5 for 1, 5, 10, or 10 (2, 5, 10).
tight	fraction of 'overrun' allowed before another tick used. For visual effect up to about 0.001 might suit a 1000 pixel wide image, allowing values just 1 pixel over the tick to be shown.

Template Parameters: Iter Type of iterator into STL container type: array, vector ...

Function template scale_axis

boost::svg::scale_axis — Scale axis using an **entire** Container of a Data series, usually to plot (not necessarily ordered, so will find minimum and maximum).

Synopsis

// In header: <boost/svg_plot/detail/auto_axes.hpp>

```
template<typename C>
void scale_axis(const C & container, double * axis_min_value,
                double * axis_max_value, double * axis_tick_increment,
                int * auto_ticks, bool check_limits,
                double autoscale_plusminus = 3., bool origin = false,
                double tight = 0., int min_ticks = 6, int steps = 0);
```

Description

Parameters:	auto_ticks	Computed number of ticks, updated by scale_axis.
	autoscale_plusminus	Multiplier of uncertainty or standard deviations to allow for confidence ellipses.
	axis_max_value	Computed maximum value for the axis, updated by scale_axis.
	axis_min_value	Computed minimum value for the axis, updated by scale_axis.
	axis_tick_increment	Computed tick increment for the axis, updated by scale_axis.
	check_limits	Whether to check all values for infinity, NaN etc.
	container	STL container, usually of a data series.
	min_ticks	Minimum number of major ticks.
	origin	If false, do not include the origin unless the range $\text{min_value} \leq 0 \leq \text{max_value}$.
	steps	0, or 2 for 2, 4, 6, 8, 10, 5 for 1, 5, 10, or 10 (2, 5, 10).
	tight	fraction of overrun allowed before another tick used. For visual effect up to about 0.001 might suit a 1000 pixel wide image, allowing values just 1 pixel over the tick to be shown.

Template Parameters: C STL container type: array, vector ...

Function template scale_axis

boost::svg::scale_axis

Synopsis

// In header: <boost/svg_plot/detail/auto_axes.hpp>

```
template<typename C>
void scale_axis(const C & container, double * x_axis_min_value,
                double * x_axis_max_value, double * x_axis_tick_increment,
                int * x_auto_ticks, double * y_axis_min_value,
                double * y_axis_max_value, double * y_axis_tick_increment,
                int * y_auto_ticks, bool check_limits = true,
                double autoscale_plusminus = 3., bool x_origin = false,
                double x_tight = 0., int x_min_ticks = 6, int x_steps = 0,
                bool y_origin = false, double y_tight = 0.,
                int y_min_ticks = 6, int y_steps = 0);
```

Description

Scale X and Y axis using a 2D STL container: std::array of std::pairs, std::vector of std::pairs, ...

Parameters:	autoscale_plusminus	Multiplier of uncertainty or standard deviations to allow for confidence ellipses.
	check_limits	Whether to check all values for infinity, NaN etc.
	container	Data series to plot - entire 2D container.
	x_auto_ticks	Computed number of ticks, updated by scale_axis.
	x_axis_max_value	Computed minimum value for the X-axis, updated by scale_axis.
	x_axis_min_value	Computed maximum value for the X-axis, updated by scale_axis.
	x_axis_tick_increment	Computed tick increment for the axis, updated by scale_axis.
	x_min_ticks	Minimum number of X-axis major ticks.
	x_origin	Do not include the origin unless the range min_value <= 0 <= max_value.
	x_steps	0, or 2 for 2, 4, 6, 8, 10, 5 for 1, 5, 10, or 10 (2, 5, 10).
	x_tight	Fraction of 'overrun' allowed before another tick used. For visual effect up to about 0.001 might suit a 1000 pixel wide image, allowing values just 1 pixel over the tick to be shown.
	y_auto_ticks	Computed number of Y-axis ticks, updated by scale_axis.
	y_axis_max_value	Computed maximum value for the Y-axis, updated by scale_axis.
	y_axis_min_value	Computed minimum value for the Y-axis, updated by scale_axis.
	y_axis_tick_increment	Updated with Y-axis tick increment.
	y_min_ticks	Minimum number of Y axis major ticks.
	y_origin	Do not include the origin unless the range min_value to max_value contains zero.
	y_steps	0, or 2 for 2, 4, 6, 8, 10, 5 for 1, 5, 10, or 10 (2, 5, 10).
	y_tight	Fraction of 'overrun' allowed before another tick used. For visual effect up to about 0.001 might suit a 1000 pixel wide image, allowing values just 1 pixel over the tick to be shown.

Template Parameters:

C STL container holding 2D pairs of X and Y.

Function template show

boost::svg::show

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>
```

```
template<typename T> size_t show(const T & container);
```

Description

Utility functions to display STL containers.

Function template show

boost::svg::show

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>
```

```
template<typename iter> size_t show(iter begin, iter end);
```

Description

Utility function to display STL containers.

Function template show_all

boost::svg::show_all

Synopsis

```
// In header: <boost/svg_plot/detail/auto_axes.hpp>
```

```
template<typename T> size_t show_all(const T & containers);
```

Description

Show all the containers values.

Header <boost/svg_plot/detail/axis_plot_frame.hpp>

SVG Plot functions common to 1D, 2D and Boxplots.

Functions are derived from base class `axis_plot_frame`.

Jacob Voytko and Paul A. Bristow

```

namespace boost {
namespace svg {

// Placing of legend box, if requested by legend_on == true.
enum legend_places { nowhere == 0, inside == -1, outside_left == 1,
                     outside_right == +2, outside_top == +3,
                     outside_bottom == +4, somewhere == +5 };

// If and how the X axes intersects Y axis.
enum x_axis_intersect { bottom == -1, x_intersects_y == 0, top == +1 };

// If and how the Y axes intersects X axis.
enum y_axis_intersect { left == -1, y_intersects_x == 0, right == +1 };

static const double reducer; // To make uncertainty and degrees of freedom estimates a bit smaller font to help distinguish from value.
static const double sin45; // Used to calculate 'length' if axis value labels are sloping.
static const double text_plusminus;
}
}

```

Global reducer

boost::svg::reducer — To make uncertainty and degrees of freedom estimates a bit smaller font to help distinguish from value.

Synopsis

```

// In header: <boost/svg_plot/detail/axis_plot_frame.hpp>

static const double reducer;

```

Global sin45

boost::svg::sin45 — Used to calculate 'length' if axis value labels are sloping.

Synopsis

```

// In header: <boost/svg_plot/detail/axis_plot_frame.hpp>

static const double sin45;

```

Global text_plusminus

boost::svg::text_plusminus

Synopsis

```

// In header: <boost/svg_plot/detail/axis_plot_frame.hpp>

static const double text_plusminus;

```

Description

Number of standard deviations used for text_plusminus text display.

Nominal factor of 2 (strictly 1.96) corresponds to 95% confidence limit.

Header <[boost/svg_plot/detail/fp_compare.hpp](#)>

Class for comparing floating point values to see if nearly equal.

Two types of comparison are provided: FPC_STRONG, "Very close" - Knuth equation 1', the default. FPC_WEAK "Close enough" - equation 2'. equations in Dougles E. Knuth, Seminumerical algorithms (3rd Ed) section 4.2.4, Vol II, pp 213-225, Addison-Wesley, 1997, ISBN: 0201896842.

Strong requires closeness relative to BOTH values being compared, Weak only requires only closeness to EITHER ONE value.

This permits one to avoid some of the problems that can arise from comparing floating-point values by circumnavigating the assumption that floating point operations always give exactly the same result.

See <http://hal.archives-ouvertes.fr/docs/00/28/14/29/PDF/floating-point-article.pdf> for more about the pitfalls.

Paul A. Bristow

Aug 2009

```
template<typename FPT = double> class close_to;
template<typename FPT = double> class smallest;

typedef close_to< double > neareq; // A shorthand for twice std::numeric_limits<double>::epsilon(), often 2e-16.
typedef smallest< double > tiny; // A shorthand for twice std::numeric_limits<double>::min_value(), often 4.4e-308.

// std::numeric_limits<>::epsilon() or similar.
template<typename T> T epsilon(T);
template<typename FPT> FPT fpt_abs(FPT);

// std::numeric_limits<>::max() or similar.
template<typename T> T max_value(T);

// std::numeric_limits<>::min() or similar.
template<typename T> T min_value(T);
template<typename FPT> FPT safe_fpt_division(FPT, FPT);
```

Class template `close_to`

`close_to` — Check two floating-point values are close within a chosen tolerance.

Synopsis

```
// In header: <boost/svg_plot/detail/fp_compare.hpp>

template<typename FPT = double>
class close_to {
public:
    // construct/copy/destruct
    template<typename T>
    explicit close_to(T, floating_point_comparison_type = FPC_STRONG);
    close_to();

    // public member functions
    bool operator()(FPT, FPT) const;
    FPT size();
    floating_point_comparison_type strength();
};
```

Description

close_to public construct/copy/destruct

1. `template<typename T>`
`explicit close_to(T tolerance,`
`floating_point_comparison_type fpc_type = FPC_STRONG);`

Constructor for fraction tolerance and strength of comparison. Checks that tolerance isn't negative - which does not make sense, and can be assumed to be a programmer error?

2. `close_to();`

Default constructor is strong comparison to twice numeric_limits<double>::epsilon().

close_to public member functions

1. `bool operator()(FPT left, FPT right) const;`

Compare two floating point values

Returns: true if they are effectively equal (approximately) within tolerance & comparison strength.

2. `FPT size();`

Returns: fraction tolerance.

3. `floating_point_comparison_type strength();`

Returns: floating_point comparison type strength, FPC_STRONG or FPC_WEAK

Class template smallest

smallest — Check floating-point value is smaller than a chosen small value.

Synopsis

```
// In header: <boost/svg_plot/detail/fp_compare.hpp>

template<typename FPT = double>
class smallest {
public:
    // construct/copy/destruct
    template<typename T> explicit smallest(T);
    smallest();

    // public member functions
    template<typename T> bool operator()(T, T);
    template<typename T> bool operator()(T);
    FPT size();
};
```

Description

smallest public construct/copy/destruct

1. `template<typename T> explicit smallest(T s);`

<

David Monniaux, <http://arxiv.org/abs/cs/0701192v4>, It is somewhat common for beginners to add a comparison check to 0 before computing a division, in order to avoid possible division-by-zero exceptions or the generation of infinite results. A first objection to this practise is that, anyway, computing $1/x$ for x very close to zero will generate very large numbers that will most probably result in overflows later. Another objection, which few programmers know about and that we wish to draw attention to, is that it may actually fail to work, depending on what the compiler does - that is, the program may actually test that $x \neq 0$, then, further down, find that $x = 0$ without any apparent change to x !

2. `smallest();`

< Default Constructor. Default `smallest_ = 2. * boost::math::tools::min_value<double>()`; multiplier `m = 2` (must be integer or `static_cast<FPT>()`) is chosen to allow for a few bits of computation error. Pessimistic multiplier is the number of arithmetic operations, assuming every operation causes a 1 least significant bit error, but a more realistic average would be half this.

smallest public member functions

1. `template<typename T> bool operator()(T fp_value, T s);`

<

Returns: true if smaller than the given small value.

2. `template<typename T> bool operator()(T fp_value);`

<

Returns: true if smaller than the defined smallest effectively-zero value.

3. `FPT size();`

<

Returns: smallest value that will be counted as effectively zero.

Function template fpt_abs

fpt_abs

Synopsis

```
// In header: <boost/svg_plot/detail/fp_compare.hpp>

template<typename FPT> FPT fpt_abs(FPT arg);
```

Description

abs function (just in case abs is not defined for FPT).

Function template safe_fpt_division

safe_fpt_division

Synopsis

```
// In header: <boost/svg_plot/detail/fp_compare.hpp>

template<typename FPT> FPT safe_fpt_division(FPT f1, FPT f2);
```

Description

Safe from under and overflow. Both f1 and f2 must be unsigned here.

Header <[boost/svg_plot/detail/functors.hpp](#)>

Functors to convert data to doubles.

SVG plot assumes all data are convertible to double or uncertain value type unc before being plotted. The functors are used to convert both 1D and 2D (pairs of data values) to be converted. Note that uncertain value class unc only holds double precision and long double data will therefore lose information. This seems reasonable design decision as any real data to be plotted is unlikely to have more than double precision (about 16 decimal digits).

Jacob Voytko and Paul A. Bristow

Mar 2009

Header <[boost/svg_plot/detail/numeric_limits_handling.hpp](#)>

Functions to check if data values are NaN or infinity or denormalised.

Since only double is used, template versions are not needed, and TR1 should provide max, min, denorm_min, infinity and isnan, but older compilers and libraries may not provide all these.

Jacob Voytko and Paul A. Bristow

Header <boost/svg_plot/detail/pair.hpp>

Provides a private implementation of operator<< for std::pair that outputs pairs with a comma-separated format, for example: "1.2, 3.4".

Hidden in namespace detail to avoid clashes with other potential implementations of std::pair operator<<.

Paul A. Bristow

Header <boost/svg_plot/detail/svg_boxplot_detail.hpp>

Boost.Plot SVG Box plot Implementation details.

See `svg_boxplot.hpp` for user functions. See also `svg_style_detail.hpp` for enum `plot_doc_structure`. Caution: these two enum and ids must match because the enum value is used to index the array of id strings. `void set_ids()` copies all strings to matching image.get_g_element(i).id() So add any new id items to both!

Jacob Voytko and Paul A. Bristow

```
namespace boost {
    namespace svg {
        namespace boxplot {

            // groups that form the boxplot svg document structure. Order controls the painting order, later ones overwriting earlier layers.
            enum boxplot_doc_structure { PLOT_BACKGROUND == 0,
                PLOT_WINDOW_BACKGROUND, X_AXIS, Y_AXIS,
                X_TICKS, Y_MAJOR_TICKS, Y_MINOR_TICKS,
                Y_MAJOR_GRID, Y_MINOR_GRID, VALUE_LABELS,
                Y_LABEL, X_LABEL, BOX_AXIS, BOX, MEDIAN,
                WHISKER, MILD_OUTLIERS, EXTREME_OUTLIERS,
                DATA_VALUE_LABELS, PLOT_TITLE, PLOT_NOTES,
                BOXPLOT_DOC_CHILDREN };

            std::string document_ids_; // String descriptors used in SVG XML (matching enum boxplot_doc_structure).
        }
    }
}
```

Global document_ids_

`boost::svg::boxplot::document_ids_` — String descriptors used in SVG XML (matching enum `boxplot_doc_structure`).

Synopsis

```
// In header: <boost/svg_plot/detail/svg_boxplot_detail.hpp>
std::string document_ids_;
```

Header <boost/svg_plot/detail/svg_style_detail.hpp>

Plot document structure whose order controls the painting order, later layers overwriting earlier layers.

Header <boost/svg_plot/detail/svg_tag.hpp>

Boost.Plot SVG plot Implementation details.

See `svg.hpp` etc for user functions. `svg_tag.hpp` defines all classes that can occur in the SVG parse tree.

Jacob Voytko and Paul A. Bristow

```

namespace boost {
namespace svg {

struct a_path;
struct c_path;

class circle_element;
class clip_path_element;
class ellipse_element;
class g_element;

struct h_path;
struct l_path;

class line_element;
struct m_path;

class path_element;
struct path_point;
struct poly_path_point;
struct polygon_element;

class polyline_element;
struct q_path;

class curve_element;
class rect_element;

struct s_path;

class svg_element;
struct t_path;

class text_element;
class text_element_text;
class text_parent;
class tspan_element;

struct v_path;
struct z_path;

// Represents a single block of text, with font & alignment.
enum align_style { left_align, right_align, center_align };
bool operator!=(const rect_element &, const rect_element &);
std::ostream & operator<<(std::ostream &, const rect_element &);
std::ostream & operator<<(std::ostream &, text_element &);
std::ostream & operator<<(std::ostream &, const poly_path_point &);
std::ostream & operator<<(std::ostream &, polygon_element &);
std::ostream & operator<<(std::ostream &, polyline_element &);
bool operator==(const rect_element &, const rect_element &);

}
}

```

Struct a_path

boost::svg::a_path — Draws a elliptical arc from the current point to (x,y), using two radii, axis rotation, and two control flags.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

struct a_path : public boost::svg::path_point {
    // construct/copy/destruct
    a_path(double, double, double, double, double, bool = false, bool = false,
          bool = false);

    // public member functions
    virtual void write(std::ostream &);

    // public data members
    bool large_arc; // true if arc >= 180 degrees wanted.
    bool relative; // If true relative else absolute.
    double rx; // X radius.
    double ry; // Y radius.
    bool sweep; // true if to draw in positive-angle direction
    double x; // X End of arc from current point.
    double x_axis_rotation; // Any rotation of the X axis.
    double y; // Y End of arc from current point.
};
```

Description

See 8.3.8 The elliptical arc curve commands.! Useful for pie charts, etc.

a_path public construct/copy/destruct

1. `a_path(double x, double y, double rx, double ry, double x_axis_rotation,
bool large_arc = false, bool sweep = false, bool relative = false);`

Construct elliptic arc path.

a_path public member functions

1. `virtual void write(std::ostream & rhs);`

write functions output SVG commands like "M1.2, 3.4",

Write elliptical arc path XML to ostream.

Struct c_path

boost::svg::c_path — Draws a cubic Bezier curve from the current point to (x, y) using (x1, y1).

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

struct c_path : public boost::svg::path_point {
    // construct/copy/destruct
    c_path(double, double, double, double, double, bool = false);

    // public member functions
    virtual void write(std::ostream &);

    // public data members
    bool relative; // If true relative else absolute.
    double x; // Current (start point).
    double x1; // Middle of curve.
    double x2; // End point.
    double y; // Current (start point).
    double y1; // Middle of curve.
    double y2; // End point.
};
```

Description

8.3.5 The curve commands: C, Q & A.

c_path public construct/copy/destruct

1. `c_path(double x1, double y1, double x2, double y2, double x, double y, bool relative = false);`

< Constructor defines all member variables.

c_path public member functions

1. `virtual void write(std::ostream & rhs);`

write functions output SVG commands like "M1.2, 3.4",

< Write a cubic Bezier curve SVG XML to ostream.

Class circle_element

`boost::svg::circle_element` — Circle from center coordinate, and radius.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class circle_element : public boost::svg::svg_element {
public:
    // construct/copy/destruct
    circle_element(double, double, double = 5);
    circle_element(double, double, double, const svg_style &,
                  const std::string & = "", const std::string & = "",
                  const std::string & = "");

    // public member functions
    void class_id(const std::string &);
    std::string class_id();
    void clip_id(const std::string &);
    std::string clip_id();
    void id(const std::string &);
    std::string id();
    bool operator!=(const svg_element &);
    bool operator==(const svg_element &);
    svg_style & style();
    const svg_style & style() const;
    virtual void write(std::ostream &);

    // protected member functions
    void write_attributes(std::ostream &);

};
```

Description

Represents a single circle. <http://www.w3.org/TR/SVG/shapes.html#CircleElement>

circle_element public construct/copy/destruct

1. `circle_element(double x, double y, double radius = 5);`

Constructor defines all private data (default radius only).

2. `circle_element(double x, double y, double radius,
 const svg_style & style_info,
 const std::string & id_name = "",
 const std::string & class_name = "",
 const std::string & clip_name = "");`

Constructor defines all private data.

circle_element public member functions

1. `void class_id(const std::string & class_id);`

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: class="info"

2. `std::string class_id();`

Class id, non-unique identifier for an element.

3. `void clip_id(const std::string & id);`

Set name of a clip path, for example: `g_ptr.clip_id(plot_window_clip_);`

4. `std::string clip_id();`

Returns: name of a clip path, for example: `g_ptr.clip_id(plot_window_clip_);`

5. `void id(const std::string & id);`

Provide a unique name for an element. Example: `id="plotBackground"`

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: id and xml:base The id and xml:base attributes are available on all SVG elements: Attribute definitions: id = "name" Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. xml:base = "<uri>" Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the id attribute. Named groups are needed for several purposes such as animation and re-usable objects.

6. `std::string id();`

Returns: the unique name for an element, for example `id() = "plotBackground".`

7. `bool operator!=(const svg_element & lhs);`

Compare `svg_element`s for inequality, useful for Boost.Test.

8. `bool operator==(const svg_element & lhs);`

Compare `svg_element`s, useful for Boost.Test.

9. `svg_style & style();`

Returns: reference to `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()`

10. `const svg_style & style() const;`

Returns: reference to `const svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()` (const version).

11. `virtual void write(std::ostream & rhs);`

write functions output SVG commands.

Output SVG XML Example: `<circle cx="9.78571" cy="185" r="5"/>`

circle_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from `svg_element` add other references, 5.3.1, like color, fill, stroke, gradients...

Class clip_path_element

`boost::svg::clip_path_element` — The clipping path restricts the region to which paint can be applied.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class clip_path_element : public boost::svg::svg_element {
public:
    // construct/copy/destruct
    clip_path_element(const std::string &, const rect_element &);

    // public member functions
    void class_id(const std::string &);
    std::string class_id();
    void clip_id(const std::string &);
    std::string clip_id();
    void id(const std::string &);
    std::string id();
    bool operator!=(const svg_element &);
    bool operator==(const svg_element &);
    svg_style & style();
    const svg_style & style() const;
    virtual void write(std::ostream &);

    // protected member functions
    void write_attributes(std::ostream &);

    // public data members
    std::string element_id; // SVG element id.
    rect_element rect; // Clipping rectangle.
};
```

Description

14.3 Clipping paths <http://www.w3.org/TR/SVG/masking.html#ClipPathProperty>.

clip_path_element public construct/copy/destruct

1. `clip_path_element(const std::string & id, const rect_element & rect);`

< Constructor defines all member variables.

clip_path_element public member functions

1. `void class_id(const std::string & class_id);`

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: class="info"

2. `std::string class_id();`

Class id, non-unique identifier for an element.

3. `void clip_id(const std::string & id);`

Set name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

4. `std::string clip_id();`

Returns: name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

5. `void id(const std::string & id);`

Provide a unique name for an element. Example: id="plotBackground"

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: id and xml:base The id and xml:base attributes are available on all SVG elements: Attribute definitions: id = "name" Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. xml:base = "<uri>" Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the id attribute. Named groups are needed for several purposes such as animation and re-usable objects.

6. `std::string id();`

Returns: the unique name for an element, for example id() ="plotBackground".

7. `bool operator!=(const svg_element & lhs);`

Compare svg_elements for inequality, useful for Boost.Test.

8. `bool operator==(const svg_element & lhs);`

Compare svg_elements, useful for Boost.Test.

9. `svg_style & style();`

Returns: reference to `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()`

10. `const svg_style & style() const;`

Returns: reference to `const svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()` (const version).

11. `virtual void write(std::ostream & rhs);`

write functions output SVG commands.

< Write clip path to ostream.

clip_path_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from [svg_element](#) add other references, 5.3.1, like color, fill, stroke, gradients...

Class ellipse_element

`boost::svg::ellipse_element` — Ellipse from center coordinate, and radius.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class ellipse_element : public boost::svg::svg_element {
public:
    // construct/copy/destruct
    ellipse_element(double, double, double = 4, double = 8);
    ellipse_element(double, double, double, double, const svg_style &,
                   const std::string & = "", const std::string & = "",
                   const std::string & = "");
    ellipse_element(double, double, const svg_style &, const std::string & = "",
                   const std::string & = "", const std::string & = "");

    // public member functions
    void class_id(const std::string &);
    std::string class_id();
    void clip_id(const std::string &);
    std::string clip_id();
    void id(const std::string &);
    std::string id();
    bool operator!=(const svg_element &);
    bool operator==(const svg_element &);
    svg_style & style();
    const svg_style & style() const;
    virtual void write(std::ostream &);

    // protected member functions
    void write_attributes(std::ostream &);
};


```

Description

Represents a single ellipse. <http://www.w3.org/TR/SVG/shapes.html#EllipseElement> 9.4 The 'ellipse' element. Default is 'horizontal' but can be rotated.

ellipse_element public construct/copy/destruct

1. `ellipse_element(double cx, double cy, double rx = 4, double ry = 8);`

rotation in degrees from horizontal (default 0.).

< Constructor defines all private data (with default radii).

2. `ellipse_element(double cx, double cy, double rx, double ry,
const svg_style & style_info,
const std::string & id_name = "",
const std::string & class_name = "",
const std::string & clip_name = "");`

< Constructor sets ellipse and its style (defaults define all private data).

3. `ellipse_element(double cx, double cy, const svg_style & style_info,
const std::string & id_name = "",
const std::string & class_name = "",
const std::string & clip_name = "");`

< Constructor that also includes style, id, class and clip.

ellipse_element public member functions

1. `void class_id(const std::string & class_id);`

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: class="info"

2. `std::string class_id();`

Class id, non-unique identifier for an element.

3. `void clip_id(const std::string & id);`

Set name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

4. `std::string clip_id();`

Returns: name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

5. `void id(const std::string & id);`

Provide a unique name for an element. Example: id="plotBackground"

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: id and xml:base The id and xml:base attributes are available on all SVG elements: Attribute definitions: id = "name" Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. xml:base = "<uri>" Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the id attribute. Named groups are needed for several purposes such as animation and re-usable objects.

6. `std::string id();`

Returns: the unique name for an element, for example id() ="plotBackground".

7. `bool operator!=(const svg_element & lhs);`

Compare `svg_element` for inequality, useful for Boost.Test.

8. `bool operator==(const svg_element & lhs);`

Compare `svg_element`, useful for Boost.Test.

9. `svg_style & style();`

Returns: reference to `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()`

10. `const svg_style & style() const;`

Returns: reference to `const svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()` (const version).

11. `virtual void write(std::ostream & rhs);`

`write` functions output SVG commands.

Output SVG XML for ellipse. Example: `<ellipse rx="250" ry="100" fill="red" />`

ellipse_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from `svg_element` add other references, 5.3.1, like color, fill, stroke, gradients...

Class g_element

`boost::svg::g_element` — `g_element` (group element) is the node element of our document tree. 'g' element is a container element for grouping together:

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class g_element : public boost::svg::svg_element {
public:
    // construct/copy/destruct
    g_element();

    // public member functions
    g_element & add_g_element();
    circle_element & circle(double, double, unsigned int = 5);
    void class_id(const std::string &);

    std::string class_id();
    void clear();
    void clip_id(const std::string &);

    std::string clip_id();
    ellipse_element & ellipse(double, double, double);
    g_element & g(int);
    polygon_element &
    hexagon(double, double, double, double, double, double,
            double, double, double, double, bool = true);
    void id(const std::string &);

    std::string id();
    line_element & line(double, double, double);
    bool operator!=(const svg_element &);

    bool operator==(const svg_element &);

    svg_element & operator[](unsigned int);

    path_element & path();
    polygon_element &
    pentagon(double, double, double, double, double, double,
             double, double, bool = true);
    polygon_element & polygon(double, double, bool = true);
    polygon_element & polygon(std::vector< poly_path_point > &, bool = true);
    polygon_element & polygon();
    polyline_element & polyline(std::vector< poly_path_point > &);

    polyline_element & polyline(double, double);
    polyline_element & polyline();
    void push_back(svg_element *);

    rect_element & rect(double, double, double, double);
    polygon_element &
    rhombus(double, double, double, double, double, double,
            bool = true);
    size_t size();
    svg_style & style();
    const svg_style & style() const;
    text_element &
    text(double = 0., double = 0., const std::string & = "",
          const text_style & = no_style, const align_style & = left_align,
          const rotate_style & = horizontal);
    polygon_element &
    triangle(double, double, double, double, double, bool = true);
    virtual void write(std::ostream &);

    // protected member functions
    void write_attributes(std::ostream &);
}
```

```
// public data members
ptr_vector< svg_element > children;
std::string clip_name; // Name of clip path.
bool clip_on; // true if to clip anything outside the clip path.
};
```

Description

<g> ... </g>

g_element ('g' element is a container element for grouping together related graphics elements).

See <http://www.w3.org/TR/SVG/struct.html#NewDocument> 5.2.1 Overview.

'g' element is a container element for grouping together<g /> </g> related graphics elements, for example, an image background rectangle with border and fill: <g id="background" fill="rgb(255,255,255)"><rect width="500" height="350"/></g>

g_element public construct/copy/destruct

1. `g_element();`

Construct **g_element** (with no clipping).

g_element public member functions

1. `g_element & add_g_element();`

Add a new group element.

Returns: A reference to the new child node just created.

2. `circle_element & circle(double x, double y, unsigned int radius = 5);`

Add a new circle element.

Returns: A reference to the new child node just created.

3. `void class_id(const std::string & class_id);`

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: class="info"

4. `std::string class_id();`

Class id, non-unique identifier for an element.

5. `void clear();`

Remove all the child nodes.

6. `void clip_id(const std::string & id);`

Set name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

7. `std::string clip_id();`

Returns: name of a clip path, for example: `g_ptr.clip_id(plot_window_clip_);`

8. `ellipse_element & ellipse(double rx, double ry, double cx, double cy);`

Add a new ellipse element.

Returns: A reference to the new child node just created.

9. `g_element & g(int i);`

i is index of children nodes.

10. `polygon_element & hexagon(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4, double x5, double y5, double x6, double y6, bool f = true);`

Add a new hexagon element.

Returns: A reference to the new child node just created.

11. `void id(const std::string & id);`

Provide a unique name for an element. Example: `id="plotBackground"`

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: id and xml:base The id and xml:base attributes are available on all SVG elements: Attribute definitions: id = "name" Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. xml:base = "<uri>" Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the id attribute. Named groups are needed for several purposes such as animation and re-usable objects.

12. `std::string id();`

Returns: the unique name for an element, for example `id() = "plotBackground".`

13. `line_element & line(double x1, double y1, double x2, double y2);`

Add a new line element.

Returns: A reference to the new child node just created.

14. `bool operator!=(const svg_element & lhs);`

Compare `svg_element`s for inequality, useful for Boost.Test.

15. `bool operator==(const svg_element & lhs);`

Compare `svg_element`s, useful for Boost.Test.

16. `svg_element & operator[](unsigned int i);`

Returns: child `svg_element` node.

17. `path_element & path();`

Add a new path element.

Returns: A reference to the new path just created.

18. `polygon_element & pentagon(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4, double x5, double y5, bool f = true);`

Add a new pentagon element.

Returns: A reference to the new child node just created.

19. `polygon_element & polygon(double x, double y, bool f = true);`

Add a new polygon element.

Returns: A reference to the new child node just created.

20. `polygon_element & polygon(std::vector< poly_path_point > & v, bool f = true);`

Add a new complete polygon element.

Returns: A reference to the new child node just created.// push_back a complete many-sided polygon to the document.

21. `polygon_element & polygon();`

Add a new polygon element.

Returns: A reference to the new polygon element just created.

22. `polyline_element & polyline(std::vector< poly_path_point > & v);`

Add a new complete polyline.

Returns: A reference to the new child node just created.

23. `polyline_element & polyline(double x, double y);`

Add a new polyline element, but 1st point only, add others later with .P(x, y)...

Returns: A reference to the new child node just created.

24. `polyline_element & polyline();`

Add a new polyline element.

Returns: A reference to the new polyline element just created.

25. `void push_back(svg_element * g);`

Add a new child node `g_element`.

26. `rect_element & rect(double x1, double y1, double x2, double y2);`

Add a new rect element.

Returns: A reference to the new child node just created.

27. `polygon_element &`
`rhombus(double x1, double y1, double x2, double y2, double x3, double y3,`
`double x4, double y4, bool f = true);`

Add a new rhombus element.

Returns: A reference to the new child node just created.

28. `size_t size();`

Returns: Number of child nodes.

29. `svg_style & style();`

Returns: reference to `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()`

30. `const svg_style & style() const;`

Returns: reference to `const svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()` (const version).

31. `text_element &`
`text(double x = 0., double y = 0., const std::string & text = "",`
`const text_style & style = no_style,`
`const align_style & align = left_align,`
`const rotate_style & rotate = horizontal);`

Add a new text element.

Returns: A reference to the new child node just created.

32. `polygon_element &`
`triangle(double x1, double y1, double x2, double y2, double x3, double y3,`
`bool f = true);`

Add a new triangle element.

Returns: A reference to the new child node just created.

33. `virtual void write(std::ostream & rhs);`

write functions output SVG commands.

Output all children of a group element. Example: `<g fill="rgb(255,255,255)" id="background"><rect x="0" y="0" width="500" height="350"/></g>`

Avoid useless output like: `<g id="legendBackground"></g>` TODO check this doesn't mean that useful style is lost?

g_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from `svg_element` add other references, 5.3.1, like color, fill, stroke, gradients...

g_element public public data members

1. `ptr_vector< svg_element > children;`

Children of this group element node, containing graphics elements like text, circle, line, polyline...

Struct h_path

`boost::svg::h_path` — Draws a horizontal line from the current point (`cpx, cpy`) to (`x, cpy`). which becomes the new current point.
No `y` needed, start from current point `y`.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

struct h_path : public boost::svg::path_point {
    // construct/copy/destroy
    h_path(double, bool = false);

    // public member functions
    virtual void write(std::ostream &);

    // public data members
    bool relative; // If true relative else absolute.
    double x; // x horizontal SVG coordinate.
};
```

Description**h_path public construct/copy/destroy**

1. `h_path(double x, bool relative = false);`

< Constructor defines all member variables.

h_path public member functions

1. `virtual void write(std::ostream & rhs);`

write functions output SVG commands like "M1.2, 3.4",

Write horizontal line SVG command.

Struct l_path

`boost::svg::l_path` — Draw a line from the current point to the given (`x,y`) coordinate which becomes the new current point.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

struct l_path : public boost::svg::path_point {
    // construct/copy/destruct
    l_path(double, double, bool = false);

    // public member functions
    virtual void write(std::ostream &);

    // public data members
    bool relative; // If true relative else absolute.
    double x; // End of line SVG X coordinate.
    double y; // End of line SVG Y coordinate.
};
```

Description

l_path public construct/copy/destruct

1. `l_path(double x, double y, bool relative = false);`

Constructor defines all member variables.

l_path public member functions

1. `virtual void write(std::ostream & rhs);`

write functions output SVG commands like "M1.2, 3.4",

Write line to SVG command.

Class line_element

`boost::svg::line_element` — Line from (x1, y1) to (x2, y2). /details Straight line from SVG location (x1, y1) to (x2, y2).

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class line_element : public boost::svg::svg_element {
public:
    // construct/copy/destruct
    line_element(double, double, double, double);
    line_element(double, double, double, double, const svg_style &,
                const std::string & = "", const std::string & = "",
                const std::string & = "");

    // public member functions
    void class_id(const std::string &);
    std::string class_id();
    void clip_id(const std::string &);
    std::string clip_id();
    void id(const std::string &);
    std::string id();
    bool operator!=(const svg_element &);
    bool operator==(const svg_element &);
    svg_style & style();
    const svg_style & style() const;
    virtual void write(std::ostream &);

    // protected member functions
    void write_attributes(std::ostream &);

    // public data members
    double x1_; // Line from (x1_, x2_) to (y1_, y2_)
    double x2_; // Line from (x1_, x2_) to (y1_, y2_)
    double y1_; // Line from (x1_, x2_) to (y1_, y2_)
    double y2_; // Line from (x1_, x2_) to (y1_, y2_)
};

};
```

Description

line_element public construct/copy/destruct

1. line_element(double x1, double y1, double x2, double y2);

Constructor assigning all **line_element** private data.

2. line_element(double x1, double y1, double x2, double y2,
 const svg_style & style_info, const std::string & id_name = "",
 const std::string & class_name = "",
 const std::string & clip_name = "");

Constructor assigning all **line_element** private data, and also inherited **svg_element** data.

line_element public member functions

1. void class_id(const std::string & class_id);

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: class="info"

2. `std::string class_id();`

Class id, non-unique identifier for an element.

3. `void clip_id(const std::string & id);`

Set name of a clip path, for example: `g_ptr.clip_id(plot_window_clip_);`

4. `std::string clip_id();`

Returns: name of a clip path, for example: `g_ptr.clip_id(plot_window_clip_);`

5. `void id(const std::string & id);`

Provide a unique name for an element. Example: `id="plotBackground"`

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: id and xml:base The id and xml:base attributes are available on all SVG elements: Attribute definitions: id = "name" Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. xml:base = "<uri>" Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the id attribute. Named groups are needed for several purposes such as animation and re-usable objects.

6. `std::string id();`

Returns: the unique name for an element, for example `id() = "plotBackground".`

7. `bool operator!=(const svg_element & lhs);`

Compare `svg_element`s for inequality, useful for Boost.Test.

8. `bool operator==(const svg_element & lhs);`

Compare `svg_element`s, useful for Boost.Test.

9. `svg_style & style();`

Returns: reference to `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()`

10. `const svg_style & style() const;`

Returns: reference to `const svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()` (const version).

11. `virtual void write(std::ostream & rhs);`

write functions output SVG commands.

output line from $(x1_ , y1_)$ to $(x2_ , y2_)$ by writing XML SVG command to draw a straight line.

line_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from `svg_element` add other references, 5.3.1, like color, fill, stroke, gradients...

Struct m_path

`boost::svg::m_path` — move to coordinates (x, y)

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

struct m_path : public boost::svg::path_point {
    // construct/copy/destruct
    m_path(double, double, bool = false);

    // public member functions
    virtual void write(std::ostream &);

    // public data members
    bool relative; // If true relative else absolute.
    double x; // End of move SVG X coordinate.
    double y; // End of move SVG Y coordinate.
};
```

Description

See 8.3.2 The "moveto" commands.

m_path public construct/copy/destruct

1. `m_path(double x, double y, bool relative = false);`

Construct a move to

m_path public member functions

1. `virtual void write(std::ostream & rhs);`

write functions output SVG commands like "M1.2, 3.4",

write moveto X and Y coordinates to stream, for example: "M52.8571,180 "

Class path_element

`boost::svg::path_element` — Path element holds places on a path used by move, line ...

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class path_element : public boost::svg::svg_element {
public:
    // construct/copy/destruct
    path_element(const path_element &);
    path_element(const svg_style &, const std::string & = "", const std::string & = "", const std::string & = "");
    path_element();

    // public member functions
    path_element & c(double, double, double, double, double);
    path_element & C(double, double, double, double, double);
    void class_id(const std::string &);

    std::string class_id();
    void clip_id(const std::string &);

    std::string clip_id();
    path_element & fill_on(bool);
    bool fill_on();
    path_element & h(double);
    path_element & H(double);
    void id(const std::string &);

    std::string id();
    path_element & I(double, double);
    path_element & L(double, double);
    path_element & m(double, double);
    path_element & M(double, double);
    bool operator!=(const svg_element &);

    bool operator==(const svg_element &);

    path_element & q(double, double, double, double);
    path_element & Q(double, double, double, double);
    path_element & s(double, double, double, double);
    path_element & S(double, double, double, double);

    svg_style & style();
    const svg_style & style() const;
    path_element & t(double, double);
    path_element & T(double, double);
    path_element & v(double);
    path_element & V(double);

    virtual void write(std::ostream &);

    path_element & z();
    path_element & Z();

    // protected member functions
    void write_attributes(std::ostream &);

    // public data members
    ptr_vector<path_point> path; // All the (x, y) coordinate pairs, < filled by calls of m, M, I , L... that push_back.
};


```

Description

<http://www.w3.org/TR/SVG/paths.html#PathElement> 8.3.1 General information about path data. A path is defined by including a 'path' element which contains a d="(path data)" attribute, where the d attribute contains the moveto, line, curve (both cubic and quadratic Beziers), arc and closepath instructions.

path_element public construct/copy/destruct

1. `path_element(const path_element & rhs);`

Copy constructor.

2. `path_element(const svg_style & style_info, const std::string & id_name = "", const std::string & class_name = "", const std::string & clip_name = "");`

Construct empty path element.

3. `path_element();`

Construct an empty path element.

path_element public member functions

1. `path_element & c(double x1, double y1, double x2, double y2, double x, double y);`

Draws a cubic Bezier curve from the current point to (x,y) using (x1,y1).(relative).

Returns: `path_element&` to make chainable.

2. `path_element & C(double x1, double y1, double x2, double y2, double x, double y);`

Draws a cubic Bezier curve from the current point to (x,y) using (x1,y1).(absolute).

Returns: `path_element&` to make chainable.

3. `void class_id(const std::string & class_id);`

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: class="info"

4. `std::string class_id();`

Class id, non-unique identifier for an element.

5. `void clip_id(const std::string & id);`

Set name of a clip path, for example: `g_ptr.clip_id(plot_window_clip_);`

6. `std::string clip_id();`

Returns: name of a clip path, for example: `g_ptr.clip_id(plot_window_clip_);`

7. `path_element & fill_on(bool on_);`

Set area fill, on or off.

Returns: `path_element&` to make chainable.

8. `bool fill_on();`

Returns: area fill, on or off.

9. `path_element & h(double x);`

Line horizontal (relative).

Returns: `path_element&` to make chainable.

10. `path_element & H(double x);`

Line horizontal (absolute).

Returns: `path_element&` to make chainable.

11. `void id(const std::string & id);`

Provide a unique name for an element. Example: `id="plotBackground"`

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: id and xml:base The id and xml:base attributes are available on all SVG elements: Attribute definitions: id = "name" Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. xml:base = "<uri>" Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the id attribute. Named groups are needed for several purposes such as animation and re-usable objects.

12. `std::string id();`

Returns: the unique name for an element, for example `id() = "plotBackground"`.

13. `path_element & l(double x, double y);`

Line to (relative).

Returns: `path_element&` to make chainable.

14. `path_element & L(double x, double y);`

Line to (absolute).

Returns: `path_element&` to make chainable.

15. `path_element & m(double x, double y);`

Move relative by x and y.

Returns: `path_element&` to make chainable.

16. `path_element & M(double x, double y);`

Move to absolute x and y.

Returns: `path_element&` to make chainable.

17. `bool operator!=(const svg_element & lhs);`

Compare `svg_element`s for inequality, useful for Boost.Test.

18. `bool operator==(const svg_element & lhs);`

Compare `svg_element`s, useful for Boost.Test.

19. `path_element & q(double x1, double y1, double x, double y);`

Quadratic Curve Bezier (relative).

Returns: `path_element&` to make chainable.

20. `path_element & Q(double x1, double y1, double x, double y);`

Quadratic Curve Bezier (absolute).

Returns: `path_element&` to make chainable.

21. `path_element & s(double x1, double y1, double x, double y);`

Draws a cubic Bezier curve from the current point to (x,y) (relative).

Returns: `path_element&` to make chainable.

22. `path_element & S(double x1, double y1, double x, double y);`

Draws a cubic Bezier curve from the current point to (x,y) (absolute).

Returns: `path_element&` to make chainable.

23. `svg_style & style();`

Returns: reference to `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()`

24. `const svg_style & style() const;`

Returns: reference to `const svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()` (const version).

25. `path_element & t(double x, double y);`

Draws a quadratic Bezier curve from the current point to (x,y)(relative).

Returns: `path_element&` to make chainable.

26. `path_element & T(double x, double y);`

Draws a quadratic Bezier curve from the current point to (x,y)(absolute).

Returns: `path_element&` to make chainable.

27. `path_element & v(double y);`

Line vertical (relative).

Returns: `path_element&` to make chainable.

28. `path_element & V(double y);`

Line vertical (absolute).

Returns: `path_element&` to make chainable.

29. `virtual void write(std::ostream & rhs);`

write functions output SVG commands.

Write SVG path command. Example:<path d="M5,175 L5,195 M148.571,175" />

30. `path_element & z();`

Path end. Note lower case z, see `path_element& Z()` below.

Returns: `path_element&` to make chainable.

31. `path_element & Z();`

Path end. Note Upper case Z also provided for compatibility with <http://www.w3.org/TR/SVG/patterns.html#PathDataClosePathCommand> 8.3.3 which allows either case.

Returns: `path_element&` to make chainable.

path_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from `svg_element` add other references, 5.3.1, like color, fill, stroke, gradients...

Struct path_point

`boost::svg::path_point` — Base class for `m_path`, `z_path`, `q_path`, `h_path`, `v_path`, `c_path`, `s_path`.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

struct path_point {
    // construct/copy/destruct
    path_point(bool);
    ~path_point();

    // public member functions
    virtual void write(std::ostream &) = 0;

    // public data members
    bool relative; // If true relative else absolute.
};
```

Description

Paths represent the outline of a shape which can be filled, stroked, used as a clipping path, or any combination of the three.

path_point public construct/copy/destruct

1. `path_point(bool relative);`

< Constructor defines all member variables.

2. `~path_point();`

Destructor.

path_point public member functions

1. `virtual void write(std::ostream & rhs) = 0;`

write functions output SVG commands like "M1.2, 3.4",

Struct poly_path_point

`boost::svg::poly_path_point` — polyline or polygon point coordinates (x, y)

Synopsis

// In header: <boost/svg_plot/detail/svg_tag.hpp>

```
struct poly_path_point {
    // construct/copy/destruct
    poly_path_point(double, double);
    poly_path_point();

    // public member functions
    void write(std::ostream &);

    // public data members
    double x; // polygon or polyline path point X SVG coordinate.
    double y; // polygon or polyline path point Y SVG coordinate.
};
```

Description

9.6 polyline & 9.7 The 'polygon' element.

poly_path_point public construct/copy/destruct

1. `poly_path_point(double x, double y);`

Construct a polygon or polyline path point from X and Y SVG coordinate.

2. `poly_path_point();`

Default constructor.

poly_path_point public member functions

1. `void write(std::ostream & o_str);`

Output SVG XML, Example: " 250,180" Leading space is redundant for 1st after "points= ", but others are separators, and awkward to know which is 1st.

Struct polygon_element

boost::svg::polygon_element — The 'polygon' element defines a closed shape consisting of a set of connected straight line segments.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

struct polygon_element : public boost::svg::svg_element {
    // construct/copy/destruct
    polygon_element(const polygon_element &);
    polygon_element();
    polygon_element(double, double, bool = true);
    polygon_element(double, double, double, double, double, double, bool = true);
    polygon_element(double, double, double, double, double, double, double,
                   double, bool = true);
    polygon_element(double, double, double, double, double, double, double,
                   double, double, bool = true);
    polygon_element(double, double, double, double, double, double, double,
                   double, double, double, bool = true);
    polygon_element(std::vector< poly_path_point > &, bool = true);

    // public member functions
    void class_id(const std::string &);
    std::string class_id();
    void clip_id(const std::string &);
    std::string clip_id();
    void id(const std::string &);
    std::string id();
    bool operator!=(const svg_element &);
    std::ostream & operator<<(std::ostream &);
    bool operator==(const svg_element &);
    polygon_element & P(double, double);
    svg_style & style();
    const svg_style & style() const;
    virtual void write(std::ostream &);

    // protected member functions
    void write_attributes(std::ostream &);

    // public data members
    bool fill; // polygon to have fill color.
    std::vector< poly_path_point > poly_points; // All the x, y coordinate pairs, < push_backed by calls of p_path(x, □
    y).
};

}
```

Description

<http://www.w3.org/TR/SVG/shapes.html#PolygonElement> The 'polygon' element 9.9.7. A polygon is defined by including a 'path' element which contains a points="(path data)" attribute, where the d attribute contains the x, y coordinate pairs.

polygon_element public construct/copy/destruct

1. `polygon_element(const polygon_element & rhs);`

Copy constructor.

2. `polygon_element();`

Default constructor empty polygon (with fill on).

3. `polygon_element(double x, double y, bool f = true);`

Constructor - One absolute (x, y) point only. Can add more path points using member function P.

4. `polygon_element(double x1, double y1, double x2, double y2, double x3, double y3, bool f = true);`

Constructor - Absolute (x, y) only. Used by triangle.

5. `polygon_element(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4, bool f = true);`

Constructor - Absolute (x, y) only. Used by rhombus.

6. `polygon_element(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4, double x5, double y5, bool f = true);`

Constructor - Absolute (x, y) only. Used by pentagon.

7. `polygon_element(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4, double x5, double y5, double x6, double y6, bool f = true);`

Constructor - Six absolute (x, y) only. Used by hexagon.

8. `polygon_element(std::vector< poly_path_point > & points, bool f = true);`

Constructor from vector of path points.

polygon_element public member functions

1. `void class_id(const std::string & class_id);`

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: class="info"

2. `std::string class_id();`

Class id, non-unique identifier for an element.

3. `void clip_id(const std::string & id);`

Set name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

4. `std::string clip_id();`

Returns: name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

5. `void id(const std::string & id);`

Provide a unique name for an element. Example: id="plotBackground"

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: id and xml:base The id and xml:base attributes are available on all SVG elements: Attribute definitions: id = "name" Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. xml:base = "<uri>" Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the id attribute. Named groups are needed for several purposes such as animation and re-usable objects.

6. `std::string id();`

Returns: the unique name for an element, for example id() ="plotBackground".

7. `bool operator!=(const svg_element & lhs);`

Compare svg_elements for inequality, useful for Boost.Test.

8. `std::ostream & operator<<(std::ostream & os);`

Output polygon info. (May be useful for Boost.Test. using os << "(" << p.x << ", " << p.y << ")" ; Usage: `polygon_element p(1, 2, 3, 4, 5, 6); my_polygon.operator << (cout);` (But NOT cout << my_polygon << endl;) Outputs: (1, 2)(3, 4)(5, 6)

9. `bool operator==(const svg_element & lhs);`

Compare svg_elements, useful for Boost.Test.

10. `polygon_element & P(double x, double y);`

Add another point (x, y) - absolute only.

Returns: `polygon_element&` to make chainable.

11. `svg_style & style();`

Returns: reference to `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()`

12. `const svg_style & style() const;`

Returns: reference to `const svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()` (const version).

13. `virtual void write(std::ostream & rhs);`

write functions output SVG commands.

SVG XML: Example: `<polygon fill="lime" stroke="blue" stroke-width="10" points="850,75 958,137.5 958,262.5 850,325 742,262.6 742,137.5" />`

polygon_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from [svg_element](#) add other references, 5.3.1, like color, fill, stroke, gradients...

Class polyline_element

`boost::svg::polyline_element` — The 'polyline' element: defines a set of connected straight line segments.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class polyline_element : public boost::svg::svg_element {
public:
    // construct/copy/destruct
    polyline_element(const polyline_element &);
    polyline_element();
    polyline_element(double, double);
    polyline_element(double, double, double, double);
    polyline_element(std::vector< poly_path_point > &);

    // public member functions
    void class_id(const std::string &);
    std::string class_id();
    void clip_id(const std::string &);
    std::string clip_id();
    void id(const std::string &);
    std::string id();
    bool operator!=(const svg_element &);
    bool operator==(const svg_element &);
    polyline_element & P(double, double);
    svg_style & style();
    const svg_style & style() const;
    virtual void write(std::ostream &);

    // protected member functions
    void write_attributes(std::ostream &);

    // public data members
    ptr_vector< poly_path_point > poly_points; // All the (x, y) coordinate pairs.,
};


```

Description

<http://www.w3.org/TR/SVG/shapes.html#PolylineElement> 9.6 The polyline element: defines a set of connected straight line segments. Typically, polyline elements define open shapes. A polyline is defined by including a 'path' element which contains a points="(path data)" attribute, where the points attribute contains the x, y coordinate pairs. perform an absolute moveto operation to the first coordinate pair in the list of points for each subsequent coordinate pair, perform an absolute lineto operation to that coordinate pair. The advantage of polyline is in reducing file size, avoiding M and repeated L before x & y coordinate pairs.

polyline_element public construct/copy/destruct

1. `polyline_element(const polyline_element & rhs);`

copy constructor.

2. `polyline_element();`

Construct an 'empty' line. Can new line path points add using [polyline_element](#) member function P.

3. `polyline_element(double x1, double y1);`

One (x, y) path point, absolute only.

4. `polyline_element(double x1, double y1, double x2, double y2);`

Two (x, y) path points, absolute only.

5. `polyline_element(std::vector< poly_path_point > & points);`

Constructor from vector of path points.

polyline_element public member functions

1. `void class_id(const std::string & class_id);`

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: class="info"

2. `std::string class_id();`

Class id, non-unique identifier for an element.

3. `void clip_id(const std::string & id);`

Set name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

4. `std::string clip_id();`

Returns: name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

5. `void id(const std::string & id);`

Provide a unique name for an element. Example: id="plotBackground"

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: id and xml:base The id and xml:base attributes are available on all SVG elements: Attribute definitions: id = "name" Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. xml:base = "<uri>" Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the id attribute. Named groups are needed for several purposes such as animation and re-usable objects.

6. `std::string id();`

Returns: the unique name for an element, for example id() ="plotBackground".

7. `bool operator!=(const svg_element & lhs);`

Compare svg_elements for inequality, useful for Boost.Test.

8. `bool operator==(const svg_element & lhs);`

Compare `svg_element`, useful for Boost.Test.

9. `polyline_element & P(double x, double y);`

Absolute (x, y) only, so Capital letter P.

Returns: `polyline_element&` to make chainable.

10. `svg_style & style();`

Returns: reference to `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()`

11. `const svg_style & style() const;`

Returns: reference to `const svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()` (const version).

12. `virtual void write(std::ostream & rhs);`

write functions output SVG commands.

Output polyline info (useful for Boost.Test). Example: `<polyline points=" 100,100 200,100 300,200 400,400"/>`

polyline_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from `svg_element` add other references, 5.3.1, like color, fill, stroke, gradients...

Struct q_path

`boost::svg::q_path` — Draws a quadratic Bezier curve from the current point to (x,y). using (x1,y1) as the control point.

Synopsis

// In header: <boost/svg_plot/detail/svg_tag.hpp>

```
struct q_path : public boost::svg::path_point {
    // construct/copy/destruct
    q_path(double, double, double, double, bool = false);

    // public member functions
    virtual void write(std::ostream &);

    // public data members
    bool relative; // If true relative else absolute.
    double x; // quadratic Bezier curve end X coordinate.
    double x1; // quadratic Bezier curve control X coordinate.
    double y; // quadratic Bezier curve end Y coordinate.
    double y1; // quadratic Bezier curve control Y coordinate.
};
```

Description

q_path public construct/copy/destruct

1. `q_path(double x1, double y1, double x, double y, bool relative = false);`

Constructor quadratic Bezier curve.

q_path public member functions

1. `virtual void write(std::ostream & rhs);`

write functions output SVG commands like "M1.2, 3.4",

< Write a quadratic Bezier curve SVG XML to ostream.

Class **curve_element**

`boost::svg::curve_element` — Quadratic Bezier curved line from (x1, y1) control point (x2, y2) to (x3, y3).

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class curve_element : public boost::svg::svg_element {
public:
    // construct/copy/destruct
    curve_element(double, double, double, double, double, double);
    curve_element(double, double, double, double, double, double,
        const svg_style &, const std::string & = "", 
        const std::string & = "", const std::string & = "");

    // public member functions
    void class_id(const std::string &);
    std::string class_id();
    void clip_id(const std::string &);
    std::string clip_id();
    void id(const std::string &);
    std::string id();
    bool operator!=(const svg_element &);
    bool operator==(const svg_element &);
    svg_style & style();
    const svg_style & style() const;
    virtual void write(std::ostream &);

    // protected member functions
    void write_attributes(std::ostream &);

    // public data members
    double x1_; // Quadratic curved line from (x1_, y1_) control point (x2_, y2_) to (y3_, y3_).
    double x2_; // Quadratic curved line from (x1_, y1_) control point (x2_, y2_) to (y3_, y3_).
    double x3_; // Quadratic curved line from (x1_, y1_) control point (x2_, y2_) to (y3_, y3_).
    double y1_; // Quadratic curved line from (x1_, y1_) control point (x2_, y2_) to (y3_, y3_).
    double y2_; // Quadratic curved line from (x1_, y1_) control point (x2_, y2_) to (y3_, y3_).
    double y3_; // Quadratic curved line from (x1_, y1_) control point (x2_, y2_) to (y3_, y3_).
};


```

Description

Note x2 is the Bezier control point - the curve will **not** pass thru this point.

curve_element public construct/copy/destruct

1.

```
curve_element(double x1, double y1, double x2, double y2, double x3,
              double y3);
```

< Quadratic curved line constructor (info inherited from parent **svg_element** class).

2.

```
curve_element(double x1, double y1, double x2, double y2, double x3,
              double y3, const svg_style & style_info,
              const std::string & id_name = "", 
              const std::string & class_name = "", 
              const std::string & clip_name = "");
```

< Quadratic curved line constructor, including **svg_element** info.

curve_element public member functions

1. `void class_id(const std::string & class_id);`

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: class="info"

2. `std::string class_id();`

Class id, non-unique identifier for an element.

3. `void clip_id(const std::string & id);`

Set name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

4. `std::string clip_id();`

Returns: name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

5. `void id(const std::string & id);`

Provide a unique name for an element. Example: id="plotBackground"

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: id and xml:base The id and xml:base attributes are available on all SVG elements: Attribute definitions: id = "name" Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. xml:base = "<uri>" Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the id attribute. Named groups are needed for several purposes such as animation and re-usable objects.

6. `std::string id();`

Returns: the unique name for an element, for example id() ="plotBackground".

7. `bool operator!=(const svg_element & lhs);`

Compare **svg_element**s for inequality, useful for Boost.Test.

8. `bool operator==(const svg_element & lhs);`

Compare **svg_element**s, useful for Boost.Test.

9. `svg_style & style();`

Returns: reference to **svg_style** to provide indirect access to colors & width via style().stroke_color(), fill_color(), width()

10. `const svg_style & style() const;`

Returns: reference to const **svg_style** to provide indirect access to colors & width via style().stroke_color(), fill_color(), width() (const version).

11. `virtual void write(std::ostream & rhs);`

write functions output SVG commands.

output quadratic curved line from (x1_, y1_) control point (x2_, y2_) to (x3_, y3_)

Example:

curve_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from [svg_element](#) add other references, 5.3.1, like color, fill, stroke, gradients...

Class rect_element

boost::svg::rect_element — Rectangle from top left coordinate, width and height.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class rect_element : public boost::svg::svg_element {
public:
    // construct/copy/destruct
    rect_element(double, double, double, double);
    rect_element(double, double, double, double, const svg_style &,
        const std::string &, const std::string &, const std::string &);

    // public member functions
    void class_id(const std::string &);
    std::string class_id();
    void clip_id(const std::string &);
    std::string clip_id();
    double height() const;
    void id(const std::string &);
    std::string id();
    bool operator!=(const svg_element &);
    bool operator!=(const rect_element &);
    bool operator==(const svg_element &);
    bool operator==(const rect_element &);
    svg_style & style();
    const svg_style & style() const;
    double width() const;
    virtual void write(std::ostream &);

    double x() const;
    double y() const;

    // protected member functions
    void write_attributes(std::ostream &);

    // public data members
    double height_; // y + height is bottom left. < x + width and y + height is bottom right.
    double width_; // x + width is top right.
    double x_; // X-axis coordinate of the side of the rectangle which has the smaller x-axis coordinate value.
    double y_; // Y-axis coordinate of the side of the rectangle which has the smaller y-axis coordinate value. < So □
    (0, 0) is top left corner of rectangle.
};


```

Description

Represents a single rectangle. <http://www.w3.org/TR/SVG/shapes.html#RectElement>

rect_element public construct/copy/destruct

1. `rect_element(double x, double y, double w, double h);`

Constructor defines all private data (no defaults).

2. `rect_element(double x, double y, double w, double h,
 const svg_style & style_info, const std::string & id_name,
 const std::string & class_name, const std::string & clip_name);`

Constructor defines all private data (inherits info from `svg_element`).

rect_element public member functions

1. `void class_id(const std::string & class_id);`

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: class="info"

2. `std::string class_id();`

Class id, non-unique identifier for an element.

3. `void clip_id(const std::string & id);`

Set name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

4. `std::string clip_id();`

Returns: name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

5. `double height() const;`

y + height is bottom left.

6. `void id(const std::string & id);`

Provide a unique name for an element. Example: id="plotBackground"

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: id and xml:base The id and xml:base attributes are available on all SVG elements: Attribute definitions: id = "name" Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. xml:base = "<uri>" Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the id attribute. Named groups are needed for several purposes such as animation and re-usable objects.

7. `std::string id();`

Returns: the unique name for an element, for example id() ="plotBackground".

8. `bool operator!=(const svg_element & lhs);`

Compare svg_elements for inequality, useful for Boost.Test.

9. `bool operator!=(const rect_element & lhs);`

< Comparison rect_elements (useful for Boost.Test).

10. `bool operator==(const svg_element & lhs);`

Compare svg_elements, useful for Boost.Test.

11. `bool operator==(const rect_element & lhs);`

Comparison (useful for Boost.Test).

12. `svg_style & style();`

Returns: reference to `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()`

13. `const svg_style & style() const;`

Returns: reference to const `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()` (const version).

14. `double width() const;`

`x + width` is top right.

15. `virtual void write(std::ostream & rhs);`

write functions output SVG commands.

Output SVG XML for rectangle. For example: `<rect x="0" y="0" width="500" height="350"/>`

16. `double x() const;`

x-axis coordinate of the side of the rectangle which has the smaller x-axis coordinate value.

17. `double y() const;`

y-axis coordinate of the side of the rectangle which has the smaller y-axis coordinate value.

rect_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from `svg_element` add other references, 5.3.1, like color, fill, stroke, gradients...

Struct s_path

`boost::svg::s_path` — Draws a cubic Bezier curve from the current point to (x,y).

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

struct s_path : public boost::svg::path_point {
    // construct/copy/destruct
    s_path(double, double, double, double, bool = false);

    // public member functions
    virtual void write(std::ostream &);

    // public data members
    bool relative; // If true relative else absolute.
    double x; // Cubic Bezier curve end X coordinate.
    double x1; // Cubic Bezier curve control X coordinate.
    double y; // Cubic Bezier curve end Y coordinate.
    double y1; // Cubic Bezier curve control Y coordinate.
};
```

Description

see also [t_path](#) for a quadratic Bezier curve.

s_path public construct/copy/destruct

1. `s_path(double x1, double y1, double x, double y, bool relative = false);`

Constructor Cubic Bezier curve.

s_path public member functions

1. `virtual void write(std::ostream & rhs);`

write functions output SVG commands like "M1.2, 3.4",

Write SVG Cubic Bezier curve command.

Class svg_element

`boost::svg::svg_element` — [svg_element](#) is base class for all the leaf elements.

Synopsis

// In header: <boost/svg_plot/detail/svg_tag.hpp>

```
class svg_element {
public:
    // construct/copy/destruct
    svg_element(const svg_style &, const std::string & = "",
                const std::string & = "", const std::string & = "");
    svg_element();
    ~svg_element();

    // protected member functions
    void write_attributes(std::ostream &);

    // public member functions
    void class_id(const std::string &);
    std::string class_id();
    void clip_id(const std::string &);
    std::string clip_id();
    void id(const std::string &);
    std::string id();
    bool operator!=(const svg_element &);
    bool operator==(const svg_element &);
    svg_style & style();
    const svg_style & style() const;
    virtual void write(std::ostream &) = 0;
};
```

Description

'g' element is a container element, <g ... /></g> for grouping together related graphics elements, for example: <g stroke="rgb(255,0,0)"><rect x="0" y="0" width="500" height="600"/></g>

`rect_element`, `circle_element`, `line_element`, `text_element`, `polygon_element`, `polyline_element`, `path_element`, `clip_path_element`, `g_element`.

`g_element` ('g' element is a container element for grouping together related graphics elements).

See <http://www.w3.org/TR/SVG/struct.html#NewDocument> 5.2.1 Overview.

svg_element public construct/copy/destruct

1. `svg_element(const svg_style & style_info, const std::string & id_name = "", const std::string & class_name = "", const std::string & clip_name = "");`

Constructor with some defaults.

2. `svg_element();`

Default constructor.

3. `~svg_element();`

destructor.

svg_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from [svg_element](#) add other references, 5.3.1, like color, fill, stroke, gradients...

svg_element public member functions

1. `void class_id(const std::string & class_id);`

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: class="info"

2. `std::string class_id();`

Class id, non-unique identifier for an element.

3. `void clip_id(const std::string & id);`

Set name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

4. `std::string clip_id();`

Returns: name of a clip path, for example: g_ptr.clip_id(plot_window_clip_);

5. `void id(const std::string & id);`

Provide a unique name for an element. Example: id="plotBackground"

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: id and xml:base The id and xml:base attributes are available on all SVG elements: Attribute definitions: id = "name" Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. xml:base = "<uri>" Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the id attribute. Named groups are needed for several purposes such as animation and re-usable objects.

6. `std::string id();`

Returns: the unique name for an element, for example id() ="plotBackground".

7. `bool operator!=(const svg_element & lhs);`

Compare svg_elements for inequality, useful for Boost.Test.

8. `bool operator==(const svg_element & lhs);`

Compare svg_elements, useful for Boost.Test.

9. `svg_style & style();`

Returns: reference to `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()`

10. `const svg_style & style() const;`

Returns: reference to const `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()` (const version).

11. `virtual void write(std::ostream & rhs) = 0;`

write functions output SVG commands.

Struct t_path

`boost::svg::t_path` — Draws a quadratic Bezier curve from the current point to (x,y).

Synopsis

// In header: <[boost/svg_plot/detail/svg_tag.hpp](#)>

```
struct t_path : public boost::svg::path_point {
    // construct/copy/destruct
    t_path(double, double, bool = false);

    // public member functions
    virtual void write(std::ostream &);

    // public data members
    bool relative; // If true relative else absolute.
    double x; // SVG X coordinate.
    double y; // SVG Y coordinate.
};
```

Description

see also `s_path` for a cubic Bezier curve.

t_path public construct/copy/destruct

1. `t_path(double x, double y, bool relative = false);`

Constructor of path that draws a quadratic Bezier curve from the current point to (x,y)

t_path public member functions

1. `virtual void write(std::ostream & rhs);`

write functions output SVG commands like "M1.2, 3.4",

Write SVG command for a cubic Bezier curve.

Class `text_element`

`boost::svg::text_element` — Holds text with position, size, font, (& styles) & orientation.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class text_element : public boost::svg::svg_element {
public:
    // construct/copy/destruct
    text_element(double = 0., double = 0., const std::string = "",
                 text_style = no_style, align_style = left_align,
                 rotate_style = horizontal),
    text_element(const text_element &);
    text_element & operator=(const text_element &);

    // private member functions
    void generate_text(std::ostream &);

    // public member functions
    text_element & alignment(align_style);
    align_style alignment();
    void class_id(const std::string &);
    std::string class_id();
    void clip_id(const std::string &);
    std::string clip_id();
    void id(const std::string &);
    std::string id();
    bool operator!=(const svg_element &);
    bool operator==(const svg_element &);
    text_element & rotation(rotate_style);
    rotate_style rotation() const;
    svg_style & style();
    const svg_style & style() const;
    void text(const std::string &);
    std::string text();
    text_style & textstyle();
    const text_style & textstyle() const;
    text_element & textstyle(text_style &);

    tspan_element & tspan(const std::string &,
                         const text_style &);

    virtual void write(std::ostream &);

    text_element & x(double);
    double x() const;
    text_element & y(double);
    double y() const;

    // protected member functions
    void write_attributes(std::ostream &);

};
```

Description

Not necessarily shown correctly (or nicely) by all browsers, alas. SVG Coordinates of 1st character EM box, see <http://www.w3.org/TR/SVG/text.html#TextElement> 10.2

So any text with y coordinate = 0 shows only any roman lower case descenders!
 (Text may contain embedded xml Unicode characters for Greek, math etc, for example: ©).

```
int size; // " font-size = 12" http://www.w3.org/TR/SVG/text.html#CharactersAndGlyphs std::string font; // font-family: "Arial" | "Times New Roman" | "Verdana" | "Lucida Sans Unicode" "sans", "serif", "times"
http://www.w3.org/TR/SVG/text.html#FontFamilyProperty 10.10 Font selection properties
std::string style_; // font-style: normal | bold | italic | oblique std::string weight; // font-weight: normal | bold | bolder | lighter | 100 | 200 .. 900 std::string stretch; // font-stretch: normal | wider | narrower ... std::string decoration; /// "underline" | "overline" | "line-through" Example: <text x="250" y="219.5" text-anchor="middle" font-family="verdana" font-size="12">0 </text>
```

text_element public construct/copy/destruct

1. `text_element(double x = 0., double y = 0., const std::string text = "", text_style ts = no_style, align_style align = left_align, rotate_style rotate = horizontal);`

`text_element` Default Constructor, defines defaults for all private members.

2. `text_element(const text_element & rhs);`

Copy constructor.

3. `text_element & operator=(const text_element & rhs);`

Assignment operator.

Returns: `text_element&` to make chainable.

text_element private member functions

1. `void generate_text(std::ostream & os);`

text_element public member functions

1. `text_element & alignment(align_style a);`

`left_align`, `right_align`, `center_align`

Returns: `text_element&` to make chainable.

2. `align_style alignment();`

`left_align`, `right_align`, `center_align`

3. `void class_id(const std::string & class_id);`

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: `class="info"`

4. `std::string class_id();`

Class id, non-unique identifier for an element.

5. `void clip_id(const std::string & id);`

Set name of a clip path, for example: `g_ptr.clip_id(plot_window_clip_);`

6. `std::string clip_id();`

Returns: name of a clip path, for example: `g_ptr.clip_id(plot_window_clip_);`

7. `void id(const std::string & id);`

Provide a unique name for an element. Example: `id="plotBackground"`

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: id and xml:base The id and xml:base attributes are available on all SVG elements: Attribute definitions: id = "name" Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. xml:base = "<uri>" Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the id attribute. Named groups are needed for several purposes such as animation and re-usable objects.

8. `std::string id();`

Returns: the unique name for an element, for example `id() = "plotBackground".`

9. `bool operator!=(const svg_element & lhs);`

Compare `svg_element`s for inequality, useful for Boost.Test.

10. `bool operator==(const svg_element & lhs);`

Compare `svg_element`s, useful for Boost.Test.

11. `text_element & rotation(rotation_style rot);`

Degrees: horizontal = 0, upward = -90, downward, upsidedown Generates: transform = "rotate(-45 100 100)"

Returns: `text_element&` to make chainable.

12. `rotation_style rotation() const;`

Returns: rotation of text element.

13. `svg_style & style();`

Returns: reference to `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()`

14. `const svg_style & style() const;`

Returns: reference to `const svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()` (const version).

15. `void text(const std::string & t);`

Get tspan text string to write.

16. `std::string text();`

Returns: text string of a `text_element`.

17. `text_style & textstyle();`

Get text style for font size, family, decoration ...

18. `const text_style & textstyle() const;`

Get text style for font size, family, decoration ...

19. `text_element & textstyle(text_style & ts);`

Set text style for font size, family, decoration ...

Returns: `text_element&` to make chainable.

20. `tspan_element & tspan(const std::string & t);`

Add text span element.

21. `tspan_element & tspan(const std::string & t, const text_style & style);`

Add text span element (with specified text style).

22. `virtual void write(std::ostream & rhs);`

write functions output SVG commands.

Output `text_element`, style & attributes to stream.

23. `text_element & x(double x);`

x coordinate of text to write.

Returns: `text_element&` to make chainable.

24. `double x() const;`

x coordinate of text to write.

25. `text_element & y(double y);`

y coordinate of text to write.

Returns: `text_element&` to make chainable.

26. `double y() const;`

y coordinate of text to write.

text_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from [svg_element](#) add other references, 5.3.1, like color, fill, stroke, gradients...

Class text_element_text

`boost::svg::text_element_text` — text (not tspan) element to be stored in [text_parent](#).

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class text_element_text : public boost::svg::text_parent {
public:
    // construct/copy/destruct
    text_element_text(const std::string &);

    text_element_text(const text_element_text &);

    // public member functions
    virtual void write(std::ostream &);

};
```

Description

See 10.4 text element <http://www.w3.org/TR/SVG/text.html#TextElement>

text_element_text public construct/copy/destruct

1. `text_element_text(const std::string & text);`

Construct from text.

2. `text_element_text(const text_element_text & rhs);`

Copy construct text element

text_element_text public member functions

1. `virtual void write(std::ostream & o_str);`

write text to stream.

Class text_parent

`boost::svg::text_parent` — An ancestor to both tspan and strings for the [text_element](#) class.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class text_parent {
public:
    // construct/copy/destruct
    text_parent(const std::string &);
    text_parent(const text_parent &);

    // public member functions
    virtual void write(std::ostream &);

};
```

Description

This allows an array of both types to be stored in `text_element`.

text_parent public construct/copy/destruct

1. `text_parent(const std::string & text);`

Construct from text.

2. `text_parent(const text_parent & rhs);`

Copy construct.

text_parent public member functions

1. `virtual void write(std::ostream &);`

write functions output SVG commands.

Class `tspan_element`

`boost::svg::tspan_element` — `tspan` (not `text`) to be stored in `text_parent`.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

class tspan_element :
    public boost::svg::text_parent, public boost::svg::svg_element
{
public:
    // construct/copy/destruct
    tspan_element(const std::string &, const text_style & = no_style);
    tspan_element(const tspan_element &);

    // public member functions
    void class_id(const std::string &);
    std::string class_id();
    void clip_id(const std::string &);
    std::string clip_id();
    tspan_element & dx(double);
    double dx();
    tspan_element & dy(double);
    double dy();
    tspan_element & fill_color(const svg_color &);
    svg_color fill_color();
    bool fill_on();
    tspan_element & font_family(const std::string &);
    const std::string & font_family();
    tspan_element & font_size(unsigned int);
    unsigned int font_size();
    tspan_element & font_style(const std::string &);
    const std::string & font_style();
    const std::string & font_style() const;
    tspan_element & font_weight(const std::string &);
    const std::string & font_weight() const;
    void id(const std::string &);
    std::string id();
    bool operator!=(const svg_element &);
    bool operator==(const svg_element &);
    tspan_element & rotation(int);
    int rotation();
    svg_style & style();
    const svg_style & style() const;
    tspan_element & text(const std::string &);
    std::string text();
    tspan_element & text_length(double);
    double text_length();
    tspan_element & textstyle(const text_style &);
    const text_style & textstyle();
    const text_style & textstyle() const;
    bool use_style();
    virtual void write(std::ostream &);

    tspan_element & x(double);
    double x();
    tspan_element & y(double);
    double y();

    // protected member functions
    void write_attributes(std::ostream &);

};
```

Description

See 10.5 `tspan` element <http://www.w3.org/TR/SVG/text.html#TSpanElement>

tspan_element public construct/copy/destruct

1. `tspan_element(const std::string & text, const text_style & style = no_style);`

Construct `tspan` element (with all defaults except text string).

2. `tspan_element(const tspan_element & rhs);`

Copy constructor.

tspan_element public member functions

1. `void class_id(const std::string & class_id);`

Class class id, non-unique identifier for an element.

<http://www.w3.org/TR/2001/REC-SVG-20010904/styling.html#ClassAttribute> 6.12 Attributes common to all elements: id and xml:base Example: `class="info"`

2. `std::string class_id();`

Class id, non-unique identifier for an element.

3. `void clip_id(const std::string & id);`

Set name of a clip path, for example: `g_ptr.clip_id(plot_window_clip_);`

4. `std::string clip_id();`

Returns: name of a clip path, for example: `g_ptr.clip_id(plot_window_clip_);`

5. `tspan_element & dx(double dx);`

Relative X position of a 1st single character of text string to use with SVG `tspan` command.

Returns: `tspan_element&` to make chainable.

6. `double dx();`

Get relative X position for `tspan` element.

7. `tspan_element & dy(double dy);`

Relative Y position of a 1st single character of text string to use with SVG `tspan` command.

Returns: `tspan_element&` to make chainable.

8. `double dy();`

Get relative Y position for tspan element.

9. `tspan_element & fill_color(const svg_color & color);`

Set fill color for a tspan element.

Returns: `tspan_element&` to make chainable.

10. `svg_color fill_color();`

Get the fill color for tspan element .

11. `bool fill_on();`

Get true if to use fill color for tspan element .

12. `tspan_element & font_family(const std::string & family);`

font family of 1st single character of text string to use with SVG tspan command.

Returns: `tspan_element&` to make chainable.

13. `const std::string & font_family();`

Get the font family for tspan element (from its `text_style`).

14. `tspan_element & font_size(unsigned int size);`

font size of 1st single character of text string to use with SVG tspan command.

Returns: `tspan_element&` to make chainable.

15. `unsigned int font_size();`

Get the font size for tspan element (from its `text_style`).

16. `tspan_element & font_style(const std::string & style);`

font style of 1st single character of text string to use with SVG tspan command. font-style: normal | bold | italic | oblique Examples: "italic" http://www.croczilla.com/~alex/conformance_suite/svg/text-fonts-02-t.svg

Returns: `tspan_element&` to make chainable.

17. `const std::string & font_style();`

Get the font style for tspan element (from its `text_style`).

18. `const std::string & font_style() const;`

Get the font style for tspan element (from its `text_style`). const version.

19. `tspan_element & font_weight(const std::string & w);`

font weight of 1st single character of text string to use with SVG `tspan` command. `svg font-weight: normal | bold | bolder | lighter | 100 | 200 .. 900` Examples: "bold", "normal" http://www.croczilla.com/~alex/conformance_suite/svg/text-fonts-02-t.svg tests conformance. Only two weights are supported by Firefox, Opera, Inkscape.

Returns: `tspan_element&` to make chainable.

20. `const std::string & font_weight() const;`

Get the font weight for `tspan` element (from its `text_style`).

21. `void id(const std::string & id);`

Provide a unique name for an element. Example: `id="plotBackground"`

See <http://www.w3.org/TR/SVG/struct.html#IDAttribute> 5.10.1 Attributes common to all elements: `id` and `xml:base` The `id` and `xml:base` attributes are available on all SVG elements: Attribute definitions: `id = "name"` Standard XML attribute for assigning a unique name to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]. `xml:base = "<uri>"` Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE]. A group of elements, as well as individual objects, can be given a name using the `id` attribute. Named groups are needed for several purposes such as animation and re-usable objects.

22. `std::string id();`

Returns: the unique name for an element, for example `id() = "plotBackground"`.

23. `bool operator!=(const svg_element & lhs);`

Compare `svg_element`s for inequality, useful for Boost.Test.

24. `bool operator==(const svg_element & lhs);`

Compare `svg_element`s, useful for Boost.Test.

25. `tspan_element & rotation(int rotation);`

< Note implementation so far only rotates the 1st character in string. < `text_element` rotation rotates the whole text string, so it much more useful.

Returns: `tspan_element&` to make chainable.

26. `int rotation();`

Get rotation for the next character for `tspan` element.

27. `svg_style & style();`

Returns: reference to `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()`

28. `const svg_style & style() const;`

Returns: reference to const `svg_style` to provide indirect access to colors & width via `style().stroke_color()`, `fill_color()`, `width()` (const version).

29. `tspan_element & text(const std::string & text);`

Set text string to use with SVG tspan command.

Returns: `tspan_element&` to make chainable.

30. `std::string text();`

Get text from a tspan element.

31. `tspan_element & text_length(double text_length);`

Set user estimate of text length (see <http://www.w3.org/TR/SVG/text.html#TSpanElement> TSPAN SVG Specification).

Returns: `tspan_element&` to make chainable.

32. `double text_length();`

Get user estimated length for a text string.

33. `tspan_element & textstyle(const text_style & style);`

Set text style (font) for a tspan element.

Returns: `tspan_element&` to make chainable.

34. `const text_style & textstyle();`

Returns: `text_style&` to permit access to font family, size ...

35. `const text_style & textstyle() const;`

Returns: `text_style&` to permit access to font family, size (const version).

36. `bool use_style();`

Get true if to use the estimated text string length.

37. `virtual void write(std::ostream & rhs);`

write functions output SVG commands.

Output SVG XML for `tspan_element`

38. `tspan_element & x(double x);`

Absolute X position of a 1st single character of text string to use with SVG tspan command.

Returns: `tspan_element&` to make chainable.

39. `double x();`

Get absolute X position for tspan element.

40. `tspan_element & y(double y);`

Absolute Y position of a 1st single character of text string to use with SVG tspan command.

Returns: `tspan_element&` to make chainable.

41. `double y();`

Get absolute Y position for tspan element.

tspan_element protected member functions

1. `void write_attributes(std::ostream & s_out);`

Output group_element id and clip-path.

Classes inherited from `svg_element` add other references, 5.3.1, like color, fill, stroke, gradients...

Struct v_path

`boost::svg::v_path` — Draws a vertical line from the current point (cpx, cpy) to (cpx, y). No x coordinate needed - use current point x.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

struct v_path : public boost::svg::path_point {
    // construct/copy/destruct
    v_path(double, bool = false);

    // public member functions
    virtual void write(std::ostream &);

    // public data members
    bool relative; // If true relative else absolute.
    double y; // Y vertical line SVG coordinate.
};
```

Description

v_path public construct/copy/destruct

1. `v_path(double y, bool relative = false);`

< Constructor (defines all member variables).

v_path public member functions

1. `virtual void write(std::ostream & rhs);`

write functions output SVG commands like "M1.2, 3.4",

Write vertical line SVG command.

Struct z_path

`boost::svg::z_path` — Close current path.

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

struct z_path : public boost::svg::path_point {
    // construct/copy/destruct
    z_path();

    // public member functions
    virtual void write(std::ostream &);

    // public data members
    bool relative; // If true relative else absolute.
};
```

Description

<http://www.w3.org/TR/SVG/paths.html#PathElement> 8.3.1 General information about path data. Close the current subpath by drawing a straight line from the current point to current subpath's initial point.

z_path public construct/copy/destruct

1. `z_path();`

Constructor defines all member variables.

z_path public member functions

1. `virtual void write(std::ostream & rhs);`

write functions output SVG commands like "M1.2, 3.4",

Write close current path SVG command.

Function operator!=

`boost::svg::operator!=`

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

bool operator!=(const rect_element & lhs, const rect_element & rhs);
```

Description

Compare inequality of two SVG rect_elements.

Function operator<<

```
boost::svg::operator<<
```

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

std::ostream & operator<<(std::ostream & os, const rect_element & r);
```

Description

Example: `rect_element r(20, 20, 50, 50); cout << r << endl;` Outputs: `rect(20, 20, 50, 50)`

Function operator<<

```
boost::svg::operator<<
```

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

std::ostream & operator<<(std::ostream & os, text_element & t);
```

Description

Outputs: text & style (useful for diagnosis). Usage: `text_element t(20, 30, "sometest", left_align, horizontal); cout << t << endl;`

Function operator<<

```
boost::svg::operator<<
```

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

std::ostream & operator<<(std::ostream & os, const poly_path_point & p);
```

Description

Output may be useful for Boost.Test. Usage: `poly_path_point p0(100, 200); cout << p0 << endl;` Outputs: (100, 200)

Function operator<<

`boost::svg::operator<<`

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

std::ostream & operator<<(std::ostream & os, polygon_element & p);
```

Description

Output poly_path_ points (May be useful for Boost.Test). `ptr_vector<poly_path_point> poly_points;` All the x, y coordinate pairs, Usage: `polygon_element p(1, 2, 3, 4, 5, 6); cout << p << endl;` Outputs: (1, 2)(3, 4)(5, 6)

Function operator<<

`boost::svg::operator<<`

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

std::ostream & operator<<(std::ostream & os, polyline_element & p);
```

Description

Output polyline info (useful for Boost.Test). Example: `<polyline points=" 100,100 200,100 300,200 400,400"/>` `ptr_vector<poly_path_point> poly_points;` // All the x, y coordinate pairs.

Function operator==

`boost::svg::operator==`

Synopsis

```
// In header: <boost/svg_plot/detail/svg_tag.hpp>

bool operator==(const rect_element & lhs, const rect_element & rhs);
```

Description

Compare equality of two SVG rect_elements.

Header <boost/svg_plot/quantile.hpp>

Estimate p th quantile of data.

Estimate p th quantile of data using one of 5 definitions. Default is the recommendation of Hyndman and Fan = definition #8.

Hyndman and Fan recommend their definition 8 (Maple's default definition), which gives quartiles between those reported by Minitab and Excel. This approach is approximately median unbiased for continuous distributions.

Hyndman and Fan, 1996, "Sample Quantiles in Statistical Packages", The American Statistician 50(4):361-365,1996
<http://www.pcreview.co.uk/forums/thread-3494699.php> // Excel - Interquartile Range Miscalculation

The interquartile range is calculated using the 1st & 3rd sample quartiles, but there are various ways to calculate those quartiles. Excel, S-Plus, etc use H&F definition 7, which returns SMALL(data,i) as quantile(data,(i-1)/(n-1)) and interpolates in between. For a continuous distribution, this will tend to give too narrow an interquartile range, since there will tend to be a small fraction of the population beyond the extreme sample observations. In particular, for odd n (=2*k+1), Excel calculates the 1st (3rd) quartile as the median of the lower (upper) "half" of the sample including the sample median (k+1 observations).

Minitab, etc use H&F definition 6, which calculates the 1st (3rd) quartile as the median of the lower (upper) "half" of the sample. This "half" sample excludes the sample median (k observations) for odd n (=2*k+1). This will tend to be a better estimate for the population quartiles, but will tend to give quartile estimates that are a bit too far from the center of the whole sample (too wide an interquartile range).

```
namespace boost {
    namespace svg {
        double median(vector< double > & );
        double quantile(vector< double > &, double, int = 8);
    }
}
```

Function median

boost::svg::median

Synopsis

```
// In header: <boost/svg_plot/quantile.hpp>

double median(vector< double > & data);
```

Description

Median of values in vector data.



Note

Assumes pre-sorted from min to max.

Function quantile

boost::svg::quantile

Synopsis

```
// In header: <boost/svg_plot/quantile.hpp>

double quantile(vector< double > & data, double p, int HF_definition = 8);
```

Description

Estimate quantile from values in vector data.



Note

Assumes values are pre-sorted from min to max.

Parameters:	HF_definition	Algorithm to use for the estimation.
	data	Population for which to estimate quantile (percentile). Data must be ordered minimum to maximum.
	p	Fraction of population, for example, p = 0.25 for 1st quartile. (Usually p = 0.25, or p = 0.75 for boxplots).
Returns:	Estimated quantile from the population.	

Header <boost/svg_plot/show_1d_settings.hpp>

Shows settings and options for 1D Plot.

Paul A. Bristow

Mar 2009

See Also:

show_2d_settings.cpp for 2-D plot.

```
namespace boost {
namespace svg {
const char * fmtFlagWords; // Strings describing each bit in std::ios_base::fmtflags.

// Outputs strings to show horizontal orientation: left, right or none.
const std::string l_or_r(int i);
void outFmtFlags(std::ios_base::fmtflags, std::ostream &, const char *);
void show_1d_plot_settings(svg_1d_plot &);

// Outputs strings to show vertical orientation: top, bottom, or none.
const std::string t_or_b(int i);
}
```

Global fmtFlagWords

boost::svg::fmtFlagWords — Strings describing each bit in std::ios_base::fmtflags.

Synopsis

```
// In header: <boost/svg_plot/show_1d_settings.hpp>
const char * fmtFlagWords;
```

Function outFmtFlags

boost::svg::outFmtFlags

Synopsis

```
// In header: <boost/svg_plot/show_1d_settings.hpp>

void outFmtFlags(std::ios_base::fmtflags fmtFlags, std::ostream & os,
                 const char * term);
```

Description

Output strings describing all bits in std::ios_base::fmtflags.

Usage: outFmtFlags(); For example, by default outputs to std::cerr
 FormatFlags: skipws showbase right dec."
 Default parameter values are:

```
void outFmtFlags(fmtflags fmtFlags = cout.flags(), ostream& os = cerr, const char* term = ".\n");
```

Function show_1d_plot_settings

boost::svg::show_1d_plot_settings

Synopsis

```
// In header: <boost/svg_plot/show_1d_settings.hpp>

void show_1d_plot_settings(svg_1d_plot & plot);
```

Description

Diagnostic display to std::cout of all settings of a 1D plot. Outputs a long list of about hundred of plot parameter settings to cout. This list is invaluable if the plot does not look as expected.



Warning

This creates about 100 lines of output, so should be used sparingly!

Header <[boost/svg_plot/show_2d_settings.hpp](#)>

Shows settings and options for 2D Plot.

See Also:

show_1d_settings.cpp for 1D plot.

```
namespace boost {
    namespace svg {
        void show_2d_plot_settings(svg_2d_plot &);
    }
}
```

Function `show_2d_plot_settings`

`boost::svg::show_2d_plot_settings`

Synopsis

```
// In header: <boost/svg_plot/show_2d_settings.hpp>

void show_2d_plot_settings(svg_2d_plot & plot);
```

Description

Diagnostic display of all settings of a 2D plot. Outputs a long list of over 100 plot parameter settings to `std::cout`. This list is invaluable if the plot does not look as expected.



Warning

This creates about 100 lines of output, so should be used sparingly!

Header <[boost/svg_plot/svg.hpp](#)>

Scalable Vector Graphic (SVG) format elements.

Provides classes and methods to create the basic SVG graph elements. Graph elements, point, path, line, circle, rect and polygon, text are used by the 1D, 2D and Boxplot functions, but could also be used for generating other graphics in SVG format.

Jacob Voytko & Paul A. Bristow

```
namespace boost {
    namespace math {
    }
    namespace svg {
        class svg;

        static const std::string package_info; // Default SVG package information about this program that produced □
                                            // the SVG image (not the image itself).
        namespace boxplot {
        }
    }
}namespace boost {
}
```

Class `svg`

`boost::svg::svg` — Class to output Scalable Vector Graph XML graph elements: point, path, line, circle, rect, polygon and text.

Synopsis

```
// In header: <boost/svg_plot/svg.hpp>

class svg {
public:
    // construct/copy/destruct
    svg();
    svg(const svg &);

    // public member functions
    g_element & add_g_element();
    const std::string & attribution();
    void author(const std::string);
    const std::string & author();
    void boost_license_on(bool);
    bool boost_license_on();
    circle_element & circle(double, double, unsigned int = 5);
    clip_path_element & clip_path(const rect_element &, const std::string &);
    const std::string & commercialuse();
    void coord_precision(int);
    int coord_precision();
    void copyright_date(const std::string);
    const std::string copyright_date();
    void copyright_holder(const std::string);
    const std::string copyright_holder();
    void description(const std::string);
    const std::string & description();
    const std::string & distribution();
    unsigned int document_size();
    void document_title(const std::string);
    const std::string document_title();
    ellipse_element & ellipse(double, double, double);
    g_element & g(int);
    polygon_element &
    hexagon(double, double, double, double, double, double,
            double, double, double, double, bool = true);
    void image_filename(const std::string);
    const std::string image_filename();
    void license(const std::string = "permits", const std::string = "permits",
                const std::string = "requires", const std::string = "permits",
                const std::string = "permits");
    void license_on(bool);
    bool license_on();
    line_element & line(double, double, double, double);
    path_element & path();
    polygon_element &
    pentagon(double, double, double, double, double, double,
             double, double, double, bool = true);
    polygon_element & polygon(double, double, bool = true);
    polygon_element & polygon(std::vector< poly_path_point > &, bool = true);
    polyline_element & polyline(double, double);
    polyline_element & polyline(double, double, double, double);
    polyline_element & polyline(std::vector< poly_path_point > &);
    rect_element & rect(double, double, double, double);
    const std::string & reproduction();
    polygon_element &
```

```

rhombus(double, double, double, double, double, double, double,
       bool = true);
void size(unsigned int, unsigned int);
text_element &
text(double, double, const std::string &, const text_style &, align_style,
     rotate_style);
polygon_element &
triangle(double, double, double, double, double, bool = true);
void write(const std::string &);
void write(std::ostream &);
void x_size(unsigned int);
unsigned int x_size();
std::pair< double, double > xy_sizes();
void y_size(unsigned int);
unsigned int y_size();
};

```

Description

Class to add basic Scalable Vector Graph XML graph elements: point, path, line, circle, rect, polygon and text to SVG images, including metadata like author, copyright and license. Finally output the final image as SVG XML to a `std::ostream` or file.

svg public construct/copy/destruct

1. `svg();`

Define default constructor.

2. `svg(const svg & rhs);`

Copy constructor copies X and Y image sizes.

svg public member functions

1. `g_element & add_g_element();`

Add information about a group element to the document. Increments the size of the array of `g_elements`, returned by `g_element.size()`.

Returns: reference to the added group element.

2. `const std::string & attribution();`

Returns: License attribution requirement.

3. `void author(const std::string a);`

Set author for the SVG document (default is <copyright_holder>).

4. `const std::string & author();`

Returns: author of the SVG document (for header as <author>).

5. `void boost_license_on(bool l);`

Set (or not) to include Boost license text in `svg` header as comment.

6. `bool boost_license_on();`

Return true if a boost license has been requested in the `svg` header as comment.

7. `circle_element & circle(double x, double y, unsigned int radius = 5);`

push_back information about a circle to the document. 'circle' element defines a circle centered at (x1, y1) and its radius.

8. `clip_path_element &`
`clip_path(const rect_element & rect, const std::string & id);`

Rectangle outside which 'painting' is 'clipped' so doesn't show.

Returns: Reference to `clip_path` element.

9. `const std::string & commercialuse();`

Returns: License commercial use requirement.

10. `void coord_precision(int digits);`

Set decimal digits to be output for X and Y coordinates.

Default stream precision 6 decimal digits is probably excessive.

4.1 Basic data types, integer or float in decimal or scientific (using e format). 3 or 4 probably enough if image size is under 1000 x 1000. This will reduce .svg file sizes significantly for curves represented with many data points.

For example, if a curve is shown using 100 points, reducing to precision(3) from 6 will reduce file size by 300 bytes. So a default of 3 is used in the default constructor above, but can be changed using this function. Used in `svg.write` below and so applies to all the entire `svg` document.

11. `int coord_precision();`

Returns: Decimal digits to be output for X and Y coordinates.

12. `void copyright_date(const std::string copyright_date);`

Set copyright date for the SVG document (for header as <copyright_date>).

13. `const std::string copyright_date();`

Returns: copyright date for the SVG document (for header as <copyright_date>).

14. `void copyright_holder(const std::string copyright_holder);`

Set document title for the SVG document (for header as <copyright_holder>).

15. `const std::string copyright_holder();`

Returns: document title for the SVG document (for header as <copyright_holder>).

16. `void description(const std::string d);`

Write description to the SVG document (for header as <desc> ... </desc>).

17. `const std::string & description();`

Returns: description of the SVG document (for header as <desc>).

18. `const std::string & distribution();`

Returns: License distribution requirement.

19. `unsigned int document_size();`

Returns: How many group elements groups have been added to the document.

20. `void document_title(const std::string title);`

Set document title for the SVG document (for header as <title> ... </title>).

21. `const std::string document_title();`

Returns: document title for the SVG document (for header as <title>).

22. `ellipse_element & ellipse(double rx, double ry, double cx, double cy);`

push_back information about a ellipse to the document. 'ellipse' element defines a ellipse centered at point (x1, y1) and its two radii.

23. `g_element & g(int i);`

from array of g_elements, indexed by group type, PLOT_BACKGROUND, PLOT_WINDOW_BACKGROUND, ...
SVG_PLOT_DOC_CHILDREN,

Returns: reference to the ith group element.

24. `polygon_element &
hexagon(double x1, double y1, double x2, double y2, double x3, double y3,
double x4, double y4, double x5, double y5, double x6, double y6,
bool f = true);`

push_back the coordinate of the points of a complete hexagon to the document.

25. `void image_filename(const std::string filename);`

Set image filename for the SVG document (for header as <filename>).

26. `const std::string image_filename();`

Returns: image filename for the SVG document (for header as <filename>).

27. `void license(const std::string reproduction = "permits",
const std::string distribution = "permits",
const std::string attribution = "requires",
const std::string commercialuse = "permits",
const std::string derivative = "permits");`

Set several license requirements for the svg document. If any are set, then a license is wanted, so `svg::is_license` is set true. This can be changed using function `license_on()`.

28. `void license_on(bool l);`

Set (or not) license using all requirements (default permits).

Implicitly set by setting any license requirement using `license` function.

29. `bool license_on();`

Return true if a license has been requested for `svg` header metataadata.

30. `line_element & line(double x1, double y1, double x2, double y2);`

Add (push_back) information about a line to the document. 'Line' element defines a line segment that starts at one point (x_1, y_1) and ends at another (x_2, y_2).

31. `path_element & path();`

Construct an empty path, ready for additions with chainable functions M., L., ...

Returns: reference to path element.

32. `polygon_element &
pentagon(double x1, double y1, double x2, double y2, double x3, double y3,
double x4, double y4, double x5, double y5, bool f = true);`

push_back the five coordinates complete pentagon to the document.

33. `polygon_element & polygon(double x, double y, bool f = true);`

push_back info about 1st point of a polygon shape (add others later with .P(x, y)).

34. `polygon_element & polygon(std::vector< poly_path_point > & v, bool f = true);`

push_back a complete many-sided polygon to the document with vertices specified as a vector of `path_points`.

35. `polyline_element & polyline(double x, double y);`

push_back info about the 1st point of a polyline (add others later with .P(x, y)).

36. `polyline_element & polyline(double x1, double y1, double x2, double y2);`

push_back info about the 1st & 2nd point of a polyline (add others later with .P(x, y)).

37. `polyline_element & polyline(std::vector< poly_path_point > & v);`

push_back a complete many-sided polygon to the document, from a vector of path_points.

38. `rect_element & rect(double x1, double y1, double x2, double y2);`

push_back information about a rectangle to the document. 'Rect' element defines a rect segment with one point (x1, y1) and opposite vertex is (x2, y2).

39. `const std::string & reproduction();`

Returns: License reproduction requirement.

40. `polygon_element &`
`rhombus(double x1, double y1, double x2, double y2, double x3, double y3,`
`double x4, double y4, bool f = true);`

push_back the four coordinate of a complete rhombus to the document.

41. `void size(unsigned int x, unsigned int y);`

Set both X and Y image size (SVG units, default pixels).

42. `text_element &`
`text(double x, double y, const std::string & text, const text_style & style,`
`align_style align, rotate_style rotate);`

push_back information about text to the document, with location, style, alignment & rotation.

43. `polygon_element &`
`triangle(double x1, double y1, double x2, double y2, double x3, double y3,`
`bool f = true);`

push_back a complete triangle to the document.

44. `void write(const std::string & filename);`

Write whole .svg 'file' contents to file.

svg.write() also has two flavors, a file and an ostream. The file version opens an ostream, and calls the stream version. The stream version first clears all unnecessary data from the graph, builds the document tree, and then calls the write function for the root document node, which calls all other nodes through the Visitor pattern.

TODO provide a filtered-stream version that writes in zipped format type .svgz ?
<http://lists.w3.org/Archives/Public/www-svg/2005Dec/0308.html> recommends MUST have correct Content-Encoding headers.

45. `void write(std::ostream & s_out);`

Write whole .svg 'file' contents to stream (perhaps a file).

46. `void x_size(unsigned int x);`

Set X-axis (horizontal) image size.

47. `unsigned int x_size();`

Returns: X-axis (horizontal width) SVG image size.

48. `std::pair< double, double > xy_sizes();`

Returns: Both X and Y sizes (horizontal width and vertical height) of the SVG image.

49. `void y_size(unsigned int y);`

Set Y-axis (vertical) image size.

50. `unsigned int y_size();`

Returns: Y-axis (vertical height) SVG image size.

Global package_info

`boost::svg::package_info` — Default SVG package information about this program that produced the SVG image (not the image itself).

Synopsis

```
// In header: <boost/svg_plot/svg.hpp>
static const std::string package_info;
```

Description

Inserted as a SVG comment, for example

"<!-- Demo of 1D plot features. -->"

and also as a

<desc><http://www.w3.org/TR/SVG/struct.html#DescriptionAndTitleElements>

Section 5.4 The 'desc' and 'title' elements.

Default XML comment is:

<!-- SVG plot written using Boost.Plot program (Creator Jacob Voytko) -->

<!-- Use, modification and distribution of Boost.Plot subject to the -->

<!-- Boost Software License, Version 1.0.-->

<!-- (See accompanying file LICENSE_1_0.txt -->

<!-- or copy at http://www.boost.org/LICENSE_1_0.txt -->

Header <[boost/svg_plot/svg_1d_plot.hpp](#)>

Create 1D plots in Scalable Vector Graphic (SVG) format.

Provides `svg_1d_plot` data and function to create plots, and `svg_1d_plot_series` to allow data values to be added.

Very many functions allow fine control of the appearance and layout of plots and data markers.

(Items common to 1D and 2D use functions and classes in `axis_plot_frame`).

```
namespace boost {
namespace svg {
    class svg_1d_plot;
    class svg_1d_plot_series;
}
}
```

Class `svg_1d_plot`

`boost::svg::svg_1d_plot` — All settings for a plot that control the appearance, and functions to get and set these settings.
(But see [svg_1d_plot_series](#) to control appearance of data points).

Synopsis

```
// In header: <boost/svg_plot/svg_1d_plot.hpp>

class svg_1d_plot {
public:

    // public member functions
    bool autoscale();
    svg_1d_plot & autoscale(bool);
    svg_1d_plot & autoscale_check_limits(bool);
    bool autoscale_check_limits();
    svg_1d_plot & autoscale_plusminus(double);
    double autoscale_plusminus();
    svg_1d_plot & axes_on(bool);
    bool axes_on();
    svg_1d_plot & background_border_color(const svg_color &);
    svg_color background_border_color();
    svg_1d_plot & background_border_width(double);
    double background_border_width();
    svg_color background_color();
    svg_1d_plot & background_color(const svg_color &);
    svg_1d_plot & boost_license_on(bool);
    bool boost_license_on();
    svg_1d_plot & confidence(double);
    double confidence();
    svg_1d_plot & coord_precision(int);
    int coord_precision();
    svg_1d_plot & copyright_date(const std::string);
    const std::string copyright_date();
    svg_1d_plot & copyright_holder(const std::string);
    const std::string copyright_holder();
    svg_1d_plot & data_lines_width(double);
    double data_lines_width();
    svg_1d_plot & derived();
    const svg_1d_plot & derived() const;
    svg_1d_plot & description(const std::string);
    const std::string & description();
    svg_1d_plot & document_title(const std::string);
    std::string document_title();
    svg_1d_plot &
    draw_line(double, double, double, const svg_color & = black);
    svg_1d_plot &
    draw_note(double, double, std::string, rotate_style = horizontal,
              align_style = center_align, const svg_color & = black,
              text_style & = no_style);
    svg_1d_plot &
```

```

draw_plot_curve(double, double, double, double, double, double,
    const svg_color & = black);
svg_1d_plot &
draw_plot_line(double, double, double, const svg_color & = black);
svg_1d_plot & image_border_margin(double);
double image_border_margin();
svg_1d_plot & image_border_width(double);
double image_border_width();
unsigned int image_x_size();
svg_1d_plot & image_x_size(unsigned int);
unsigned int image_y_size();
svg_1d_plot & image_y_size(unsigned int);
svg_1d_plot & legend_background_color(const svg_color &);
svg_color legend_background_color();
svg_1d_plot & legend_border_color(const svg_color &);
svg_color legend_border_color();
const std::pair< double, double > legend_bottom_right();
bool legend_box_fill_on();
svg_1d_plot & legend_color(const svg_color &);
svg_color legend_color();
svg_1d_plot & legend_font_family(const std::string &);
const std::string & legend_font_family();
svg_1d_plot & legend_font_weight(const std::string &);
const std::string & legend_font_weight();
svg_1d_plot & legend_header_font_size(int);
int legend_header_font_size();
svg_1d_plot & legend_lines(bool);
bool legend_lines();
svg_1d_plot & legend_on(bool);
bool legend_on();
bool legend_outside();
svg_1d_plot & legend_place(legend_places);
legend_places legend_place();
svg_1d_plot & legend_title(const std::string);
const std::string legend_title();
svg_1d_plot & legend_title_font_size(unsigned int);
unsigned int legend_title_font_size();
svg_1d_plot & legend_top_left(double, double);
const std::pair< double, double > legend_top_left();
svg_1d_plot & legend_width(double);
double legend_width();
svg_1d_plot &
license(std::string = "permits", std::string = "permits",
    std::string = "requires", std::string = "permits",
    std::string = "permits");
const std::string license_attribution();
const std::string license_commercialuse();
const std::string license_distribution();
svg_1d_plot & license_on(bool);
bool license_on();
const std::string license_reproduction();
svg_1d_plot & limit_color(const svg_color &);
svg_color limit_color();
svg_1d_plot & limit_fill_color(const svg_color &);
svg_color limit_fill_color();
svg_1d_plot & one_sd_color(const svg_color &);
svg_color one_sd_color();
template<typename T>
    svg_1d_plot_series & plot(const T &, const std::string & = "");
template<typename T>
    svg_1d_plot_series & plot(const T &, const T &, const std::string & = "");
template<typename T, typename U>
    svg_1d_plot_series &

```

```

plot(const T &, const std::string & = "", U = unspecified);
template<typename T, typename U>
svg_1d_plot_series &
plot(const T &, const T &, const std::string & = "", U = unspecified);
svg_1d_plot & plot_background_color(const svg_color &);

svg_color plot_background_color();
svg_1d_plot & plot_border_color(const svg_color &);

svg_color plot_border_color();
svg_1d_plot & plot_border_width(double);

double plot_border_width();
svg_1d_plot & plot_window_on(bool);

bool plot_window_on();
svg_1d_plot & plot_window_x(double, double);
std::pair<double, double> plot_window_x();

double plot_window_x_left();
double plot_window_x_right();
svg_1d_plot & plot_window_y(double, double);
std::pair<double, double> plot_window_y();

double plot_window_y_bottom();
double plot_window_y_top();
svg_1d_plot & size(unsigned int, unsigned int);
std::pair<double, double> size();

svg_1d_plot & three_sd_color(const svg_color &);

svg_color three_sd_color();
svg_1d_plot & title(const std::string &);

const std::string title();
svg_1d_plot & title_color(const svg_color &);

svg_color title_color();
svg_1d_plot & title_font_alignment(align_style);

align_style title_font_alignment();
svg_1d_plot & title_font_decoration(const std::string &);

const std::string & title_font_decoration();

svg_1d_plot & title_font_family(const std::string &);

const std::string & title_font_family();

svg_1d_plot & title_font_rotation(rotate_style);

int title_font_rotation();

svg_1d_plot & title_font_size(unsigned int);

unsigned int title_font_size();

svg_1d_plot & title_font_stretch(const std::string &);

const std::string & title_font_stretch();

svg_1d_plot & title_font_style(const std::string &);

const std::string & title_font_style();

svg_1d_plot & title_font_weight(const std::string &);

const std::string & title_font_weight();

svg_1d_plot & title_on(bool);

bool title_on();

svg_1d_plot & two_sd_color(const svg_color &);

svg_color two_sd_color();

svg_1d_plot & write(const std::string &);

svg_1d_plot & write(std::ostream &);

svg_1d_plot & x_addlimits_color(const svg_color &);

svg_color x_addlimits_color();

svg_1d_plot & x_addlimits_on(bool);

bool x_addlimits_on();

double x_auto_max_value();

double x_auto_min_value();

double x_auto_tick_interval();

int x_auto_ticks();

bool x_autoscale();

svg_1d_plot & x_autoscale(bool);

svg_1d_plot & x_autoscale(std::pair<double, double>);

svg_1d_plot & x_autoscale(const T &);

svg_1d_plot & x_autoscale(const T &, const T &);

```

```

svg_1d_plot & x_axis_color(const svg_color &);
svg_color x_axis_color();
svg_1d_plot & x_axis_label_color(const svg_color &);
svg_color x_axis_label_color();
svg_1d_plot & x_axis_on(bool);
bool x_axis_on();
const std::string x_axis_position();
svg_1d_plot & x_axis_vertical(double);
bool x_axis_vertical();
svg_1d_plot & x_axis_width(double);
double x_axis_width();
svg_1d_plot & x_datetime_color(const svg_color &);
svg_color x_datetime_color();
svg_1d_plot & x_datetime_on(bool);
bool x_datetime_on();
svg_1d_plot &
x_decor(const std::string &, const std::string & = "",
        const std::string & = "");
svg_1d_plot & x_df_color(const svg_color &);
svg_color x_df_color();
svg_1d_plot & x_df_on(bool);
bool x_df_on();
svg_1d_plot & x_id_color(const svg_color &);
svg_color x_id_color();
svg_1d_plot & x_id_on(bool);
bool x_id_on();
svg_1d_plot & x_label(const std::string &);
std::string x_label();
svg_1d_plot & x_label_color(const svg_color &);
svg_color x_label_color();
svg_1d_plot & x_label_font_family(const std::string &);
const std::string & x_label_font_family();
svg_1d_plot & x_label_font_size(unsigned int);
unsigned int x_label_font_size();
svg_1d_plot & x_label_on(bool);
bool x_label_on();
svg_1d_plot & x_label_units(const std::string &);
std::string x_label_units();
svg_1d_plot & x_label_units_on(bool);
bool x_label_units_on();
svg_1d_plot & x_label_width(double);
double x_label_width();
svg_1d_plot & x_labels_strip_e0s(bool);
svg_1d_plot & x_major_grid_color(const svg_color &);
svg_color x_major_grid_color();
svg_1d_plot & x_major_grid_on(bool);
bool x_major_grid_on();
svg_1d_plot & x_major_grid_width(double);
double x_major_grid_width();
svg_1d_plot & x_major_interval(double);
double x_major_interval();
svg_1d_plot & x_major_label_rotation(rotate_style);
rotate_style x_major_label_rotation();
svg_1d_plot & x_major_labels_side(int);
int x_major_labels_side();
svg_1d_plot & x_major_tick(double);
double x_major_tick();
svg_1d_plot & x_major_tick_color(const svg_color &);
svg_color x_major_tick_color();
svg_1d_plot & x_major_tick_length(double);
double x_major_tick_length();
svg_1d_plot & x_major_tick_width(double);
double x_major_tick_width();

```

```

svg_1d_plot & x_max(double);
double x_max();
svg_1d_plot & x_min(double);
double x_min();
svg_1d_plot & x_min_ticks(int);
int x_min_ticks();
svg_1d_plot & x_minor_grid_color(const svg_color &);

svg_color x_minor_grid_color();
svg_1d_plot & x_minor_grid_on(bool);
bool x_minor_grid_on();
svg_1d_plot & x_minor_grid_width(double);
double x_minor_grid_width();
double x_minor_interval();
svg_1d_plot & x_minor_interval(double);
svg_1d_plot & x_minor_tick_color(const svg_color &);

svg_color x_minor_tick_color();
svg_1d_plot & x_minor_tick_length(double);
double x_minor_tick_length();
svg_1d_plot & x_minor_tick_width(double);
double x_minor_tick_width();
svg_1d_plot & x_num_minor_ticks(unsigned int);
unsigned int x_num_minor_ticks();
svg_1d_plot & x_order_color(const svg_color &);

svg_color x_order_color();
svg_1d_plot & x_order_on(bool);
bool x_order_on();
svg_1d_plot & x_plusminus_color(const svg_color &);

svg_color x_plusminus_color();
svg_1d_plot & x_plusminus_on(bool);
bool x_plusminus_on();
const std::string x_prefix();
svg_1d_plot & x_range(double, double);
std::pair< double, double > x_range();
const std::string x_separator();
svg_1d_plot & x_size(unsigned int);
unsigned int x_size();
svg_1d_plot & x_steps(int);
int x_steps();
const std::string x_suffix();
svg_1d_plot & x_ticks_down_on(bool);
bool x_ticks_down_on();
svg_1d_plot & x_ticks_on_window_or_axis(int);
int x_ticks_on_window_or_axis();
svg_1d_plot & x_ticks_up_on(bool);
bool x_ticks_up_on();
svg_1d_plot & x_ticks_values_color(const svg_color &);

svg_color x_ticks_values_color();
svg_1d_plot & x_ticks_values_font_family(const std::string &);

const std::string & x_ticks_values_font_family();
svg_1d_plot & x_ticks_values_font_size(unsigned int);
unsigned int x_ticks_values_font_size();
svg_1d_plot & x_ticks_values_ioflags(std::ios_base::fmtflags);
std::ios_base::fmtflags x_ticks_values_ioflags();
svg_1d_plot & x_ticks_values_precision(int);
int x_ticks_values_precision();
svg_1d_plot & x_tight(double);
double x_tight();
svg_1d_plot & x_value_font_size(unsigned int);
unsigned int x_value_font_size();
svg_1d_plot & x_value_ioflags(std::ios_base::fmtflags);
std::ios_base::fmtflags x_value_ioflags();
svg_1d_plot & x_value_precision(int);
int x_value_precision();

```

```

svg_1d_plot & x_values_color(const svg_color &);
svg_color x_values_color();
svg_1d_plot & x_values_font_family(const std::string &);
const std::string & x_values_font_family();
svg_1d_plot & x_values_font_size(unsigned int);
unsigned int x_values_font_size();
svg_1d_plot & x_values_ioflags(std::ios_base::fmtflags);
std::ios_base::fmtflags x_values_ioflags();
svg_1d_plot & x_values_on(bool);
bool x_values_on();
svg_1d_plot & x_values_precision(int);
int x_values_precision();
svg_1d_plot & x_values_rotation(rotate_style);
int x_values_rotation();
svg_1d_plot & x_with_zero(bool);
bool x_with_zero();
svg_1d_plot & y_axis_color(const svg_color &);
svg_color y_axis_color();
svg_1d_plot & y_axis_on(bool);
bool y_axis_on();
svg_1d_plot & y_label(const std::string &);
std::string y_label();
svg_1d_plot & y_label_color(const svg_color &);
svg_color y_label_color();
svg_1d_plot & y_label_units(const std::string &);
std::string y_label_units();
bool y_labels_strip_e0s();
double y_minor_interval();
unsigned int y_size();
svg_1d_plot & y_size(unsigned int);
};

```

Description

`axis_plot_frame.hpp` contains functions common to 1 and 2-D.

Several versions of the function `plot` are provided to allow data to be in different sources, and to allow either all data in a container or just a sub-range to be plotted.

See Also:

`svg_2d_plot.hpp` for the 2-D version.

svg_1d_plot public member functions

1. `bool autoscale();`

Returns: true if to use autoscale values autoscaling for X-axis.

2. `svg_1d_plot & autoscale(bool b);`

Set true if to use autoscale values for X-axis.

3. `svg_1d_plot & autoscale_check_limits(bool b);`

Set true to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed if user can be sure all values are 'inside limits'.

4. `bool autoscale_check_limits();`

Returns: true if to check that values used for autoscaling are within limits.

5. `svg_1d_plot & autoscale_plusminus(double);`

Set how many std_dev or standard deviations to allow for ellipses when autoscaling.

6. `double autoscale_plusminus();`

Returns: How many std_dev or standard deviations allowed for ellipses when autoscaling.

7. `svg_1d_plot & axes_on(bool is);`

Set true if to draw **both** x and y axes (note plural axes).

8. `bool axes_on();`

Returns: true if to draw **both** x and y axis on.

9. `svg_1d_plot & background_border_color(const svg_color & col);`

Set plot background border color.

10. `svg_color background_border_color();`

Returns: plot background border color.

11. `svg_1d_plot & background_border_width(double w);`

Set plot background border width.

12. `double background_border_width();`

Returns: Plot background border width.

13. `svg_color background_color();`

Returns: Plot background color.

14. `svg_1d_plot & background_color(const svg_color & col);`

Set plot background color.

15. `svg_1d_plot & boost_license_on(bool l);`

Set true if the Boost license conditions should be included in the SVG document.

16. `bool boost_license_on();`

Returns: true if the Boost license conditions should be included in the SVG document.

17. `svg_1d_plot & confidence(double);`

Set confidence alpha for display of confidence intervals (default 0.05 for 95%).

18. `double confidence();`

Returns: Confidence alpha for display of confidence intervals (default 0.05 for 95%).

19. `svg_1d_plot & coord_precision(int digits);`

Precision of SVG coordinates in decimal digits (default 3). 3 decimal digits precision is sufficient for small images. 4 or 5 decimal digits precision will give higher quality plots, especially for larger images, at the expense of larger .svg files, particularly if there are very many data points.

20. `int coord_precision();`

Returns: precision of SVG coordinates in decimal digits.

21. `svg_1d_plot & copyright_date(const std::string d);`

Writes copyright date to the SVG document. and as metadata:<meta name="date" content="2007" />

22. `const std::string copyright_date();`

Returns: SVG document copyright_date.

23. `svg_1d_plot & copyright_holder(const std::string d);`

Writes copyright_holder metadata to the SVG document (for header as) /and as metadata:<meta name="copyright" content="Paul A. Bristow" />

24. `const std::string copyright_holder();`

Returns: SVG document copyright holder.

25. `svg_1d_plot & data_lines_width(double width);`

Set the width of lines joining data points.

26. `double data_lines_width();`

Returns: the width of lines joining data points.

27. `svg_1d_plot & derived();`

Uses Curiously Recurring Template Pattern to allow 1D and 2D to reuse common code. See http://en.wikipedia.org/wiki/Curiously_Recurring_Template_Pattern.

28. `const svg_1d_plot & derived() const;`

const version of derived()

29. `svg_1d_plot & description(const std::string d);`

Writes description to the document for header as.

`<desc> My Description </desc>.`

30. `const std::string & description();`

Returns: Description of the document for header as`<desc> My description </desc>.`

31. `svg_1d_plot & document_title(const std::string d);`

Set document title to the document for header as.

`<title> My Title </title>.`

32. `std::string document_title();`

Returns: Document title to the document for header as`<title> My Title </title>.`

33. `svg_1d_plot &
draw_line(double x1, double y1, double x2, double y2,
const svg_color & col = black);`

Annotate plot with a line from SVG Coordinates X1, Y1 to X2, Y2. (Default color black). Note NOT the data values. See draw_plot_line if want to use user coordinates.

34. `svg_1d_plot &
draw_note(double x, double y, std::string note, rotate_style rot = horizontal,
align_style al = center_align, const svg_color & = black,
text_style & tsty = no_style);`

Annotate plot with a text string (perhaps including Unicode), putting note at SVG Coordinates X, Y.

Defaults color black, rotation horizontal and align = center_align Using center_align is recommended as it will ensure that will center correctly (even if original string is made much longer because it contains Unicode, for example Greek or math symbols, taking about 6 characters per symbol) because the render engine does the centering.

35. `svg_1d_plot &
draw_plot_curve(double x1, double y1, double x2, double y2, double x3,
double y3, const svg_color & col = black);`

Annotate plot with a line from user's Cartesian Coordinates X1, Y1 via X2, Y2 to X3, Y3.

For example, -10, -10, +10, +10, Default color black.

36. `svg_1d_plot &
draw_plot_line(double x1, double y1, double x2, double y2,
const svg_color & col = black);`

Annotate plot with a line from user's Cartesian Coordinates X1, Y1 to X2, Y2.

For example, -10, -10, +10, +10, Default color black.

37. `svg_1d_plot & image_border_margin(double w);`

Set the margin around the plot window border (svg units, default pixels).

This prevents the plot window getting too close to other elements of the plot.

38. `double image_border_margin();`

Returns: the margin around the plot window border (svg units, default pixels).

39. `svg_1d_plot & image_border_width(double w);`

Set the svg image border width (svg units, default pixels).

40. `double image_border_width();`

Returns: the svg image border width (svg units, default pixels).

41. `unsigned int image_x_size();`

Obselete - deprecated use x_size()

42. `svg_1d_plot & image_x_size(unsigned int i);`

Obselete - deprecated - use x_size().

43. `unsigned int image_y_size();`

Obselete - deprecated - use y_size()

44. `svg_1d_plot & image_y_size(unsigned int i);`

Obselete - deprecated - use y_size()

45. `svg_1d_plot & legend_background_color(const svg_color & col);`

Set the background fill color of the legend box.

46. `svg_color legend_background_color();`

Returns: the background fill color of the legend box.

47. `svg_1d_plot & legend_border_color(const svg_color & col);`

Set the border stroke color of the legend box.

48. `svg_color legend_border_color();`

Returns: the border stroke color of the legend box.

49. `const std::pair< double, double > legend_bottom_right();`

Returns: SVG coordinate (default pixels) of bottom right of legend box.

50. `bool legend_box_fill_on();`

Returns: true if legend box has a background fill color.

51. `svg_1d_plot & legend_color(const svg_color & col);`

Set the color of the title of the legend.

52. `svg_color legend_color();`

Returns: the color of the title of the legend.

53. `svg_1d_plot & legend_font_family(const std::string & family);`

Set the font family for the legend title.

54. `const std::string & legend_font_family();`

Returns: the font family for the legend title.

55. `svg_1d_plot & legend_font_weight(const std::string & weight);`

Set the font weight for the legend title.

56. `const std::string & legend_font_weight();`

Returns: Font weight for the legend title.

57. `svg_1d_plot & legend_header_font_size(int size);`

Set legend header font size (svg units, default pixels).

58. `int legend_header_font_size();`

Returns: legend header font size (svg units, default pixels).

59. `svg_1d_plot & legend_lines(bool is);`

Set true if legend should include samples of the lines joining data points. This allows different series of data points to be distinguished by different color and/or width. This is especially useful to show plots of different functions and/or different parameters in different colors.

60. `bool legend_lines();`

Returns: true if legend should include samples of the lines joining data points.

61. `svg_1d_plot & legend_on(bool cmd);`

Set true if a legend is wanted.

62. `bool legend_on();`

Returns: true if a legend is wanted.

63. `bool legend_outside();`

Returns: if the legend should be outside the plot area.

64. `svg_1d_plot & legend_place(legend_places l);`

Set the position of the legend.,

See Also:

`boost::svg::legend_places`

65. `legend_places legend_place();`

See Also:

`boost::svg::legend_places`

Returns: the position of the legend,

66. `svg_1d_plot & legend_title(const std::string title);`

Set the title for the legend.

67. `const std::string legend_title();`

Returns: Title for the legend.

68. `svg_1d_plot & legend_title_font_size(unsigned int size);`

Returns: Font family for the legend title.

69. `unsigned int legend_title_font_size();`

Returns: Font size for the legend title (svg units, default pixels).

70. `svg_1d_plot & legend_top_left(double x, double y);`

Set position of top left of legend box (svg coordinates, default pixels). (Bottom right is controlled by contents, so the user cannot set it).

71. `const std::pair<double, double> legend_top_left();`

Returns: SVG coordinate (default pixels) of top left of legend box.

72. `svg_1d_plot & legend_width(double width);`

Set the width for the legend box.

73. `double legend_width();`

Returns: Width for the legend box.

74. `svg_1d_plot & license(std::string repro = "permits", std::string distrib = "permits", std::string attrib = "requires", std::string commercial = "permits", std::string derivative = "permits");`

Set license conditions for reproduction, attribution, commercial use, and derivative works, usually "permits", "requires", or "prohibits", and set license_on == true.

75. `const std::string license_attribution();`

Returns: SVG document attribution license conditions, usually "permits", "requires", or "prohibits".

76. `const std::string license_commercialuse();`

Returns: SVG document commercial use license conditions, usually "permits", "requires", or "prohibits".

77. `const std::string license_distribution();`

Returns: SVG document distribution license conditions, usually "permits", "requires", or "prohibits".

78. `svg_1d_plot & license_on(bool l);`

Set if license conditions should be included in the SVG document.

79. `bool license_on();`

Returns: true if license conditions should be included in the SVG document.

80. `const std::string license_reproduction();`

Returns: SVG document reproduction license conditions, usually "permits", "requires", or "prohibits".

81. `svg_1d_plot & limit_color(const svg_color &);`

Set the color for 'at limit' point stroke color.

82. `svg_color limit_color();`

Returns: the color for the 'at limit' point stroke color.

83. `svg_1d_plot & limit_fill_color(const svg_color &);`

Set the color for 'at limit' point fill color.

84. `svg_color limit_fill_color();`

Returns: the color for the 'at limit' point fill color.

85. `svg_1d_plot & one_sd_color(const svg_color &);`

Set the color for the one standard deviation (~67% confidence) ellipse fill.

86. `svg_color one_sd_color();`

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

87. `template<typename T>`
`svg_1d_plot_series &`
`plot(const T & container, const std::string & title = "");`

Example:

Add a data series to the plot (by default, converting to unc doubles), with optional title.



Note

This version assumes that **ALL** the data values in the container are used.

```
std::vector<float> my_data; // my container.  

my_data.push_back(2.f); // Fill container with some data.  

my_data.push_back(3.f);  

my_data.push_back(4.f);  

my_1d_plot.plot(my_data, "All data in my container"); // Plot all data in container.
```

Parameters: `container` Container (for example vector) for the data to be added to the plot.
`title` Optional title for the data series (default none).

Template Parameters: `T` Floating-point type of the data (`T` must be convertible to `double`).

Returns: Reference to data series just added (to make chainable).

88. `template<typename T>`
`svg_1d_plot_series &`
`plot(const T & begin, const T & end, const std::string & title = "");`

Add a data series to the plot (by default, converting to unc doubles), with optional title.



Note

This version permits a partial range of the container, from `begin` to `end`, to be used.

Example:

```
my_1d_plot.plot(my_data.begin(), my_data.end(), "My container"); // Whole container of data values.
my_1d_plot.plot(&my_data[1], &my_data[4], "my_data 1 to 4"); // Add part of data series
```



Warning

last == end which is one past the last, so this only does 1, 2 & 3 - **not 4!**

Parameters:	begin Iterator to 1st data item in container. end Iterator to one-beyond-end of data in container. title Optional title for the plot (default is no title).
Template Parameters:	T floating-point type of the data (T must be convertible to double).
Returns:	Reference to the data series just added.

89. `template<typename T, typename U>
svg_1d_plot_series &
plot(const T & container, const std::string & title = "",
U functor = unspecified);`

Add a data series to the plot, with optional title.



Note

This version of plot includes a functor, allowing other than just convert data values to double (the default).

Parameters:	container Container for data (for example std::vector) that contains the data to be added to the plot. functor Custom functor to convert data value to double. title Optional title for the plot (default is no title).
Template Parameters:	T floating-point type of the data (which must be convertible to double). U functor floating-point type (default is double_1d_convert).
Returns:	a reference to data series just added (to make chainable).

90. `template<typename T, typename U>
svg_1d_plot_series &
plot(const T & begin, const T & end, const std::string & title = "",
U functor = unspecified);`

Add a data series to the plot, with optional title. (Version with custom functor, rather than to double).



Note

This version permits a **partial** range, within begin to end, of the container to be used.

Parameters:	begin Iterator to 1st data item in container. end Iterator to one-beyond-end of data in container. functor Custom functor. title Optional title for the plot (default is no title).
Template Parameters:	T Floating-point type of the data (which must be convertible to double). U Functor floating-point type (default is double_1d_convert).
Returns:	a reference to data series just added (to make chainable).

91. `svg_1d_plot & plot_background_color(const svg_color & col);`

Set the fill color of the plot window background.

92 `svg_color plot_background_color();`

Returns: the fill color of the plot window background.

93. `svg_1d_plot & plot_border_color(const svg_color & col);`

Set the color for the plot window background.

94. `svg_color plot_border_color();`

Returns: the color for the plot window background.

95. `svg_1d_plot & plot_border_width(double w);`

Set the margin around the plot window border (svg units, default pixels).

This prevents the plot window getting too close to other elements of the plot.

96. `double plot_border_width();`

Returns: the width for the plot window border (svg units, default pixels).

97. `svg_1d_plot & plot_window_on(bool cmd);`

Set true if a plot window is wanted (or false if the whole image is to be used).

98. `bool plot_window_on();`

Returns: true if a plot window is wanted (or false if the whole image is to be used).

99. `svg_1d_plot & plot_window_x(double min_x, double max_x);`

Set the minimum and maximum (cartesian data units) for the plot window X axis. This is normally calculated from other plot values.

100. `std::pair< double, double > plot_window_x();`

Returns: both the left and right (X axis) of the plot window.

101. `double plot_window_x_left();`

Returns: left of the plot window.

102. `double plot_window_x_right();`

Returns: right of the plot window.

103. `svg_1d_plot & plot_window_y(double min_y, double max_y);`

Set the minimum and maximum (cartesian data units) for the plot window Y axis. This is normally calculated from other plot values.

14 `std::pair< double, double > plot_window_y();`

Returns: both the top and bottom (Y axis) of the plot window.

15 `double plot_window_y_bottom();`

Returns: top of the plot window.

16 `double plot_window_y_top();`

Returns: top of the plot window.

17 `svg_1d_plot & size(unsigned int x, unsigned int y);`

Set SVG image size (SVG units, default pixels).

18 `std::pair< double, double > size();`

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

19 `svg_1d_plot & three_sd_color(const svg_color &);`

Set the color for three standard deviation (~99% confidence) ellipse fill.

10 `svg_color three_sd_color();`

Returns: Color for three standard deviation (~99% confidence) ellipse fill.

11 `svg_1d_plot & title(const std::string title);`

Set a title for plot. The string may include Unicode for greek letter and symbols. **example:** A title that includes a greek omega and degree symbols:

```
my_plot.title("Plot of &#x3A9; function (&#x00B0;C)");
```

Unicode symbols are at <http://unicode.org/charts/symbols.html>.

12 `const std::string title();`

Returns: Title for plot (whose string may include Unicode for greek letter and symbols).

13 `svg_1d_plot & title_color(const svg_color & col);`

Set the color of any title of the plot.

14 `svg_color title_color();`

Returns: the color of any title of the plot.

15 `svg_1d_plot & title_font_alignment(align_style alignment);`

Set the alignment for the title.

16 `align_style title_font_alignment();`

Returns: the alignment for the title.

17 `svg_1d_plot & title_font_decoration(const std::string & decoration);`

Set the font decoration for the title (default normal, or underline, overline or strike-thru).

18 `const std::string & title_font_decoration();`

Returns: Font decoration for the title (default normal, or underline, overline or strike-thru).

19 `svg_1d_plot & title_font_family(const std::string & family);`

Set the font family for the title (for example: `.title_font_family("Lucida Sans Unicode");`)

20 `const std::string & title_font_family();`

Returns: Font family for the title.

21 `svg_1d_plot & title_font_rotation(rotate_style rotate);`

Set the rotation for the title font (degrees, 0 to 360 in steps using `rotate_style`, for example horizontal, uphill...)

22 `int title_font_rotation();`

Returns: the rotation for the title font (degrees).

23 `svg_1d_plot & title_font_size(unsigned int i);`

Sets the font size for the title (SVG units, default pixels).

24 `unsigned int title_font_size();`

Returns: Font size for the title (SVG units, default pixels).

25 `svg_1d_plot & title_font_stretch(const std::string & stretch);`

Set the font stretch for the title (default normal), wider or narrow.

26 `const std::string & title_font_stretch();`

Returns: Font stretch for the title.

27 `svg_1d_plot & title_font_style(const std::string & style);`

Set the font style for the title (default normal).

18 `const std::string & title_font_style();`

Returns: Font style for the title (default normal).

19 `svg_1d_plot & title_font_weight(const std::string & weight);`

Set the font weight for the title (default normal).

20 `const std::string & title_font_weight();`

Returns: Font weight for the title.

21 `svg_1d_plot & title_on(bool cmd);`

If set true, show a title for the plot. Note: is set true by setting a title.

22 `bool title_on();`

Returns: true if will show a title for the plot.

23 `svg_1d_plot & two_sd_color(const svg_color &);`

Set the color for two standard deviation (~95% confidence) ellipse fill.

24 `svg_color two_sd_color();`

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

25 `svg_1d_plot & write(const std::string & file);`

Write SVG image to the specified file, providing the suffix .svg if no suffix given.

`write()` has two versions: to an `ostream` and to a file. The stream version first clears all unnecessary data from the graph, builds the document tree, and then calls the `write` function for the root document node, which calls all other nodes through the Visitor pattern.



Note

This file version opens an `ostream`, and calls the `ostream` version of `write`.

Parameters: `file` Filename to write.

Returns: `*this` to make chainable.

26 `svg_1d_plot & write(std::ostream & s_out);`

Write SVG image to the specified `std::ostream`.



Note

This function also is used by the write to file function.

The default stream precision of 6 decimal digits is probably excessive. 4.1 Basic data types, integer or float in decimal or scientific (using E format). If image size is under 1000 x 1000, the SVG plot default precision of 3 is probably sufficient. This reduces .svg file sizes significantly for curves represented with many data points. For example, if a curve is shown using 100 points, reducing to coord_precision(3) from default of 6 will reduce file size by 300 bytes.

Parameters: `s_out` `std::ostream` to write.

Returns: `*this` to make chainable.

17 `svg_1d_plot & x_addlimits_color(const svg_color & col);`

Set the color of X confidence limits of value, for example, the color in "<1.23, 1.45>".

18 `svg_color x_addlimits_color();`

Returns: the color of X confidence limits of value, for example, the color of "<1.23, 1.45>".

19 `svg_1d_plot & x_addlimits_on(bool b);`

Set if to append confidence limits to data point X values near data points markers.

20 `bool x_addlimits_on();`

Returns: true if to append confidence limits estimate to data point X values near data points markers.

21 `double x_auto_max_value();`

Returns: X-axis maximum value computed by autoscale.

22 `double x_auto_min_value();`

Returns: X-axis minimum value computed by autoscale.

23 `double x_auto_tick_interval();`

Returns: the X-axis major tick interval computed by autoscale.

24 `int x_auto_ticks();`

Returns: the X-axis number of major ticks computed by autoscale.

25 `bool x_autoscale();`

Returns: true if to use autoscale value for X-axis.

26 `svg_1d_plot & x_autoscale(bool b);`

Set true if to use autoscale values for X-axis.

17 `svg_1d_plot & x_autoscale(std::pair< double, double > p);`

autoscale X axis using a pair of doubles.

18 `svg_1d_plot & x_autoscale(const T & container);`

<

19 `svg_1d_plot & x_autoscale(const T & begin, const T & end);`

<

20 `svg_1d_plot & x_axis_color(const svg_color & col);`

Set the color of the X-axis line.

21 `svg_color x_axis_color();`

Returns: the color of the X-axis line.

22 `svg_1d_plot & x_axis_label_color(const svg_color & col);`

Set X axis label color, for example, red.

23 `svg_color x_axis_label_color();`

Returns: X axis label color. <X-axis ticks values label style.

24 `svg_1d_plot & x_axis_on(bool is);`

If set true, draw a horizontal X-axis line.

25 `bool x_axis_on();`

Returns: true if will draw a horizontal X-axis line.

26 `const std::string x_axis_position();`

Returns: the position (or intersection with Y-axis) of the X-axis.

27 `svg_1d_plot & x_axis_vertical(double fraction);`

Set vertical position of X-axis for 1D as fraction of plot window.

28 `bool x_axis_vertical();`

Returns: vertical position of X-axis for 1D as fraction of plot window.

19 `svg_1d_plot & x_axis_width(double width);`

Set the width of X-axis lines.

20 `double x_axis_width();`

Returns: the width of X-axis lines.

21 `svg_1d_plot & x_datetime_color(const svg_color & col);`

Set the color of X date time , for example, the color of text in "".

22 `svg_color x_datetime_color();`

Returns: the color of X date time, for example, the color of text in "".

23 `svg_1d_plot & x_datetime_on(bool b);`

Set true if to append date time to data point X values near data points markers.

24 `bool x_datetime_on();`

Returns: true if to append an date time to data point X values near data points markers.

25 `svg_1d_plot & x_decor(const std::string & pre, const std::string & sep = "",
const std::string & suf = "");`

Set prefix, separator and suffix together for x_ values. Note if you want a space, you must use a Unicode space " ", for example, ", " rather than ASCII space", ". If 1st char in separator == , then Y values and info will be on a newline below.

26 `svg_1d_plot & x_df_color(const svg_color & col);`

Set the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

27 `svg_color x_df_color();`

Returns: the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

28 `svg_1d_plot & x_df_on(bool b);`

Set true if to append a degrees of freedom estimate to data point X values near data points markers.

29 `bool x_df_on();`

Returns: true if to append a degrees of freedom estimate to data point X values near data points markers.

30 `svg_1d_plot & x_id_color(const svg_color & col);`

Set the color of X id or name, for example, the color of text in "my_id".

I1 `svg_color x_id_color();`

Returns: the color of X X id or name, for example, the color of text in "my_id".

I2 `svg_1d_plot & x_id_on(bool b);`

Set true if to append append an ID or name to data point X values near data points markers.

I3 `bool x_id_on();`

Returns: true if to append an ID or name to data point X values near data points markers.

I4 `svg_1d_plot & x_label(const std::string & str);`

Set the text to label the X-axis (and set x_label_on(true)).

I5 `std::string x_label();`

Returns: the text to label the X-axis.

I6 `svg_1d_plot & x_label_color(const svg_color & col);`

Returns: the color of the Y-axis line.

I7 `svg_color x_label_color();`

Returns: the color of X-axis label (including any units).

I8 `svg_1d_plot & x_label_font_family(const std::string & family);`

Set X tick value label font family. Available fonts depend on the program rendering the SVG XML, usually a browser. The default font (usually "default_font") is used if a render program does not provide the font specified. These are probably usable:

```
"arial", "impact", "courier", "lucida console", "Lucida sans unicode", "verdana", "calibri", "century",
"lucida calligraphy", "tahoma", "vivaldi", "informal roman", "lucida handwriting", "lucida bright", "helvetica"
```

I9 `const std::string & x_label_font_family();`

Returns: X tick value label font family.

I10 `svg_1d_plot & x_label_font_size(unsigned int i);`

Set X axis label font size (svg units, default pixels).

I11 `unsigned int x_label_font_size();`

Returns: X axis label font size (svg units, default pixels).

I12 `svg_1d_plot & x_label_on(bool cmd);`

Returns: true if X major ticks should mark downwards.

18 `bool x_label_on();`

Set true if want to show X-axis label text. Also switched on by setting label text. (on the assumption that if label text is set, display is also wanted, but can be switched off if **not** required).

19 `svg_1d_plot & x_label_units(const std::string & str);`

Set the text to add units to the X-axis label.

20 `std::string x_label_units();`

Returns: the text to add units to the X-axis label. <The label will only be shown if `x_label_on() == true`.

21 `svg_1d_plot & x_label_units_on(bool cmd);`

Set true if want X axis label to include units (as well as label like "length"). <.

See Also:

`x_label_units` which also sets true.

22 `bool x_label_units_on();`

Set true if want X axis label to include units (as well as label like "length").

23 `svg_1d_plot & x_label_width(double width);`

Set the width (boldness) of X-axis label (including any units). (not recommended until browsers implement better).

24 `double x_label_width();`

Returns: the width (boldness) of X-axis label (including any units).

25 `svg_1d_plot & x_labels_strip_e0s(bool cmd);`

Set if to strip redundant zeros, signs and exponents, for example, reducing "1.2e+000" to "1.2" This markedly reduces visual clutter, and is the default.

26 `svg_1d_plot & x_major_grid_color(const svg_color & col);`

Set the color of X-axis major grid lines.

27 `svg_color x_major_grid_color();`

Set the color of X-axis major grid lines.

28 `svg_1d_plot & x_major_grid_on(bool is);`

If set true, will include a major X-axis grid.

14 `bool x_major_grid_on();`

Returns: true if will include a major X-axis grid.

15 `svg_1d_plot & x_major_grid_width(double w);`

Set the width of X-axis major grid lines.

16 `double x_major_grid_width();`

Returns: the color of X-axis major grid lines.

17 `svg_1d_plot & x_major_interval(double inter);`

Set the interval between X-axis major ticks.

18 `double x_major_interval();`

Returns: the interval between X-axis major ticks.

19 `svg_1d_plot & x_major_label_rotation(rotate_style rot);`

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

`rotate_style`

20 `rotate_style x_major_label_rotation();`

See Also:

`rotate_style`

Returns: rotation for X ticks major value labels.

21 `svg_1d_plot & x_major_labels_side(int side);`

Position of labels for X major ticks on horizontal X axis line.

Parameters: `side` > 0 X tick value labels to left of Y axis line (default), 0 (false) no major X tick value labels on Y axis, 0 X tick labels to right of Y axis line.

22 `int x_major_labels_side();`

Returns: the side for X ticks major value labels.

23 `svg_1d_plot & x_major_tick(double d);`

Set interval (Cartesian units) between major ticks.

24 `double x_major_tick();`

Returns: interval (Cartesian units) between major ticks.

25 `svg_1d_plot & x_major_tick_color(const svg_color & col);`

Set the color of X-axis major ticks.

26 `svg_color x_major_tick_color();`

Returns: the color of X-axis major ticks.

27 `svg_1d_plot & x_major_tick_length(double length);`

Set length of X major ticks (SVG units, default pixels).

28 `double x_major_tick_length();`

Set length of X major ticks (SVG units, default pixels).

29 `svg_1d_plot & x_major_tick_width(double width);`

Set width of X major ticks (SVG units, default pixels).

30 `double x_major_tick_width();`

Set width of X major ticks (SVG units, default pixels).

31 `svg_1d_plot & x_max(double x);`

Set the maximum value on the X-axis.

32 `double x_max();`

autoscale set & get parameters, <Note: all these *MUST* precede x_autoscale(data) call.

Returns: the maximum value on the X-axis.

33 `svg_1d_plot & x_min(double min_x);`

Set the minimum value on the X-axis.

34 `double x_min();`

Returns: the minimum value on the X-axis.

35 `svg_1d_plot & x_min_ticks(int min_ticks);`

Set X-axis autoscale to include at least minimum number of ticks (default = 6).

36 `int x_min_ticks();`

Returns: X-axis autoscale minimum number of ticks.

27 `svg_1d_plot & x_minor_grid_color(const svg_color & col);`

Set the color of X-axis minor grid lines.

28 `svg_color x_minor_grid_color();`

Returns: the color of X-axis minor grid lines.

29 `svg_1d_plot & x_minor_grid_on(bool is);`

If set true, will include a minor X-axis grid.

30 `bool x_minor_grid_on();`

Returns: true if will include a major X-axis grid.

31 `svg_1d_plot & x_minor_grid_width(double w);`

Set the width of X-axis minor grid lines.

32 `double x_minor_grid_width();`

Returns: the width of X-axis minor grid lines.

33 `double x_minor_interval();`

Returns: interval between X minor ticks.

34 `svg_1d_plot & x_minor_interval(double interval);`

Set interval between X-axis minor ticks.

35 `svg_1d_plot & x_minor_tick_color(const svg_color & col);`

Set the color of X-axis minor ticks.

36 `svg_color x_minor_tick_color();`

Returns: the color of X-axis minor ticks.

37 `svg_1d_plot & x_minor_tick_length(double length);`

Set length of X minor ticks (SVG units, default pixels).

38 `double x_minor_tick_length();`

Returns: length of X minor ticks (SVG units, default pixels).

39 `svg_1d_plot & x_minor_tick_width(double width);`

Set width of X minor ticks (SVG units, default pixels).

20 `double x_minor_tick_width();`

Returns: width of X minor ticks (SVG units, default pixels).

21 `svg_1d_plot & x_num_minor_ticks(unsigned int num);`

Set number of X-axis minor ticks between major ticks.

22 `unsigned int x_num_minor_ticks();`

Returns: number of X-axis minor ticks between major ticks.

23 `svg_1d_plot & x_order_color(const svg_color & col);`

Set the color of X order #, for example, the color of #42.

24 `svg_color x_order_color();`

Returns: the color of X order #, for example, the color of #42.

25 `svg_1d_plot & x_order_on(bool b);`

Set true if to append append an order # to data point X values near data points markers.

26 `bool x_order_on();`

Returns: true if to append an order # to data point X values near data points markers.

27 `svg_1d_plot & x_plusminus_color(const svg_color & col);`

Set the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

28 `svg_color x_plusminus_color();`

Returns: the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

29 `svg_1d_plot & x_plusminus_on(bool b);`

Set if to append std_dev estimate to data point X values near data points markers.

30 `bool x_plusminus_on();`

Returns: true if to append std_dev estimate to data point X values near data points markers.

31 `const std::string x_prefix();`

Returns: the prefix.

22 `svg_1d_plot & x_range(double min_x, double max_x);`

Set the range of values on the X-axis. The minimum and maximum values must be finite and not too near to the minima or maxima that can be represented by floating point double, std::numeric_limits<double>::min() or std::numeric_limits<double>::max(), and the range must not be too small.

23 `std::pair< double, double > x_range();`

Returns: the range of values on the X-axis.

24 `const std::string x_separator();`

Returns: the separator, perhaps including Unicode.

25 `svg_1d_plot & x_size(unsigned int i);`

Set SVG image X-axis size (SVG units, default pixels).

26 `unsigned int x_size();`

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

27 `svg_1d_plot & x_steps(int steps);`

Set autoscale to set ticks in steps multiples of:

2,4,6,8,10, if 2

or 1,5,10 if 5

or 2,5,10 if 10.

default = 0 (none).



Note

: Must precede x_autoscale(data) call).

28 `int x_steps();`

Returns: autoscale to set ticks in steps.

29 `const std::string x_suffix();`

Returns: the suffix (only used if separator != "")

30 `svg_1d_plot & x_ticks_down_on(bool cmd);`

Set true if Y major ticks should mark upwards.

31 `bool x_ticks_down_on();`

Returns: true if Y major ticks should mark upwards.

22 `svg_1d_plot & x_ticks_on_window_or_axis(int side);`

Set position of X ticks on window or axis.

Parameters: side -1 X ticks on bottom of plot window, 0 X ticks on X-axis horizontal line, +1 X ticks top of plot window.

23 `int x_ticks_on_window_or_axis();`

Returns: true if X axis ticks wanted on the window (rather than on axis).
<-1 bottom of plot window, 0 on horizontal X axis , +1 top of plot window.

24 `svg_1d_plot & x_ticks_up_on(bool cmd);`

Set true if X major ticks should mark upwards.

25 `bool x_ticks_up_on();`

Returns: true if X major ticks should mark upwards.

26 `svg_1d_plot & x_ticks_values_color(const svg_color & col);`

Set X axis tick value label color.

27 `svg_color x_ticks_values_color();`

Returns: X-axis ticks value label color.

28 `svg_1d_plot & x_ticks_values_font_family(const std::string & family);`

Set X ticks value label font family. Available fonts depend on the program rendering the SVG XML, usually a browser. The default font (usually "default_font") is used if a render program does not provide the font specified. These are probably usable:

"arial", "impact", "courier", "lucida console", "Lucida sans unicode", "verdana", "calibri", "century",
"lucida calligraphy", "tahoma", "vivaldi", "informal roman", "lucida handwriting", "lucida bright", "helvetica"

29 `const std::string & x_ticks_values_font_family();`

Returns: X ticks value label font family.

30 `svg_1d_plot & x_ticks_values_font_size(unsigned int i);`

Set X ticks value label font size (svg units, default pixels).

31 `unsigned int x_ticks_values_font_size();`

Set X ticks value label font size (svg units, default pixels).

32 `svg_1d_plot & x_ticks_values_ioflags(std::ios_base::fmtflags f);`

Set iostream format flags of data point X values near data points markers.

23 `std::ios_base::fmtflags x_ticks_values_ioflags();`

Returns: ostream format flags of data point X values near data points markers.

24 `svg_1d_plot & x_ticks_values_precision(int p);`

Set ostream decimal digits precision of data point X values near data points markers.

25 `int x_ticks_values_precision();`

Returns: ostream decimal digits precision of data point X values near data points markers.

26 `svg_1d_plot & x_tight(double tight);`

Set tolerance to autoscale to permit data points slightly outside both end ticks.

27 `double x_tight();`

Returns: tolerance given to autoscale to permit data points slightly outside both end ticks. <Get results of autoscaling.

28 `svg_1d_plot & x_value_font_size(unsigned int i);`

Set X tick value label font size (svg units, default pixels).

29 `unsigned int x_value_font_size();`

Returns: X tick value label font size (svg units, default pixels).

30 `svg_1d_plot & x_value_ioflags(std::ios_base::fmtflags flags);`

Set ostream std::ios::fmtflags for X value label (default decimal == 0X201). Mainly useful for changing to scientific, fixed or hexadecimal format. For example:

```
myplot.x_value_ioflags(std::ios::dec | std::ios::scientific)
```

31 `std::ios_base::fmtflags x_value_ioflags();`

Returns: stream std::ios::fmtflags for control of format of X value labels.

32 `svg_1d_plot & x_value_precision(int digits);`

Set precision of X-tick label values in decimal digits (default 3). 3 decimal digits precision is sufficient for small images. 4 or 5 decimal digits precision will give more cluttered plots. If the range of labels is very small, then more digits will be essential.

33 `int x_value_precision();`

Returns: Precision of X-tick label values in decimal digits

34 `svg_1d_plot & x_values_color(const svg_color & col);`

Set the color of data point X values near data points markers.

25 `svg_color x_values_color();`

Returns: the color of data point X values near data points markers.

26 `svg_1d_plot & x_values_font_family(const std::string & family);`

Set font family of data point X values near data points markers.

27 `const std::string & x_values_font_family();`

Returns: font family of data point X values near data points markers.

28 `svg_1d_plot & x_values_font_size(unsigned int i);`

Set font size of data point X values near data points markers.

29 `unsigned int x_values_font_size();`

Returns: font size of data point X values near data points markers.

30 `svg_1d_plot & x_values_ioflags(std::ios_base::fmtflags f);`

Set istream format flags of data point X values near data points markers.

31 `std::ios_base::fmtflags x_values_ioflags();`

Returns: istream format flags of data point X values near data points markers.

32 `svg_1d_plot & x_values_on(bool b);`

Set true to show data point values near data points markers.

33 `bool x_values_on();`

Returns: true if to show data point values near data points markers.

34 `svg_1d_plot & x_values_precision(int p);`

Set ostream decimal digits precision of data point X values near data points markers.

35 `int x_values_precision();`

Returns: ostream decimal digits precision of data point X values near data points markers.

36 `svg_1d_plot & x_values_rotation(rotate_style rotate);`

Returns: the rotation (rotate_style) of data point X values near data points markers.

27 `int x_values_rotation();`

Set the rotation (rotate_style) of data point X values near data points markers.

28 `svg_1d_plot & x_with_zero(bool b);`

Set X-axis autoscale to include zero (default = false).

29 `bool x_with_zero();`

Returns: true if X-axis autoscale to include zero (default = false).

30 `svg_1d_plot & y_axis_color(const svg_color & col);`

Set the color of the Y-axis line.

31 `svg_color y_axis_color();`

Returns: the color of the Y-axis line.

32 `svg_1d_plot & y_axis_on(bool is);`

If set true, draw a vertical Y-axis line.

33 `bool y_axis_on();`

Returns: true if will draw a horizontal X-axis line.

34 `svg_1d_plot & y_label(const std::string & str);`

Set the text for the Y-axis label (and set y_label_on(true)).

35 `std::string y_label();`

Returns: the text for the Y-axis label. < The label will only be shown if y_label_on() == true.

36 `svg_1d_plot & y_label_color(const svg_color & col);`

Set the color of Y-axis label (including any units).

37 `svg_color y_label_color();`

Returns: the color of Y-axis label (including any units).

38 `svg_1d_plot & y_label_units(const std::string & str);`

Set the text to add units to the Y-axis label.

39 `std::string y_label_units();`

Returns: the text to add units to the X-axis label.

30 `bool y_labels_strip_e0s();`

Returns: if to strip redundant zeros, signs and exponents, for example, reducing "1.2e+000" to "1.2"

31 `double y_minor_interval();`

Returns: interval between Y minor ticks.

32 `unsigned int y_size();`

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

33 `svg_1d_plot & y_size(unsigned int i);`

Set SVG image Y-axis size (SVG units, default pixels).

Class `svg_1d_plot_series`

`boost::svg::svg_1d_plot_series` — Holds a series of data values (points) to be plotted.

Synopsis

```
// In header: <boost/svg_plot/svg_1d_plot.hpp>

class svg_1d_plot_series {
public:
    // construct/copy/destruct
    template<typename C> svg_1d_plot_series(C, C, const std::string & = "");

    // public member functions
    svg_1d_plot_series & bezier_on(bool);
    bool bezier_on();
    svg_1d_plot_series & fill_color(const svg_color &);

    svg_color fill_color();
    svg_1d_plot_series & limit_point_color(const svg_color &);

    svg_1d_plot_series & line_color(const svg_color &);

    svg_1d_plot_series & line_on(bool);
    bool line_on();

    svg_1d_plot_series & line_width(double);
    double line_width();

    size_t series_count();
    size_t series_limits_count();
    svg_1d_plot_series & shape(point_shape);

    point_shape shape();
    svg_1d_plot_series & size(int);
    int size();

    svg_1d_plot_series & stroke_color(const svg_color &);

    svg_color stroke_color();

    svg_1d_plot_series & symbols(const std::string &);

    const std::string symbols();
};
```

Description

Scan each data point sorting them into the appropriate std::vectors, normal or not (NaN or infinite). Member functions allow control of data points markers and lines joining them, and their appearance, shape, color and size. Data points can include their value, and optionally uncertainty and number of degrees of freedom. Each data series can have a title that can be shown on a legend box with identifying symbols.

svg_1d_plot_series public construct/copy/destruct

1.

```
template<typename C>
svg_1d_plot_series(C begin, C end, const std::string & title = "");
```

Parameters:

begin	iterator to 1st element in container to show.
end	iterator to last element in container to show.
title	Title of series of data values.

Template Parameters:

C	An iterator into STL container: array, std::vector<double>, std::vector<unc>, std::vector<Meas>, std::set, std::map ...
---	---

svg_1d_plot_series public member functions

1.

```
svg_1d_plot_series & bezier_on(bool on_);
```

Set true if to draw bezier curved line joining plot points.

Returns: Reference to **svg_1d_plot_series** to make chainable.

2.

```
bool bezier_on();
```

Returns: true if to draw bezier curved line joining plot points.

3.

```
svg_1d_plot_series & fill_color(const svg_color & col_);
```

Set fill color for plot point marker(s).

Returns: Reference to **svg_1d_plot_series** to make chainable.

4.

```
svg_color fill_color();
```

Get fill color for plot point marker(s).

Returns: Fill color for plot point marker(s).

5.

```
svg_1d_plot_series & limit_point_color(const svg_color & col_);
```

Set of stroke color of 'at limits' points.

Returns: Reference to **svg_1d_plot_series** to make chainable.

6.

```
svg_1d_plot_series & line_color(const svg_color & col_);
```

Set color of any line joining plot points.

Returns: Reference to **svg_1d_plot_series** to make chainable.

7. `svg_1d_plot_series & line_on(bool on_);`

Set true if to draw a line joining plot points.

Returns: Reference to `svg_1d_plot_series` to make chainable.

8. `bool line_on();`

Returns: true if to draw a line joining plot points.

9. `svg_1d_plot_series & line_width(double wid_);`

Set width of any line joining plot points.

Returns: Reference to `svg_1d_plot_series` to make chainable.

10. `double line_width();`

Returns: Width of any line joining plot points.

11. `size_t series_count();`

Returns: Number of normal 'OK to plot' data values in data series.

12. `size_t series_limits_count();`

Returns: Number of 'at limit' values: too big, too small or NaN data values in data series.

13. `svg_1d_plot_series & shape(point_shape shape_);`

Set shape for plot point marker(s).

Returns: Reference to `svg_1d_plot_series` to make chainable.

14. `point_shape shape();`

Get shape for plot point marker(s).

Returns: Shape for plot point marker(s).

15. `svg_1d_plot_series & size(int size_);`

Set size of plot point marker(s).

Returns: Reference to `svg_1d_plot_series` to make chainable.

16. `int size();`

Returns: size of plot point marker(s).

17. `svg_1d_plot_series & stroke_color(const svg_color & col_);`

Set stroke color for plot point marker(s).

Returns: Reference to `svg_1d_plot_series` to make chainable.

18. `svg_color stroke_color();`

Get stroke color for plot point marker(s).

Returns: Stroke color for plot point marker(s).

19. `svg_1d_plot_series & symbols(const std::string s);`

Set symbol for plot point marker(s).

Returns: Reference to `svg_1d_plot_series` to make chainable.

20. `const std::string symbols();`

Returns: symbol for plot point marker(s).

Header <[boost/svg_plot/svg_2d_plot.hpp](#)>

Create 2D XY plots in Scalable Vector Graphic (SVG) format.

Provides `svg_2d_plot` data and function to create plots, and `svg_2d_plot_series` to allow data values to be added.

Very many functions allow fine control of the appearance and layout of plots, data markers and lines.
(Many items common to 1D and 2D use functions and classes in `axis_plot_frame`).

Jacob Voytko & Paul A. Bristow

```
namespace boost {
    namespace svg {
        class svg_2d_plot;
        class svg_2d_plot_series;
    }
}
```

Class `svg_2d_plot`

`boost::svg::svg_2d_plot` — Provides `svg_2d_plot` data and member functions to create plots.

Very many functions allow very fine control of the appearance and layout of plots, data markers and lines.

Synopsis

```
// In header: <boost/svg_plot/svg_2d_plot.hpp>

class svg_2d_plot {
public:
    // construct/copy/destruct
    svg_2d_plot();

    // public member functions
    bool autoscale();
    svg_2d_plot & autoscale(bool);
    svg_2d_plot & autoscale_check_limits(bool);
    bool autoscale_check_limits();
    svg_2d_plot & autoscale_plusminus(double);
    double autoscale_plusminus();
    svg_2d_plot & axes_on(bool);
    bool axes_on();
    svg_2d_plot & background_border_color(const svg_color &);
    svg_color background_border_color();
    svg_2d_plot & background_border_width(double);
    double background_border_width();
    svg_color background_color();
    svg_2d_plot & background_color(const svg_color &);
    svg_2d_plot & boost_license_on(bool);
    bool boost_license_on();
    svg_2d_plot & confidence(double);
    double confidence();
    svg_2d_plot & coord_precision(int);
    int coord_precision();
    svg_2d_plot & copyright_date(const std::string);
    const std::string copyright_date();
    svg_2d_plot & copyright_holder(const std::string);
    const std::string copyright_holder();
    svg_2d_plot & data_lines_width(double);
    double data_lines_width();
    svg_2d_plot & derived();
    const svg_2d_plot & derived() const;
    svg_2d_plot & description(const std::string);
    const std::string & description();
    svg_2d_plot & document_title(const std::string);
    std::string document_title();
    svg_2d_plot &
    draw_line(double, double, double, const svg_color & = black);
    svg_2d_plot &
    draw_note(double, double, std::string, rotate_style = horizontal,
              align_style = center_align, const svg_color & = black,
              text_style & = no_style);
    svg_2d_plot &
    draw_plot_curve(double, double, double, double, double,
                   const svg_color & = black);
    svg_2d_plot &
    draw_plot_line(double, double, double, double, const svg_color & = black);
    svg_2d_plot & image_border_margin(double);
    double image_border_margin();
    svg_2d_plot & image_border_width(double);
    double image_border_width();
    unsigned int image_x_size();
    svg_2d_plot & image_x_size(unsigned int);
    unsigned int image_y_size();
    svg_2d_plot & image_y_size(unsigned int);
```

```

svg_2d_plot & legend_background_color(const svg_color &);
svg_color legend_background_color();
svg_2d_plot & legend_border_color(const svg_color &);
svg_color legend_border_color();
const std::pair< double, double > legend_bottom_right();
bool legend_box_fill_on();
svg_2d_plot & legend_color(const svg_color &);
svg_color legend_color();
svg_2d_plot & legend_font_family(const std::string &);
const std::string & legend_font_family();
svg_2d_plot & legend_font_weight(const std::string &);
const std::string & legend_font_weight();
svg_2d_plot & legend_header_font_size(int);
int legend_header_font_size();
svg_2d_plot & legend_lines(bool);
bool legend_lines();
svg_2d_plot & legend_on(bool);
bool legend_on();
bool legend_outside();
svg_2d_plot & legend_place(legend_places);
legend_places legend_place();
svg_2d_plot & legend_title(const std::string &);
const std::string legend_title();
svg_2d_plot & legend_title_font_size(unsigned int);
unsigned int legend_title_font_size();
svg_2d_plot & legend_top_left(double, double);
const std::pair< double, double > legend_top_left();
svg_2d_plot & legend_width(double);
double legend_width();
svg_2d_plot &
license(std::string = "permits", std::string = "permits",
        std::string = "requires", std::string = "permits",
        std::string = "permits");
const std::string license_attribution();
const std::string license_commercialuse();
const std::string license_distribution();
svg_2d_plot & license_on(bool);
bool license_on();
const std::string license_reproduction();
svg_2d_plot & limit_color(const svg_color &);
svg_color limit_color();
svg_2d_plot & limit_fill_color(const svg_color &);
svg_color limit_fill_color();
svg_2d_plot & one_sd_color(const svg_color &);
svg_color one_sd_color();
template<typename T>
  svg_2d_plot_series & plot(const T &, const std::string & = "");
template<typename T, typename U>
  svg_2d_plot_series &
  plot(const T &, const std::string & = "", U = unspecified);
template<typename T>
  svg_2d_plot_series & plot(const T &, const T &, const std::string & = "");
template<typename T, typename U>
  svg_2d_plot_series &
  plot(const T &, const T &, const std::string & = "", U = unspecified);
svg_2d_plot & plot_background_color(const svg_color &);
svg_color plot_background_color();
svg_2d_plot & plot_border_color(const svg_color &);
svg_color plot_border_color();
svg_2d_plot & plot_border_width(double);
double plot_border_width();
svg_2d_plot & plot_window_on(bool);
bool plot_window_on();

```

```

svg_2d_plot & plot_window_x(double, double);
std::pair< double, double > plot_window_x();
double plot_window_x_left();
double plot_window_x_right();
svg_2d_plot & plot_window_y(double, double);
std::pair< double, double > plot_window_y();
double plot_window_y_bottom();
double plot_window_y_top();
svg_2d_plot & size(unsigned int, unsigned int);
std::pair< double, double > size();
svg_2d_plot & three_sd_color(const svg_color &);
svg_color three_sd_color();
svg_2d_plot & title(const std::string &);
const std::string title();
svg_2d_plot & title_color(const svg_color &);
svg_color title_color();
svg_2d_plot & title_font_alignment(align_style);
align_style title_font_alignment();
svg_2d_plot & title_font_decoration(const std::string &);
const std::string & title_font_decoration();
svg_2d_plot & title_font_family(const std::string &);
const std::string & title_font_family();
svg_2d_plot & title_font_rotation(rotate_style);
int title_font_rotation();
svg_2d_plot & title_font_size(unsigned int);
unsigned int title_font_size();
svg_2d_plot & title_font_stretch(const std::string &);
const std::string & title_font_stretch();
svg_2d_plot & title_font_style(const std::string &);
const std::string & title_font_style();
svg_2d_plot & title_font_weight(const std::string &);
const std::string & title_font_weight();
svg_2d_plot & title_on(bool);
bool title_on();
svg_2d_plot & two_sd_color(const svg_color &);
svg_color two_sd_color();
svg_2d_plot & write(const std::string &);
svg_2d_plot & write(std::ostream &);
svg_2d_plot & x_addlimits_color(const svg_color &);
svg_color x_addlimits_color();
svg_2d_plot & x_addlimits_on(bool);
bool x_addlimits_on();
double x_auto_max_value();
double x_auto_min_value();
double x_auto_tick_interval();
int x_auto_ticks();
bool x_autoscale();
svg_2d_plot & x_autoscale(bool);
svg_2d_plot & x_autoscale(std::pair< double, double >);
svg_2d_plot & x_autoscale(const T &);
svg_2d_plot & x_autoscale(const T &, const T &);
axis_line_style & x_axis();
svg_2d_plot & x_axis_color(const svg_color &);
svg_color x_axis_color();
svg_2d_plot & x_axis_label_color(const svg_color &);
svg_color x_axis_label_color();
svg_2d_plot & x_axis_on(bool);
bool x_axis_on();
const std::string x_axis_position();
svg_2d_plot & x_axis_vertical(double);
bool x_axis_vertical();
svg_2d_plot & x_axis_width(double);
double x_axis_width();

```

```

svg_2d_plot & x_datetime_color(const svg_color &);
svg_color x_datetime_color();
svg_2d_plot & x_datetime_on(bool);
bool x_datetime_on();
svg_2d_plot &
x_decor(const std::string &, const std::string & = "",
        const std::string & = "");
svg_2d_plot & x_df_color(const svg_color &);
svg_color x_df_color();
svg_2d_plot & x_df_on(bool);
bool x_df_on();
svg_2d_plot & x_id_color(const svg_color &);
svg_color x_id_color();
svg_2d_plot & x_id_on(bool);
bool x_id_on();
svg_2d_plot & x_label(const std::string &);
std::string x_label();
svg_2d_plot & x_label_color(const svg_color &);
svg_color x_label_color();
svg_2d_plot & x_label_font_family(const std::string &);
const std::string & x_label_font_family();
svg_2d_plot & x_label_font_size(unsigned int);
unsigned int x_label_font_size();
svg_2d_plot & x_label_on(bool);
bool x_label_on();
svg_2d_plot & x_label_units(const std::string &);
std::string x_label_units();
svg_2d_plot & x_label_units_on(bool);
bool x_label_units_on();
svg_2d_plot & x_label_width(double);
double x_label_width();
svg_2d_plot & x_labels_strip_e0s(bool);
svg_2d_plot & x_major_grid_color(const svg_color &);
svg_color x_major_grid_color();
svg_2d_plot & x_major_grid_on(bool);
bool x_major_grid_on();
svg_2d_plot & x_major_grid_width(double);
double x_major_grid_width();
svg_2d_plot & x_major_interval(double);
double x_major_interval();
svg_2d_plot & x_major_label_rotation(rotate_style);
rotate_style x_major_label_rotation();
svg_2d_plot & x_major_labels_side(int);
int x_major_labels_side();
svg_2d_plot & x_major_tick(double);
double x_major_tick();
svg_2d_plot & x_major_tick_color(const svg_color &);
svg_color x_major_tick_color();
svg_2d_plot & x_major_tick_length(double);
double x_major_tick_length();
svg_2d_plot & x_major_tick_width(double);
double x_major_tick_width();
svg_2d_plot & x_max(double);
double x_max();
svg_2d_plot & x_min(double);
double x_min();
svg_2d_plot & x_min_ticks(int);
int x_min_ticks();
svg_2d_plot & x_minor_grid_color(const svg_color &);
svg_color x_minor_grid_color();
svg_2d_plot & x_minor_grid_on(bool);
bool x_minor_grid_on();
svg_2d_plot & x_minor_grid_width(double);

```

```

double x_minor_grid_width();
double x_minor_interval();
svg_2d_plot & x_minor_interval(double);
svg_2d_plot & x_minor_tick_color(const svg_color &);

svg_color x_minor_tick_color();
svg_2d_plot & x_minor_tick_length(double);
double x_minor_tick_length();
svg_2d_plot & x_minor_tick_width(double);
double x_minor_tick_width();
svg_2d_plot & x_num_minor_ticks(unsigned int);
unsigned int x_num_minor_ticks();
svg_2d_plot & x_order_color(const svg_color &);

svg_color x_order_color();
svg_2d_plot & x_order_on(bool);
bool x_order_on();
svg_2d_plot & x_plusminus_color(const svg_color &);

svg_color x_plusminus_color();
svg_2d_plot & x_plusminus_on(bool);
bool x_plusminus_on();
const std::string x_prefix();

svg_2d_plot & x_range(double, double);
std::pair<double, double> x_range();
const std::string x_separator();
svg_2d_plot & x_size(unsigned int);
unsigned int x_size();
svg_2d_plot & x_steps(int);
int x_steps();
const std::string x_suffix();
ticks_labels_style & x_ticks();
svg_2d_plot & x_ticks_down_on(bool);
bool x_ticks_down_on();
svg_2d_plot & x_ticks_on_window_or_axis(int);
int x_ticks_on_window_or_axis();
svg_2d_plot & x_ticks_up_on(bool);
bool x_ticks_up_on();
svg_2d_plot & x_ticks_values_color(const svg_color &);

svg_color x_ticks_values_color();
svg_2d_plot & x_ticks_values_font_family(const std::string &);

const std::string & x_ticks_values_font_family();
svg_2d_plot & x_ticks_values_font_size(unsigned int);
unsigned int x_ticks_values_font_size();
svg_2d_plot & x_ticks_values_ioflags(std::ios_base::fmtflags);
std::ios_base::fmtflags x_ticks_values_ioflags();
svg_2d_plot & x_ticks_values_precision(int);
int x_ticks_values_precision();
svg_2d_plot & x_tight(double);
double x_tight();
svg_2d_plot & x_value_font_size(unsigned int);
unsigned int x_value_font_size();
svg_2d_plot & x_value_ioflags(std::ios_base::fmtflags);
std::ios_base::fmtflags x_value_ioflags();
svg_2d_plot & x_value_precision(int);
int x_value_precision();
svg_2d_plot & x_values_color(const svg_color &);

svg_color x_values_color();
svg_2d_plot & x_values_font_family(const std::string &);

const std::string & x_values_font_family();
svg_2d_plot & x_values_font_size(unsigned int);
unsigned int x_values_font_size();
svg_2d_plot & x_values_ioflags(std::ios_base::fmtflags);
std::ios_base::fmtflags x_values_ioflags();
svg_2d_plot & x_values_on(bool);
bool x_values_on();

```

```

svg_2d_plot & x_values_precision(int);
int x_values_precision();
svg_2d_plot & x_values_rotation(rotate_style);
int x_values_rotation();
svg_2d_plot & x_with_zero(bool);
bool x_with_zero();
template<typename T> svg_2d_plot & xy_autoscale(const T &);
bool xy_autoscale();
bool xy_values_on();
svg_2d_plot & xy_values_on(bool);
svg_2d_plot & y_addlimits_color(const svg_color &);
const svg_color y_addlimits_color();
bool y_addlimits_on();
svg_2d_plot & y_addlimits_on(bool);
bool y_autoscale();
svg_2d_plot & y_autoscale(bool);
svg_2d_plot & y_autoscale(double, double);
svg_2d_plot & y_autoscale(std::pair<double, double >);
template<typename T> svg_2d_plot & y_autoscale(const T &, const T &);
template<typename T> svg_2d_plot & y_autoscale(const T &);
axis_line_style & y_axis();
svg_2d_plot & y_axis_color(const svg_color &);
svg_color y_axis_color();
svg_2d_plot & y_axis_label_color(const svg_color &);
svg_color y_axis_label_color();
svg_2d_plot & y_axis_on(bool);
bool y_axis_on();
const std::string y_axis_position();
svg_2d_plot & y_axis_value_color(const svg_color &);
svg_color y_axis_value_color();
svg_2d_plot & y_axis_width(double);
double y_axis_width();
svg_2d_plot &
y_decor(const std::string &, const std::string & = "",
        const std::string & = "");
svg_2d_plot & y_df_color(const svg_color &);
const svg_color y_df_color();
bool y_df_on();
svg_2d_plot & y_df_on(bool);
svg_2d_plot & y_label(const std::string &);
std::string y_label();
svg_2d_plot & y_label_axis(const std::string &);
std::string y_label_axis();
svg_2d_plot & y_label_color(const svg_color &);
svg_color y_label_color();
svg_2d_plot & y_label_font_family(const std::string &);
const std::string & y_label_font_family();
svg_2d_plot & y_label_font_size(unsigned int);
unsigned int y_label_font_size();
svg_2d_plot & y_label_on(bool);
bool y_label_on();
svg_2d_plot & y_label_units(const std::string &);
std::string y_label_units();
svg_2d_plot & y_label_units_on(bool);
bool y_label_units_on();
svg_2d_plot & y_label_weight(std::string);
const std::string & y_label_weight();
svg_2d_plot & y_label_width(double);
double y_label_width();
svg_2d_plot & y_labels_strip_e0s(bool);
bool y_labels_strip_e0s();
svg_2d_plot & y_major_grid_color(const svg_color &);
const svg_color y_major_grid_color();

```

```

svg_2d_plot & y_major_grid_on(bool);
bool y_major_grid_on();
svg_2d_plot & y_major_grid_width(double);
double y_major_grid_width();
double y_major_interval();
svg_2d_plot & y_major_interval(double);
svg_2d_plot & y_major_label_rotation(rotate_style);
int y_major_label_rotation();
svg_2d_plot & y_major_labels_side(int);
int y_major_labels_side();
svg_2d_plot & y_major_tick_color(const svg_color &);
const svg_color y_major_tick_color();
double y_major_tick_length();
svg_2d_plot & y_major_tick_length(double);
svg_2d_plot & y_major_tick_width(double);
double y_major_tick_width();
double y_max();
double y_min();
svg_2d_plot & y_minor_grid_color(const svg_color &);
const svg_color y_minor_grid_color();
svg_2d_plot & y_minor_grid_on(bool);
bool y_minor_grid_on();
svg_2d_plot & y_minor_grid_width(double);
double y_minor_grid_width();
double y_minor_interval();
svg_2d_plot & y_minor_tick_color(const svg_color &);
const svg_color y_minor_tick_color();
svg_2d_plot & y_minor_tick_length(double);
double y_minor_tick_length();
svg_2d_plot & y_minor_tick_width(double);
double y_minor_tick_width();
svg_2d_plot & y_num_minor_ticks(unsigned int);
unsigned int y_num_minor_ticks();
svg_2d_plot & y_plusminus_color(const svg_color &);
const svg_color y_plusminus_color();
bool y_plusminus_on();
svg_2d_plot & y_plusminus_on(bool);
const std::string y_prefix();
svg_2d_plot & y_range(double, double);
std::pair< double, double > y_range();
const std::string y_separator();
unsigned int y_size();
svg_2d_plot & y_size(unsigned int);
const std::string y_suffix();
ticks_labels_style & y_ticks();
svg_2d_plot & y_ticks_left_on(bool);
bool y_ticks_left_on();
svg_2d_plot & y_ticks_on_window_or_axis(int);
int y_ticks_on_window_or_axis();
svg_2d_plot & y_ticks_right_on(bool);
bool y_ticks_right_on();
svg_2d_plot & y_ticks_values_color(const svg_color &);
svg_color y_ticks_values_color();
svg_2d_plot & y_ticks_values_font_family(const std::string &);
const std::string & y_ticks_values_font_family();
svg_2d_plot & y_ticks_values_font_size(unsigned int);
unsigned int y_ticks_values_font_size();
svg_2d_plot & y_ticks_values_ioflags(std::ios_base::fmtflags);
std::ios_base::fmtflags y_ticks_values_ioflags();
svg_2d_plot & y_ticks_values_precision(int);
int y_ticks_values_precision();
svg_2d_plot & y_value_ioflags(std::ios_base::fmtflags);
int y_value_ioflags();

```

```

svg_2d_plot & y_value_precision(int);
int y_value_precision();
svg_2d_plot & y_values_color(const svg_color &);

svg_color y_values_color();
svg_2d_plot & y_values_font_family(const std::string &);

const std::string & y_values_font_family();

svg_2d_plot & y_values_font_size(unsigned int);

unsigned int y_values_font_size();

svg_2d_plot & y_values_ioflags(std::ios_base::fmtflags);

std::ios_base::fmtflags y_values_ioflags();

bool y_values_on();

svg_2d_plot & y_values_on(bool);

svg_2d_plot & y_values_precision(int);

int y_values_precision();

svg_2d_plot & y_values_rotation(rotate_style);

int y_values_rotation();

};


```

Description

See Also:

[svg_2d_plot_series](#) that allows data values to be added.

[svg_1d_plot.hpp](#) for 1-D version.

[svg_2d_plot](#) allows us to store plot state locally in [svg_plot](#).

(We don't store it in [svg](#) because transforming the points after they are written to the document would be difficult. We store the Cartesian coordinates locally and transform them before we write them).

([svg_2d_plot](#) inherits from [axis_plot_frame.hpp](#) containing functions common to 1 and 2-D).

svg_2d_plot public construct/copy/destruct

1. [svg_2d_plot\(\)](#)

Default constructor inheriting from public [axis_plot_frame<svg_2d_plot>](#), providing all the very many default plot options, some of which use some or all of the style defaults.

All these settings can be changed by these chainable functions.

Example:

```

svg_2d_plot my_plot;
my_plot.background_color(ghostwhite) // Whole image.
    .legend_border_color(yellow) // Just the legend box.
    .legend_background_color(lightyellow) // Fill color of the legend box.
    .plot_background_color(svg_color(white)) // Just the plot window
    .plot_border_color(svg_color(green)) // The border rectangle color.
    .plot_border_width(1) // Thin border (SVG units, default pixels).
    .title_color(red) // Title of whole image.
;

```

See Also:

Rationale for default plot options and style settings in documentation.

svg_2d_plot public member functions

1. [bool autoscale\(\)](#)

Returns: true if to use autoscale values autoscaling for X-axis.

2. `svg_2d_plot & autoscale(bool b);`

Set true if to use autoscale values for X-axis.

3. `svg_2d_plot & autoscale_check_limits(bool b);`

Set true to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed if user can be sure all values are 'inside limits'.

4. `bool autoscale_check_limits();`

Returns: true if to check that values used for autoscaling are within limits.

5. `svg_2d_plot & autoscale_plusminus(double);`

Set how many std_dev or standard deviations to allow for ellipses when autoscaling.

6. `double autoscale_plusminus();`

Returns: How many std_dev or standard deviations allowed for ellipses when autoscaling.

7. `svg_2d_plot & axes_on(bool is);`

Set true if to draw **both** x and y axes (note plural axes).

8. `bool axes_on();`

Returns: true if to draw **both** x and y axis on.

9. `svg_2d_plot & background_border_color(const svg_color & col);`

Set plot background border color.

10. `svg_color background_border_color();`

Returns: plot background border color.

11. `svg_2d_plot & background_border_width(double w);`

Set plot background border width.

12. `double background_border_width();`

Returns: Plot background border width.

13. `svg_color background_color();`

Returns: Plot background color.

14. `svg_2d_plot & background_color(const svg_color & col);`

Set plot background color.

15. `svg_2d_plot & boost_license_on(bool l);`

Set true if the Boost license conditions should be included in the SVG document.

16. `bool boost_license_on();`

Returns: true if the Boost license conditions should be included in the SVG document.

17. `svg_2d_plot & confidence(double);`

Set confidence alpha for display of confidence intervals (default 0.05 for 95%).

18. `double confidence();`

Returns: Confidence alpha for display of confidence intervals (default 0.05 for 95%).

19. `svg_2d_plot & coord_precision(int digits);`

Precision of SVG coordinates in decimal digits (default 3). 3 decimal digits precision is sufficient for small images. 4 or 5 decimal digits precision will give higher quality plots, especially for larger images, at the expense of larger .svg files, particularly if there are very many data points.

20. `int coord_precision();`

Returns: precision of SVG coordinates in decimal digits.

21. `svg_2d_plot & copyright_date(const std::string d);`

Writes copyright date to the SVG document. and as metadata:<meta name="date" content="2007" />

22. `const std::string copyright_date();`

Returns: SVG document copyright_date.

23. `svg_2d_plot & copyright_holder(const std::string d);`

Writes copyright_holder metadata to the SVG document (for header as) /and as metadata:<meta name="copyright" content="Paul A. Bristow" />

24. `const std::string copyright_holder();`

Returns: SVG document copyright holder.

25. `svg_2d_plot & data_lines_width(double width);`

Set the width of lines joining data points.

26. `double data_lines_width();`

Returns: the width of lines joining data points.

27. `svg_2d_plot & derived();`

Uses Curiously Recurring Template Pattern to allow 1D and 2D to reuse common code. See http://en.wikipedia.org/wiki/Curiously_Recurring_Template_Pattern.

28. `const svg_2d_plot & derived() const;`

const version of derived()

29. `svg_2d_plot & description(const std::string d);`

Writes description to the document for header as.

`<desc> My Description </desc>.`

30. `const std::string & description();`

Returns: Description of the document for header as`<desc> My description </desc>.`

31. `svg_2d_plot & document_title(const std::string d);`

Set document title to the document for header as.

`<title> My Title </title>.`

32. `std::string document_title();`

Returns: Document title to the document for header as`<title> My Title </title>.`

33. `svg_2d_plot & draw_line(double x1, double y1, double x2, double y2,
const svg_color & col = black);`

Annotate plot with a line from SVG Coordinates X1, Y1 to X2, Y2. (Default color black). Note NOT the data values. See draw_plot_line if want to use user coordinates.

34. `svg_2d_plot &
draw_note(double x, double y, std::string note, rotate_style rot = horizontal,
align_style al = center_align, const svg_color & col = black,
text_style tsty = no_style);`

Annotate plot with a text string (perhaps including Unicode), putting note at SVG Coordinates X, Y.

Defaults color black, rotation horizontal and align = center_align Using center_align is recommended as it will ensure that will center correctly (even if original string is made much longer because it contains Unicode, for example Greek or math symbols, taking about 6 characters per symbol) because the render engine does the centering.

35. `svg_2d_plot &
draw_plot_curve(double x1, double y1, double x2, double y2, double x3,
double y3, const svg_color & col = black);`

Annotate plot with a line from user's Cartesian Coordinates X1, Y1 via X2, Y2 to X3, Y3.

For example, -10, -10, +10, +10, Default color black.

36. `svg_2d_plot & draw_plot_line(double x1, double y1, double x2, double y2, const svg_color & col = black);`

Annotate plot with a line from user's Cartesian Coordinates X1, Y1 to X2, Y2.

For example, -10, -10, +10, +10, Default color black.

37. `svg_2d_plot & image_border_margin(double w);`

Set the margin around the plot window border (svg units, default pixels).

This prevents the plot window getting too close to other elements of the plot.

38. `double image_border_margin();`

Returns: the margin around the plot window border (svg units, default pixels).

39. `svg_2d_plot & image_border_width(double w);`

Set the svg image border width (svg units, default pixels).

40. `double image_border_width();`

Returns: the svg image border width (svg units, default pixels).

41. `unsigned int image_x_size();`

Obselete - deprecated use x_size()

42. `svg_2d_plot & image_x_size(unsigned int i);`

Obselete - deprecated - use x_size().

43. `unsigned int image_y_size();`

Obselete - deprecated - use y_size()

44. `svg_2d_plot & image_y_size(unsigned int i);`

Obselete - deprecated - use y_size()

45. `svg_2d_plot & legend_background_color(const svg_color & col);`

Set the background fill color of the legend box.

46. `svg_color legend_background_color();`

Returns: the background fill color of the legend box.

47. `svg_2d_plot & legend_border_color(const svg_color & col);`

Set the border stroke color of the legend box.

48. `svg_color legend_border_color();`

Returns: the border stroke color of the legend box.

49. `const std::pair< double, double > legend_bottom_right();`

Returns: SVG coordinate (default pixels) of bottom right of legend box.

50. `bool legend_box_fill_on();`

Returns: true if legend box has a background fill color.

51. `svg_2d_plot & legend_color(const svg_color & col);`

Set the color of the title of the legend.

52. `svg_color legend_color();`

Returns: the color of the title of the legend.

53. `svg_2d_plot & legend_font_family(const std::string & family);`

Set the font family for the legend title.

54. `const std::string & legend_font_family();`

Returns: the font family for the legend title.

55. `svg_2d_plot & legend_font_weight(const std::string & weight);`

Set the font weight for the legend title.

56. `const std::string & legend_font_weight();`

Returns: Font weight for the legend title.

57. `svg_2d_plot & legend_header_font_size(int size);`

Set legend header font size (svg units, default pixels).

58. `int legend_header_font_size();`

Returns: legend header font size (svg units, default pixels).

59. `svg_2d_plot & legend_lines(bool is);`

Set true if legend should include samples of the lines joining data points. This allows different series of data points to be distinguished by different color and/or width. This is especially useful to show plots of different functions and/or different parameters in different colors.

60. `bool legend_lines();`

Returns: true if legend should include samples of the lines joining data points.

61. `svg_2d_plot & legend_on(bool cmd);`

Set true if a legend is wanted.

62. `bool legend_on();`

Returns: true if a legend is wanted.

63. `bool legend_outside();`

Returns: if the legend should be outside the plot area.

64. `svg_2d_plot & legend_place(legend_places l);`

Set the position of the legend.,

See Also:

`boost::svg::legend_places`

65. `legend_places legend_place();`

See Also:

`boost::svg::legend_places`

Returns: the position of the legend,

66. `svg_2d_plot & legend_title(const std::string title);`

Set the title for the legend.

67. `const std::string legend_title();`

Returns: Title for the legend.

68. `svg_2d_plot & legend_title_font_size(unsigned int size);`

Returns: Font family for the legend title.

69. `unsigned int legend_title_font_size();`

Returns: Font size for the legend title (svg units, default pixels).

70. `svg_2d_plot & legend_top_left(double x, double y);`

Set position of top left of legend box (svg coordinates, default pixels). (Bottom right is controlled by contents, so the user cannot set it).

71. `const std::pair< double, double > legend_top_left();`

Returns: SVG coordinate (default pixels) of top left of legend box.

72. `svg_2d_plot & legend_width(double width);`

Set the width for the legend box.

73. `double legend_width();`

Returns: Width for the legend box.

74. `svg_2d_plot &`
`license(std::string repro = "permits", std::string distrib = "permits",`
 `std::string attrib = "requires", std::string commercial = "permits",`
 `std::string derivative = "permits");`

Set license conditions for reproduction, attribution, commercial use, and derivative works, usually "permits", "requires", or "prohibits", and set license_on == true.

75. `const std::string license_attribution();`

Returns: SVG document attribution license conditions, usually "permits", "requires", or "prohibits".

76. `const std::string license_commercialuse();`

Returns: SVG document commercial use license conditions, usually "permits", "requires", or "prohibits".

77. `const std::string license_distribution();`

Returns: SVG document distribution license conditions, usually "permits", "requires", or "prohibits".

78. `svg_2d_plot & license_on(bool l);`

Set if license conditions should be included in the SVG document.

79. `bool license_on();`

Returns: true if license conditions should be included in the SVG document.

80. `const std::string license_reproduction();`

Returns: SVG document reproduction license conditions, usually "permits", "requires", or "prohibits".

81. `svg_2d_plot & limit_color(const svg_color &);`

Set the color for 'at limit' point stroke color.

82 `svg_color limit_color();`

Returns: the color for the 'at limit' point stroke color.

83. `svg_2d_plot & limit_fill_color(const svg_color &);`

Set the color for 'at limit' point fill color.

84. `svg_color limit_fill_color();`

Returns: the color for the 'at limit' point fill color.

85. `svg_2d_plot & one_sd_color(const svg_color &);`

Set the color for the one standard deviation (~67% confidence) ellipse fill.

86. `svg_color one_sd_color();`

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

87. `template<typename T>
svg_2d_plot_series &
plot(const T & container, const std::string & title = "");`

Add a container of a data series to the plot.

(Version converting to Meas using double with pair_double_2d_convert).

Example:

```
my_plot.plot(data1, "Sqrt(x));
```



Note

This version assumes that **ALL** the data values in the container is used.

Template Parameters: `T` Type of data in series (must be convertible to Meas).

Returns: Reference to data series just added to make chainable.

88. `template<typename T, typename U>
svg_2d_plot_series &
plot(const T & container, const std::string & title = "",
U functor = unspecified);`

Add a container of a data series to the plot.

This version permits a custom functor (rather than default conversion to double).



Note

that this version assumes that **ALL** the data values in the container is used.

Returns: Reference to data series just added to make chainable.

89. `template<typename T>
svg_2d_plot_series &
plot(const T & begin, const T & end, const std::string & title = "");`

Add a data series to the plot (by default, converting automatically to `unc doubles`).
 This version permits **part** of the container to be used, a partial range, using iterators begin to end.
 For example:

```
my_2d_plot.plot(my_data.begin(), my_data.end(), "My container");
```

```
my_2d_plot.plot(&my_data[1], &my_data[3], "my_data 1 to 3"); // Add part of data series.
```

Returns: Reference to data series just added to make chainable.

90. `template<typename T, typename U>
svg_2d_plot_series &
plot(const T & begin, const T & end, const std::string & title = "",
U functor = unspecified);`

Returns: Reference to data series just added to make chainable.

91. `svg_2d_plot & plot_background_color(const svg_color & col);`

Set the fill color of the plot window background.

92. `svg_color plot_background_color();`

Returns: the fill color of the plot window background.

93. `svg_2d_plot & plot_border_color(const svg_color & col);`

Set the color for the plot window background.

94. `svg_color plot_border_color();`

Returns: the color for the plot window background.

95. `svg_2d_plot & plot_border_width(double w);`

Set the margin around the plot window border (svg units, default pixels).

This prevents the plot window getting too close to other elements of the plot.

96. `double plot_border_width();`

Returns: the width for the plot window border (svg units, default pixels).

97. `svg_2d_plot & plot_window_on(bool cmd);`

Set true if a plot window is wanted (or false if the whole image is to be used).

98 `bool plot_window_on();`

Returns: true if a plot window is wanted (or false if the whole image is to be used).

99 `svg_2d_plot & plot_window_x(double min_x, double max_x);`

Set the minimum and maximum (cartesian data units) for the plot window X axis. This is normally calculated from other plot values.

100 `std::pair< double, double > plot_window_x();`

Returns: both the left and right (X axis) of the plot window.

101 `double plot_window_x_left();`

Returns: left of the plot window.

102 `double plot_window_x_right();`

Returns: right of the plot window.

103 `svg_2d_plot & plot_window_y(double min_y, double max_y);`

Set the minimum and maximum (cartesian data units) for the plot window Y axis. This is normally calculated from other plot values.

104 `std::pair< double, double > plot_window_y();`

Returns: both the top and bottom (Y axis) of the plot window.

105 `double plot_window_y_bottom();`

Returns: top of the plot window.

106 `double plot_window_y_top();`

Returns: top of the plot window.

107 `svg_2d_plot & size(unsigned int x, unsigned int y);`

Set SVG image size (SVG units, default pixels).

108 `std::pair< double, double > size();`

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

109 `svg_2d_plot & three_sd_color(const svg_color &);`

Set the color for three standard deviation (~99% confidence) ellipse fill.

10 `svg_color three_sd_color();`

Returns: Color for three standard deviation (~99% confidence) ellipse fill.

11 `svg_2d_plot & title(const std::string title);`

Set a title for plot. The string may include Unicode for greek letter and symbols. **example:** A title that includes a greek omega and degree symbols:

```
my_plot.title("Plot of &#x3A9; function (&#x00B0;C)");
```

Unicode symbols are at <http://unicode.org/charts/symbols.html>.

12 `const std::string title();`

Returns: Title for plot (whose string may include Unicode for greek letter and symbols).

13 `svg_2d_plot & title_color(const svg_color & col);`

Set the color of any title of the plot.

14 `svg_color title_color();`

Returns: the color of any title of the plot.

15 `svg_2d_plot & title_font_alignment(align_style alignment);`

Set the alignment for the title.

16 `align_style title_font_alignment();`

Returns: the alignment for the title.

17 `svg_2d_plot & title_font_decoration(const std::string & decoration);`

Set the font decoration for the title (default normal, or underline, overline or strike-thru).

18 `const std::string & title_font_decoration();`

Returns: Font decoration for the title (default normal, or underline, overline or strike-thru).

19 `svg_2d_plot & title_font_family(const std::string & family);`

Set the font family for the title (for example: .title_font_family("Lucida Sans Unicode");

20 `const std::string & title_font_family();`

Returns: Font family for the title.

21 `svg_2d_plot & title_font_rotation(rotate_style rotate);`

Set the rotation for the title font (degrees, 0 to 360 in steps using rotate_style, for example horizontal, uphill...)

12 `int title_font_rotation();`

Returns: the rotation for the title font (degrees).

13 `svg_2d_plot & title_font_size(unsigned int i);`

Sets the font size for the title (SVG units, default pixels).

14 `unsigned int title_font_size();`

Returns: Font size for the title (SVG units, default pixels).

15 `svg_2d_plot & title_font_stretch(const std::string & stretch);`

Set the font stretch for the title (default normal), wider or narrow.

16 `const std::string & title_font_stretch();`

Returns: Font stretch for the title.

17 `svg_2d_plot & title_font_style(const std::string & style);`

Set the font style for the title (default normal).

18 `const std::string & title_font_style();`

Returns: Font style for the title (default normal).

19 `svg_2d_plot & title_font_weight(const std::string & weight);`

Set the font weight for the title (default normal).

20 `const std::string & title_font_weight();`

Returns: Font weight for the title.

21 `svg_2d_plot & title_on(bool cmd);`

If set true, show a title for the plot. Note: is set true by setting a title.

22 `bool title_on();`

Returns: true if will show a title for the plot.

23 `svg_2d_plot & two_sd_color(const svg_color &);`

Set the color for two standard deviation (~95% confidence) ellipse fill.

14 `svg_color two_sd_color();`

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

15 `svg_2d_plot & write(const std::string & file);`

Write the plot image to a named file (default suffix .svg, added if no type already appended to file name).

Returns: Reference to `svg_2d_plot` to make chainable.

16 `svg_2d_plot & write(std::ostream & s_out);`

Write the SVG image to a `std::ostream`.

Returns: Reference to `svg_2d_plot` to make chainable.

17 `svg_2d_plot & x_addlimits_color(const svg_color & col);`

Set the color of X confidence limits of value, for example, the color in "<1.23, 1.45>".

18 `svg_color x_addlimits_color();`

Returns: the color of X confidence limits of value, for example, the color of "<1.23, 1.45>".

19 `svg_2d_plot & x_addlimits_on(bool b);`

Set if to append confidence limits to data point X values near data points markers.

20 `bool x_addlimits_on();`

Returns: true if to append confidence limits estimate to data point X values near data points markers.

21 `double x_auto_max_value();`

Returns: X-axis maximum value computed by autoscale.

22 `double x_auto_min_value();`

Returns: X-axis minimum value computed by autoscale.

23 `double x_auto_tick_interval();`

Returns: the X-axis major tick interval computed by autoscale.

24 `int x_auto_ticks();`

Returns: the X-axis number of major ticks computed by autoscale.

25 `bool x_autoscale();`

Returns: true if to use autoscale value for X-axis.

16 `svg_2d_plot & x_autoscale(bool b);`

Set true if to use autoscale values for X-axis.

17 `svg_2d_plot & x_autoscale(std::pair< double, double > p);`

autoscale X axis using a pair of doubles.

18 `svg_2d_plot & x_autoscale(const T & container);`

<

19 `svg_2d_plot & x_autoscale(const T & begin, const T & end);`

<

20 `axis_line_style & x_axis();`

Returns: true if horizontal X-axis line to be drawn.

21 `svg_2d_plot & x_axis_color(const svg_color & col);`

Set the color of the X-axis line.

22 `svg_color x_axis_color();`

Returns: the color of the X-axis line.

23 `svg_2d_plot & x_axis_label_color(const svg_color & col);`

Set X axis label color, for example, red.

24 `svg_color x_axis_label_color();`

Returns: X axis label color. <X-axis ticks values label style.

25 `svg_2d_plot & x_axis_on(bool is);`

If set true, draw a horizontal X-axis line.

26 `bool x_axis_on();`

Returns: true if will draw a horizontal X-axis line.

27 `const std::string x_axis_position();`

Returns: the position (or intersection with Y-axis) of the X-axis.

18 `svg_2d_plot & x_axis_vertical(double fraction);`

Set vertical position of X-axis for 1D as fraction of plot window.

19 `bool x_axis_vertical();`

Returns: vertical position of X-axis for 1D as fraction of plot window.

10 `svg_2d_plot & x_axis_width(double width);`

Set the width of X-axis lines.

11 `double x_axis_width();`

Returns: the width of X-axis lines.

12 `svg_2d_plot & x_datetime_color(const svg_color & col);`

Set the color of X date time , for example, the color of text in "".

13 `svg_color x_datetime_color();`

Returns: the color of X date time, for example, the color of text in "".

14 `svg_2d_plot & x_datetime_on(bool b);`

Set true if to append date time to data point X values near data points markers.

15 `bool x_datetime_on();`

Returns: true if to append an date time to data point X values near data points markers.

16 `svg_2d_plot & x_decor(const std::string & pre, const std::string & sep = "", const std::string & suf = "");`

Set prefix, separator and suffix together for x_ values. Note if you want a space, you must use a Unicode space " ", for example, ", " rather than ASCII space", ". If 1st char in separator == , then Y values and info will be on a newline below.

17 `svg_2d_plot & x_df_color(const svg_color & col);`

Set the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

18 `svg_color x_df_color();`

Returns: the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

19 `svg_2d_plot & x_df_on(bool b);`

Set true if to append a degrees of freedom estimate to data point X values near data points markers.

I0 `bool x_df_on();`

Returns: true if to append a degrees of freedom estimate to data point X values near data points markers.

I1 `svg_2d_plot & x_id_color(const svg_color & col);`

Set the color of X id or name, for example, the color of text in "my_id".

I2 `svg_color x_id_color();`

Returns: the color of X X id or name, for example, the color of text in "my_id".

I3 `svg_2d_plot & x_id_on(bool b);`

Set true if to append append an ID or name to data point X values near data points markers.

I4 `bool x_id_on();`

Returns: true if to append an ID or name to data point X values near data points markers.

I5 `svg_2d_plot & x_label(const std::string & str);`

Set the text to label the X-axis (and set x_label_on(true)).

I6 `std::string x_label();`

Returns: the text to label the X-axis.

I7 `svg_2d_plot & x_label_color(const svg_color & col);`

Returns: the color of the Y-axis line.

I8 `svg_color x_label_color();`

Returns: the color of X-axis label (including any units).

I9 `svg_2d_plot & x_label_font_family(const std::string & family);`

Set X tick value label font family. Available fonts depend on the program rendering the SVG XML, usually a browser. The default font (usually "default_font") is used if a render program does not provide the font specified. These are probably usable:

```
"arial", "impact", "courier", "lucida console", "Lucida sans unicode", "verdana", "calibri", "century",
"lucida calligraphy", "tahoma", "vivaldi", "informal roman", "lucida handwriting", "lucida bright", "helvetica"
```

I0 `const std::string & x_label_font_family();`

Returns: X tick value label font family.

I1 `svg_2d_plot & x_label_font_size(unsigned int i);`

Set X axis label font size (svg units, default pixels).

12 `unsigned int x_label_font_size();`

Returns: X axis label font size (svg units, default pixels).

13 `svg_2d_plot & x_label_on(bool cmd);`

Set to include an X-axis text label.

See Also:

`x_label_units`

Returns: Reference to `svg_2d_plot` to make chainable.

14 `bool x_label_on();`

Returns: true if to include an X-axis text label.

15 `svg_2d_plot & x_label_units(const std::string & str);`

Set the text to add units to the X-axis label.

16 `std::string x_label_units();`

Returns: the text to add units to the X-axis label. <The label will only be shown if `x_label_on() == true`.

17 `svg_2d_plot & x_label_units_on(bool cmd);`

Set true if want X axis label to include units (as well as label like "length"). <.

See Also:

`x_label_units` which also sets true.

18 `bool x_label_units_on();`

Set true if want X axis label to include units (as well as label like "length").

19 `svg_2d_plot & x_label_width(double width);`

Set the width (boldness) of X-axis label (including any units). (not recommended until browsers implement better).

20 `double x_label_width();`

Returns: the width (boldness) of X-axis label (including any units).

21 `svg_2d_plot & x_labels_strip_e0s(bool cmd);`

Set if to strip redundant zeros, signs and exponents, for example, reducing "1.2e+000" to "1.2" This markedly reduces visual clutter, and is the default.

12 `svg_2d_plot & x_major_grid_color(const svg_color & col);`

Set the color of X-axis major grid lines.

13 `svg_color x_major_grid_color();`

Set the color of X-axis major grid lines.

14 `svg_2d_plot & x_major_grid_on(bool is);`

If set true, will include a major X-axis grid.

15 `bool x_major_grid_on();`

Returns: true if will include a major X-axis grid.

16 `svg_2d_plot & x_major_grid_width(double w);`

Set the width of X-axis major grid lines.

17 `double x_major_grid_width();`

Returns: the color of X-axis major grid lines.

18 `svg_2d_plot & x_major_interval(double inter);`

Set the interval between X-axis major ticks.

19 `double x_major_interval();`

Returns: the interval between X-axis major ticks.

20 `svg_2d_plot & x_major_label_rotation(rotate_style rot);`

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

`rotate_style`

21 `rotate_style x_major_label_rotation();`

See Also:

`rotate_style`

Returns: rotation for X ticks major value labels.

22 `svg_2d_plot & x_major_labels_side(int);`

Set which side (up, down or none) for major ticks label values:

Returns: Reference to `svg_2d_plot` to make chainable.

23 `int x_major_labels_side();`

Returns: side for label values for major ticks.

24 `svg_2d_plot & x_major_tick(double d);`

Set interval (Cartesian units) between major ticks.

25 `double x_major_tick();`

Returns: interval (Cartesian units) between major ticks.

26 `svg_2d_plot & x_major_tick_color(const svg_color & col);`

Set the color of X-axis major ticks.

27 `svg_color x_major_tick_color();`

Returns: the color of X-axis major ticks.

28 `svg_2d_plot & x_major_tick_length(double length);`

Set length of X major ticks (SVG units, default pixels).

29 `double x_major_tick_length();`

Set length of X major ticks (SVG units, default pixels).

30 `svg_2d_plot & x_major_tick_width(double width);`

Set width of X major ticks (SVG units, default pixels).

31 `double x_major_tick_width();`

Set width of X major ticks (SVG units, default pixels).

32 `svg_2d_plot & x_max(double x);`

Set the maximum value on the X-axis.

33 `double x_max();`

autoscale set & get parameters, <Note: all these *MUST* precede x_autoscale(data) call.

Returns: the maximum value on the X-axis.

34 `svg_2d_plot & x_min(double min_x);`

Set the minimum value on the X-axis.

25 `double x_min();`

Returns: the minimum value on the X-axis.

26 `svg_2d_plot & x_min_ticks(int min_ticks);`

Set X-axis autoscale to include at least minimum number of ticks (default = 6).

27 `int x_min_ticks();`

Returns: X-axis autoscale minimum number of ticks.

28 `svg_2d_plot & x_minor_grid_color(const svg_color & col);`

Set the color of X-axis minor grid lines.

29 `svg_color x_minor_grid_color();`

Returns: the color of X-axis minor grid lines.

30 `svg_2d_plot & x_minor_grid_on(bool is);`

If set true, will include a minor X-axis grid.

31 `bool x_minor_grid_on();`

Returns: true if will include a major X-axis grid.

32 `svg_2d_plot & x_minor_grid_width(double w);`

Set the width of X-axis minor grid lines.

33 `double x_minor_grid_width();`

Returns: the width of X-axis minor grid lines.

34 `double x_minor_interval();`

Returns: interval between X minor ticks.

35 `svg_2d_plot & x_minor_interval(double interval);`

Set interval between X-axis minor ticks.

36 `svg_2d_plot & x_minor_tick_color(const svg_color & col);`

Set the color of X-axis minor ticks.

37 `svg_color x_minor_tick_color();`

Returns: the color of X-axis minor ticks.

28 `svg_2d_plot & x_minor_tick_length(double length);`

Set length of X minor ticks (SVG units, default pixels).

29 `double x_minor_tick_length();`

Returns: length of X minor ticks (SVG units, default pixels).

30 `svg_2d_plot & x_minor_tick_width(double width);`

Set width of X minor ticks (SVG units, default pixels).

31 `double x_minor_tick_width();`

Returns: width of X minor ticks (SVG units, default pixels).

32 `svg_2d_plot & x_num_minor_ticks(unsigned int num);`

Set number of X-axis minor ticks between major ticks.

33 `unsigned int x_num_minor_ticks();`

Returns: number of X-axis minor ticks between major ticks.

34 `svg_2d_plot & x_order_color(const svg_color & col);`

Set the color of X order #, for example, the color of #42.

35 `svg_color x_order_color();`

Returns: the color of X order #, for example, the color of #42.

36 `svg_2d_plot & x_order_on(bool b);`

Set true if to append append an order # to data point X values near data points markers.

37 `bool x_order_on();`

Returns: true if to append an order # to data point X values near data points markers.

38 `svg_2d_plot & x_plusminus_color(const svg_color & col);`

Set the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

39 `svg_color x_plusminus_color();`

Returns: the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

20 `svg_2d_plot & x_plusminus_on(bool b);`

Set if to append std_dev estimate to data point X values near data points markers.

21 `bool x_plusminus_on();`

Returns: true if to append std_dev estimate to data point X values near data points markers.

22 `const std::string x_prefix();`

Returns: the prefix.

23 `svg_2d_plot & x_range(double min_x, double max_x);`

Set the range of values on the X-axis. The minimum and maximum values must be finite and not too near to the minima or maxima that can be represented by floating point double, std::numeric_limits<double>::min() or std::numeric_limits<double>::max(), and the range must not be too small.

24 `std::pair< double, double > x_range();`

Returns: the range of values on the X-axis.

25 `const std::string x_separator();`

Returns: the separator, perhaps including Unicode.

26 `svg_2d_plot & x_size(unsigned int i);`

Set SVG image X-axis size (SVG units, default pixels).

27 `unsigned int x_size();`

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

28 `svg_2d_plot & x_steps(int steps);`

Set autoscale to set ticks in steps multiples of:

2,4,6,8,10, if 2

or 1,5,10 if 5

or 2,5,10 if 10.

default = 0 (none).



Note

: Must precede x_autoscale(data) call).

29 `int x_steps();`

Returns: autoscale to set ticks in steps.

20 `const std::string x_suffix();`

Returns: the suffix (only used if separator != "")

21 `ticks_labels_style & x_ticks();`

Returns: true if ticks are to be marked on the X-axis.

22 `svg_2d_plot & x_ticks_down_on(bool cmd);`

Set true if Y major ticks should mark upwards.

23 `bool x_ticks_down_on();`

Returns: true if Y major ticks should mark upwards.

24 `svg_2d_plot & x_ticks_on_window_or_axis(int);`

Returns: reference to `svg_2d_plot` to make chainable.

25 `int x_ticks_on_window_or_axis();`

Returns: if ticks on the plot window or on the X-axis.

26 `svg_2d_plot & x_ticks_up_on(bool cmd);`

Set true if X major ticks should mark upwards.

27 `bool x_ticks_up_on();`

Returns: true if X major ticks should mark upwards.

28 `svg_2d_plot & x_ticks_values_color(const svg_color & col);`

Set X axis tick value label color.

29 `svg_color x_ticks_values_color();`

Returns: X-axis ticks value label color.

30 `svg_2d_plot & x_ticks_values_font_family(const std::string & family);`

Set X ticks value label font family. Available fonts depend on the program rendering the SVG XML, usually a browser. The default font (usually "default_font") is used if a render program does not provide the font specified. These are probably usable:

```
"arial", "impact", "courier", "lucida console", "Lucida sans unicode", "verdana", "calibri", "century",
"lucida calligraphy", "tahoma", "vivaldi", "informal roman", "lucida handwriting", "lucida bright", "helvetica"
```

31 `const std::string & x_ticks_values_font_family();`

Returns: X ticks value label font family.

22 `svg_2d_plot & x_ticks_values_font_size(unsigned int i);`

Set X ticks value label font size (svg units, default pixels).

23 `unsigned int x_ticks_values_font_size();`

Set X ticks value label font size (svg units, default pixels).

24 `svg_2d_plot & x_ticks_values_ioflags(std::ios_base::fmtflags f);`

Set ostream format flags of data point X values near data points markers.

25 `std::ios_base::fmtflags x_ticks_values_ioflags();`

Returns: ostream format flags of data point X values near data points markers.

26 `svg_2d_plot & x_ticks_values_precision(int p);`

Set ostream decimal digits precision of data point X values near data points markers.

27 `int x_ticks_values_precision();`

Returns: ostream decimal digits precision of data point X values near data points markers.

28 `svg_2d_plot & x_tight(double tight);`

Set tolerance to autoscale to permit data points slightly outside both end ticks.

29 `double x_tight();`

Returns: tolerance given to autoscale to permit data points slightly outside both end ticks. <Get results of autoscaling.

30 `svg_2d_plot & x_value_font_size(unsigned int i);`

Set X tick value label font size (svg units, default pixels).

31 `unsigned int x_value_font_size();`

Returns: X tick value label font size (svg units, default pixels).

32 `svg_2d_plot & x_value_ioflags(std::ios_base::fmtflags flags);`

Set ostream std::ios::fmtflags for X value label (default decimal == 0X201). Mainly useful for changing to scientific, fixed or hexadecimal format. For example:

```
myplot.x_value_ioflags(std::ios::dec | std::ios::scientific)
```

23 `std::ios_base::fmtflags x_value_ioflags();`

Returns: stream std::ios::fmtflags for control of format of X value labels.

24 `svg_2d_plot & x_value_precision(int digits);`

Set precision of X-tick label values in decimal digits (default 3). 3 decimal digits precision is sufficient for small images. 4 or 5 decimal digits precision will give more cluttered plots. If the range of labels is very small, then more digits will be essential.

25 `int x_value_precision();`

Returns: Precision of X-tick label values in decimal digits

26 `svg_2d_plot & x_values_color(const svg_color & col);`

Set the color of data point X values near data points markers.

27 `svg_color x_values_color();`

Returns: the color of data point X values near data points markers.

28 `svg_2d_plot & x_values_font_family(const std::string & family);`

Set font family of data point X values near data points markers.

29 `const std::string & x_values_font_family();`

Returns: font family of data point X values near data points markers.

30 `svg_2d_plot & x_values_font_size(unsigned int i);`

Set font size of data point X values near data points markers.

31 `unsigned int x_values_font_size();`

Returns: font size of data point X values near data points markers.

32 `svg_2d_plot & x_values_ioflags(std::ios_base::fmtflags f);`

Set iostream format flags of data point X values near data points markers.

33 `std::ios_base::fmtflags x_values_ioflags();`

Returns: iostream format flags of data point X values near data points markers.

34 `svg_2d_plot & x_values_on(bool b);`

Set true to show data point values near data points markers.

35 `bool x_values_on();`

Returns: true if to show data point values near data points markers.

26 `svg_2d_plot & x_values_precision(int p);`

Set iostream decimal digits precision of data point X values near data points markers.

27 `int x_values_precision();`

Returns: iostream decimal digits precision of data point X values near data points markers.

28 `svg_2d_plot & x_values_rotation(rotate_style rotate);`

Returns: the rotation (rotate_style) of data point X values near data points markers.

29 `int x_values_rotation();`

Set the rotation (rotate_style) of data point X values near data points markers.

30 `svg_2d_plot & x_with_zero(bool b);`

Set X-axis autoscale to include zero (default = false).

31 `bool x_with_zero();`

Returns: true if X-axis autoscale to include zero (default = false).

32 `template<typename T> svg_2d_plot & xy_autoscale(const T & container);`

Whole data series to use to calculate autoscaled values for **both** X and Y axes.

Returns: reference to `svg_2d_plot` to make chainable.

33 `bool xy_autoscale();`

Returns: true if to autoscale both X and Y Axes.

34 `bool xy_values_on();`

Returns: true if values of X and Y data points are shown (for example: 1.23).

35 `svg_2d_plot & xy_values_on(bool b);`

Set true if values of X and Y data points are to be shown (as 1.23).

(Will override `x_values_on` and/or `y_values_on` that would otherwise cause overwriting).

Returns: reference to `svg_2d_plot` to make chainable.

36 `svg_2d_plot & y_addlimits_color(const svg_color & col);`

Set color of Y confidence interval.

Returns: Reference to `svg_2d_plot` to make chainable.

27 `const svg_color y_addlimits_color();`

Returns: Color of Y confidence interval.

28 `bool y_addlimits_on();`

Returns: true if values of Y data points are to include confidence interval.

29 `svg_2d_plot & y_addlimits_on(bool b);`

Set true if values of Y data points are to include confidence interval.

Returns: Reference to `svg_2d_plot` to make chainable.

30 `bool y_autoscale();`

Returns: true if to autoscale minimum and maximum for Y-axis.

31 `svg_2d_plot & y_autoscale(bool b);`

Set true if to autoscale minimum and maximum for Y-axis.

Returns: Reference to `svg_2d_plot` to make chainable.

32 `svg_2d_plot & y_autoscale(double minimum, double maximum);`

Set minimum & maximum Y values to use to autoscale Y-axis.

Parameters: `maximum` Value to use for autoscale that will be rounded up.
`um`

`minimum` Value to use for autoscale that will be rounded down.

Returns: reference to `svg_2d_plot` to make chainable.

33 `svg_2d_plot & y_autoscale(std::pair< double, double > p);`

Set Y min & max values as a **pair** to use to autoscale.

Returns: reference to `svg_2d_plot` to make chainable.

34 `template<typename T> svg_2d_plot & y_autoscale(const T & begin, const T & end);`

Data series using iterator's range to use to calculate autoscaled values.

Template Parameters: `T` an STL container: array, vector ...

Returns: reference to `svg_2d_plot` to make chainable.

35 `template<typename T> svg_2d_plot & y_autoscale(const T & container);`

Whole data series to use to calculate autoscaled values.

36 `axis_line_style & y_axis();`

Returns: true if vertical Y-axis line to be drawn.

37 `svg_2d_plot & y_axis_color(const svg_color & col);`

Set Y-axis linecolor. (set only stroke color).

Returns: Reference to `svg_2d_plot` to make chainable.

38 `svg_color y_axis_color();`

Returns: Ty_axis stroke color.

39 `svg_2d_plot & y_axis_label_color(const svg_color & col);`

Set y_axis stroke color.



Note

Setting the stroke color may produce fuzzy characters :-)

Returns: Reference to `svg_2d_plot` to make chainable.

40 `svg_color y_axis_label_color();`

Returns: Y-axis label stroke color.

41 `svg_2d_plot & y_axis_on(bool is);`

If set true, draw a vertical Y-axis line.

42 `bool y_axis_on();`

Returns: true if will draw a horizontal X-axis line.

43 `const std::string y_axis_position();`

Returns: Text information about Y-axis position.

44 `svg_2d_plot & y_axis_value_color(const svg_color & col);`

Set color of Y-axis **value** labels.

Returns: Reference to `svg_2d_plot` to make chainable.

45 `svg_color y_axis_value_color();`

Returns: Color of Y-axis tick value labels.

46 `svg_2d_plot & y_axis_width(double width);`

Set width of Y-axis line.

Returns: Reference to `svg_2d_plot` to make chainable.

37 `double y_axis_width();`

Returns: Width of Y-axis line.

38 `svg_2d_plot & y_decor(const std::string & pre, const std::string & sep = "", const std::string & suf = "");`

Set prefix, separator and suffix for Y-axis.

Example:

```
my_1d_plot.x_decor("[ x = ", "", "\u00A0;sec"]);
```



Note

If you want a space, you must use a **Unicode** space.

Returns: Reference to `svg_2d_plot` to make chainable.

39 `svg_2d_plot & y_df_color(const svg_color & col);`

Set color of Y degrees of freedom.

Returns: Reference to `svg_2d_plot` to make chainable.

40 `const svg_color y_df_color();`

Returns: Color of Y degrees of freedom.

41 `bool y_df_on();`

Returns: true if values of Y data points are to include degrees of freedom estimates.

42 `svg_2d_plot & y_df_on(bool b);`

Set true if values of Y data points are to include degrees of freedom estimates.

Returns: Reference to `svg_2d_plot` to make chainable.

43 `svg_2d_plot & y_label(const std::string & str);`

Set the text for the Y-axis label (and set `y_label_on(true)`).

44 `std::string y_label();`

Returns: the text for the Y-axis label. < The label will only be shown if `y_label_on() == true`.

45 `svg_2d_plot & y_label_axis(const std::string & str);`

Set text to label Y-axis.

Returns: Reference to `svg_2d_plot` to make chainable.

36 `std::string y_label_axis();`

Returns: text to label Y-axis.

37 `svg_2d_plot & y_label_color(const svg_color & col);`

Set the color of Y-axis label (including any units).

38 `svg_color y_label_color();`

Returns: the color of Y-axis label (including any units).

39 `svg_2d_plot & y_label_font_family(const std::string & family);`

Set Y-axis label text font family (for example: "Lucida Sans Unicode"). Available fonts depend on the program rendering the SVG XML, usually a browser. The default font (usually "Lucida Sans Unicode") is used if a renderer (in a browser or a converter to PDF like RenderX) does not provide the font specified. A Unicode font has a better chance of providing Unicode symbols, for example, specified as ∞. These fonts are probably usable:

```
"arial", "impact", "courier", "lucida console", "Lucida Sans Unicode", "Verdana", "calibri", "century",
"lucida calligraphy", "tahoma", "vivaldi", "informal roman", "lucida handwriting", "lucida bright", "helvetica"
```

Returns: Reference to `svg_2d_plot` to make chainable.

40 `const std::string & y_label_font_family();`

Returns: Font family for label on Y-axis.

41 `svg_2d_plot & y_label_font_size(unsigned int i);`

Set Y-axis label text font size.

42 `unsigned int y_label_font_size();`

Returns: Y-axis label text font size.

43 `svg_2d_plot & y_label_on(bool cmd);`

Set true if to label Y-axis with name (and units).

Returns: Reference to `svg_2d_plot` to make chainable.

44 `bool y_label_on();`

Returns: true if Y-axis is to labelled.

45 `svg_2d_plot & y_label_units(const std::string & str);`

Set the text to add units to the Y-axis label.

36 `std::string y_label_units();`

Returns: the text to add units to the X-axis label.

37 `svg_2d_plot & y_label_units_on(bool b);`

Set true to add **units** text to the Y-axis label.

See Also:

`svg_2d_plot::y_label_units`

Returns: Reference to `svg_2d_plot` to make chainable.

38 `bool y_label_units_on();`

Returns: true if to add units text to the Y-axis label.

39 `svg_2d_plot & y_label_weight(std::string s);`

Set Y-axis label text font weight (for example: "bold"). ("bold" is only one that works so far, and quality may be poor for some browsers).

Returns: Reference to `svg_2d_plot` to make chainable.

40 `const std::string & y_label_weight();`

Returns: Y-axis label text font weight (for example: "bold").

41 `svg_2d_plot & y_label_width(double width);`

Set width of Y-axis value labels.

Returns: Reference to `svg_2d_plot` to make chainable.

42 `double y_label_width();`

Returns: Width of Y-axis value labels.

43 `svg_2d_plot & y_labels_strip_e0s(bool cmd);`

If true then strip unnecessary zeros, signs from labels.

Returns: Reference to `svg_2d_plot` to make chainable.

44 `bool y_labels_strip_e0s();`

Returns: true if to strip unnecessary zeros, signs from labels.

45 `svg_2d_plot & y_major_grid_color(const svg_color & col);`

Set color of Y major grid lines.

Returns: Reference to `svg_2d_plot` to make chainable.

36 `const svg_color y_major_grid_color();`

Returns: Color of Y major grid lines.

37 `svg_2d_plot & y_major_grid_on(bool is);`

Set true to include major grid lines.

Returns: Reference to `svg_2d_plot` to make chainable.

38 `bool y_major_grid_on();`

Returns: true to include major grid lines.

39 `svg_2d_plot & y_major_grid_width(double width);`

Set width of major grid lines.

Returns: Reference to `svg_2d_plot` to make chainable.

40 `double y_major_grid_width();`

Set width of major grid lines.

41 `double y_major_interval();`

Returns: major interval between ticks on Y-axis.

42 `svg_2d_plot & y_major_interval(double inter);`

Set major interval between ticks on Y-axis.

Returns: reference to `svg_2d_plot` to make chainable.

43 `svg_2d_plot & y_major_label_rotation(rotate_style rot);`

Rotation or orientation of labels for major ticks on vertical Y-axis line.

See Also:

`rotate_style` for possible values: horizontal, uphill...

Parameters: `rot` Default orientation is horizontal.

Returns: Reference to `svg_2d_plot` to make chainable.

44 `int y_major_label_rotation();`

Returns: Rotation of Y-axis major tick labels.

35 `svg_2d_plot & y_major_labels_side(int place);`

Position of labels for major ticks on vertical Y-axis line.

- `side > 0` label to left of Y-axis line (default),
- `side = 0` (false) means no major tick labels on Y-axis.
- `side > 0` means to right of Y-axis line.

Returns: Reference to `svg_2d_plot` to make chainable.

36 `int y_major_labels_side();`

Returns: Position of labels (if any) for major ticks on vertical Y-axis line.

37 `svg_2d_plot & y_major_tick_color(const svg_color & col);`

Set color of Y major tick lines.

Returns: Reference to `svg_2d_plot` to make chainable.

38 `const svg_color y_major_tick_color();`

Returns: Color of Y major tick lines.

39 `double y_major_tick_length();`

Returns: Major tick length on Y-axis.

40 `svg_2d_plot & y_major_tick_length(double length);`

Set major tick length on Y-axis.

Returns: Reference to `svg_2d_plot` to make chainable.

41 `svg_2d_plot & y_major_tick_width(double width);`

Set width of major ticks on Y-axis.

Returns: reference to `svg_2d_plot` to make chainable.

42 `double y_major_tick_width();`

Returns: Width of major ticks on Y-axis.

43 `double y_max();`

Returns: Maximum for Y-axis.

44 `double y_min();`

Returns: Minimum for Y-axis.

35 `svg_2d_plot & y_minor_grid_color(const svg_color & col);`

Set color of Y minor grid lines.

Returns: Reference to `svg_2d_plot` to make chainable.

36 `const svg_color y_minor_grid_color();`

Returns: Color of Y minor grid lines.

37 `svg_2d_plot & y_minor_grid_on(bool is);`

Set true to include minor grid lines.

Returns: Reference to `svg_2d_plot` to make chainable.

38 `bool y_minor_grid_on();`

Set true to include minor grid lines.

39 `svg_2d_plot & y_minor_grid_width(double width);`

Set width of minor grid lines.

Returns: reference to `svg_2d_plot` to make chainable.

40 `double y_minor_grid_width();`

Returns: Width of minor grid lines.

41 `double y_minor_interval();`

Returns: interval between Y minor ticks.

42 `svg_2d_plot & y_minor_tick_color(const svg_color & col);`

Set color of Y minor tick lines.

Returns: Reference to `svg_2d_plot` to make chainable.

43 `const svg_color y_minor_tick_color();`

Returns: Color of Y minor tick lines.

44 `svg_2d_plot & y_minor_tick_length(double length);`

Set minor tick length on Y-axis.

Returns: Reference to `svg_2d_plot` to make chainable.

45 `double y_minor_tick_length();`

Returns: Minor tick length on Y-axis.

36 `svg_2d_plot & y_minor_tick_width(double width);`

Set width of minor ticks on Y-axis.

Returns: Reference to `svg_2d_plot` to make chainable.

37 `double y_minor_tick_width();`

Returns: Width of minor ticks on Y-axis.

38 `svg_2d_plot & y_num_minor_ticks(unsigned int num);`

Set number of minor ticks on Y-axis.

Returns: Reference to `svg_2d_plot` to make chainable.

39 `unsigned int y_num_minor_ticks();`

Returns: Number of minor ticks on Y-axis.

40 `svg_2d_plot & y_plusminus_color(const svg_color & col);`

Set color of Y uncertainty of value.

Returns: Reference to `svg_2d_plot` to make chainable.

Returns: Reference to `svg_2d_plot` to make chainable.

41 `const svg_color y_plusminus_color();`

Returns: Color of Y uncertainty of value.

42 `bool y_plusminus_on();`

Returns: true if values of Y data points are to include uncertainty estimates.

43 `svg_2d_plot & y_plusminus_on(bool b);`

Set true if values of Y data points are to include uncertainty estimates.

44 `const std::string y_prefix();`

Get the prefix (only used if separator != "")

45 `svg_2d_plot & y_range(double min_y, double max_y);`

Set the range (max and min) for Y-axis from the parameters provided.

Returns: Reference to `svg_2d_plot` to make chainable.

36 `std::pair< double, double > y_range();`

Returns: The range (max and min) for Y-axis.

37 `const std::string y_separator();`

Get separator (also controls use of the prefix & suffix - they are only used if separator != "").

Note



For a space, you must use a Unicode space

`"\u00A0;"`

rather than " ".

38 `unsigned int y_size();`

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

39 `svg_2d_plot & y_size(unsigned int i);`

Set SVG image Y-axis size (SVG units, default pixels).

40 `const std::string y_suffix();`

Get the suffix (only used if separator != "")

41 `ticks_labels_style & y_ticks();`

Returns: true if ticks are to be marked on the Y-axis.

42 `svg_2d_plot & y_ticks_left_on(bool cmd);`

Set true if ticks on the Y-axis are to be on left of axis line.

Returns: Reference to `svg_2d_plot` to make chainable.

43 `bool y_ticks_left_on();`

Returns: true if ticks on the Y-axis are to be on left of axis line.

44 `svg_2d_plot & y_ticks_on_window_or_axis(int);`

Set Y ticks on window or axis

Returns: reference to `svg_2d_plot` to make chainable.

45 `int y_ticks_on_window_or_axis();`

Returns: true if Y-axis ticks wanted on the window (rather than on axis).
 -1 left of plot window, 0 on Y-axis, +1 right of plot window.

36 `svg_2d_plot & y_ticks_right_on(bool cmd);`

Set true if ticks on the Y-axis are to be on right of axis line.

Returns: Reference to `svg_2d_plot` to make chainable.

37 `bool y_ticks_right_on();`

Returns: true if ticks on the Y-axis are to be on right of axis line.

38 `svg_2d_plot & y_ticks_values_color(const svg_color & col);`

Set color for Y_axis tick values.

Returns: Reference to `svg_2d_plot` to make chainable.

39 `svg_color y_ticks_values_color();`

Returns: color for Y-axis ticks values.

40 `svg_2d_plot & y_ticks_values_font_family(const std::string & family);`

Set font family for Y-axis ticks values. Available fonts depend on the program rendering the SVG XML, usually a browser. The default font (usually "verdana") is used if a render program does not provide the font specified.

These are probably usable:

```
"arial", "impact", "courier", "lucida console", "Lucida sans unicode", "verdana", "calibri", "century",
"lucida calligraphy", "tahoma", "vivaldi", "informal roman", "lucida handwriting", "lucida bright", "helvetica"
```

Returns: Reference to `svg_2d_plot` to make chainable.

41 `const std::string & y_ticks_values_font_family();`

Returns: font family for Y-axis ticks values.

42 `svg_2d_plot & y_ticks_values_font_size(unsigned int i);`

Set font size for Y-axis ticks values (svg units, default pixels).

Returns: Reference to `svg_2d_plot` to make chainable.

43 `unsigned int y_ticks_values_font_size();`

Returns: Font size for Y-axis values.

44 `svg_2d_plot & y_ticks_values_ioflags(std::ios_base::fmtflags f);`

Set `std::iostream` format flags of ticks Y values. Useful to set hexadecimal, fixed and scientific, (`std::ios::scientific`).

45 `std::ios_base::fmtflags y_ticks_values_iflags();`

Returns: std::iostream format flags of ticks Y values. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

46 `svg_2d_plot & y_ticks_values_precision(int p);`

Set std::iostream decimal digits precision of ticks Y values.

47 `int y_ticks_values_precision();`

Returns: std::iostream decimal digits precision of ticks Y values..

48 `svg_2d_plot & y_value_iflags(std::ios_base::fmtflags flags);`

Set std::ioflags of Y tick label values (default 0x201 == dec).

Example:

```
my_plot.x_value_iflags(ios::dec | ios::scientific).x_value_precision(2);
```

Returns: Reference to `svg_2d_plot` to make chainable.

49 `int y_value_iflags();`

Returns: All stream ioflags for control of format of Y tick value labels.

50 `svg_2d_plot & y_value_precision(int digits);`

Set precision of Y tick label values in decimal digits (default 3).

Example:

```
my_plot.x_value_iflags(ios::dec | ios::scientific).x_value_precision(2);
```

Returns: Reference to `svg_2d_plot` to make chainable.

51 `int y_value_precision();`

Returns: Precision of Y tick value labels in decimal digits (default 3).

52 `svg_2d_plot & y_values_color(const svg_color & col);`

Set color for Y-axis values.

Returns: reference to `svg_2d_plot` to make chainable.

53 `svg_color y_values_color();`

Returns: Color for Y-axis values.

44 `svg_2d_plot & y_values_font_family(const std::string & family);`

Set font family for Y-axis values. Available fonts depend on the program rendering the SVG XML, usually a browser. The default font (usually "verdana") is used if a render program does not provide the font specified. These are probably usable: "arial", "impact", "courier", "lucida console", "Lucida sans unicode", "verdana", "calibri", "century", "lucida calligraphy", "tahoma", "vivaldi", "informal roman", "lucida handwriting", "lucida bright", "helvetica"

Returns: Reference to `svg_2d_plot` to make chainable.

45 `const std::string & y_values_font_family();`

Returns: Font family for Y-axis values.

46 `svg_2d_plot & y_values_font_size(unsigned int i);`

Set font size for Y-axis values.

Returns: Reference to `svg_2d_plot` to make chainable.

47 `unsigned int y_values_font_size();`

Returns: Font size for Y-axis values.

48 `svg_2d_plot & y_values_ioflags(std::ios_base::fmtflags f);`

Set iostream format flags for data point values.

Returns: reference to `svg_2d_plot` to make chainable.

49 `std::ios_base::fmtflags y_values_ioflags();`

Returns: std::iostream format flags for data point values.

50 `bool y_values_on();`

(Will override `xy_values_on` that would otherwise cause overwriting). So the last `values_on` setting will prevail.

Returns: true if values of Y data points are shown (for example: 1.23).

51 `svg_2d_plot & y_values_on(bool b);`

Set true if values of Y data points are shown (for example: 1.23, 2.34).

Returns: Reference to `svg_2d_plot` to make chainable.

52 `svg_2d_plot & y_values_precision(int p);`

Set iostream precision for data points Y values.

Returns: Reference to `svg_2d_plot` to make chainable.

53 `int y_values_precision();`

Returns: `iostream` precision for data points Y values.

44 `svg_2d_plot & y_values_rotation(rotate_style rotate);`

Set rotation for value labels on Y-axis ticks.

See Also:

`rotate_style`

Returns: Reference to `svg_2d_plot` to make chainable.

45 `int y_values_rotation();`

Returns: rotation for value labels on Y-axis.

Class `svg_2d_plot_series`

`boost::svg::svg_2d_plot_series` — Holds a series of 2D data values (points) to be plotted.

Synopsis

```
// In header: <boost/svg_plot/svg_2d_plot.hpp>

class svg_2d_plot_series {
public:
    // construct/copy/destruct
    template<typename T> svg_2d_plot_series(T, T, std::string = "");

    // public member functions
    svg_2d_plot_series & area_fill(const svg_color &);
    svg_color & area_fill();
    svg_2d_plot_series & bar_area_fill(const svg_color &);
    svg_color & bar_area_fill();
    svg_2d_plot_series & bar_color(const svg_color &);
    svg_color & bar_color();
    svg_2d_plot_series & bar_opt(bar_option);
    bar_option bar_opt();
    svg_2d_plot_series & bar_width(double);
    double bar_width();
    svg_2d_plot_series & bezier_on(bool);
    bool bezier_on();
    svg_2d_plot_series & fill_color(const svg_color &);
    svg_2d_plot_series & histogram(histogram_option);
    int limits_count();
    svg_2d_plot_series & line_color(const svg_color &);
    svg_color & line_color();
    svg_2d_plot_series & line_on(bool);
    bool line_on();
    plot_line_style line_style();
    svg_2d_plot_series & line_width(double);
    double line_width();
    svg_2d_plot_series & shape(point_shape);
    point_shape shape();
    svg_2d_plot_series & size(int);
    int size();
    svg_2d_plot_series & stroke_color(const svg_color &);
    int values_count();
};
```

Description

Data values are sorted into normal and 'at limits': NaN, infinity or too small or too large.

Member functions allow control of data points markers and lines joining them, and their appearance, shape, color and size.

Data points can include their value, and optionally uncertainty and optionally number of degrees of freedom.

Each data series can have a title that can be shown on a legend box with identifying symbols.

`std::multimap` is used rather than `std::vector` of `std::pair`'s because `std::multimap` sorts and ensures that lines joining data points are unaffected by the order in which data is presented. (For 1-D a vector of doubles can be used).

svg_2d_plot_series public construct/copy/destruct

1. `template<typename T>`
`svg_2d_plot_series(T begin, T end, std::string title = "");`

Constructor for a data series to plot

Parameters:

begin	Starting iterator into container of data series begin() to start at the beginning.
end	Ending iterator into container of data series, end() to finish with the last item.
title	Title for the plot.

Template Parameters:

T	an STL container: for example: <code>multimap</code> .
---	--

svg_2d_plot_series public member functions

1. `svg_2d_plot_series & area_fill(const svg_color & col_);`

Set Data series area fill color. \note @c area_fill(false) will produce a @b blank color, and so NO FILL.

`area_fill(blank)` will produce the default non-blank color (black?).

2. `svg_color & area_fill();`

Returns: Color for any area fill below line(s) joining data points.

3. `svg_2d_plot_series & bar_area_fill(const svg_color & col);`

Parameters: col Set bar area fill color.

Returns: Reference to `svg_2d_plot_series` to make chainable.

4. `svg_color & bar_area_fill();`

Returns: Color of bar area fill.

5. `svg_2d_plot_series & bar_color(const svg_color & col);`

Set bar color.

Returns: Reference to `svg_2d_plot_series` to make chainable.

6. `svg_color & bar_color();`

Returns: Bar color.

7. `svg_2d_plot_series & bar_opt(bar_option);`

Set bar options.

Returns: Reference to `svg_2d_plot_series` to make chainable.

8. `bar_option bar_opt();`

Returns: Bar options.

9. `svg_2d_plot_series & bar_width(double wid_);`

Set Bar width.

Returns: Reference to `svg_2d_plot_series` to make chainable.

10. `double bar_width();`

Returns: Bar width.

11. `svg_2d_plot_series & bezier_on(bool on_);`

Set true to draw bezier curved line linking data points.

Returns: Reference to `svg_2d_plot_series` to make chainable.

12. `bool bezier_on();`

Returns: true if line joining data points should be a bezier curve.

13. `svg_2d_plot_series & fill_color(const svg_color & col_);`

Set data series point marker fill color.

14. `svg_2d_plot_series & histogram(histogram_option opt_);`

Parameters: `opt_` no_histogram = 0,

bar = +1 // Stick or column line (stroke width) vertical to X-axis.

Returns: Reference to `svg_2d_plot_series` to make chainable.

15. `int limits_count();`

Returns: number of values 'at limit' in a data series.

16. `svg_2d_plot_series & line_color(const svg_color & col_);`

Set Data series line color.

17. `svg_color & line_color();`

Returns: color of a line to join data points.

18. `svg_2d_plot_series & line_on(bool on_);`

Set true to draw line linking data points.

Returns: Reference to `svg_2d_plot_series` to make chainable.

19. `bool line_on();`

Returns: true if a line is to join data points.

20. `plot_line_style line_style();`

Returns: Line style for line joining data points.

21. `svg_2d_plot_series & line_width(double wid_);`

Set data series line width.

(Sets legend line width too).

Returns: Reference to `svg_2d_plot_series` to make chainable.

22. `double line_width();`

Returns: Width of line joining data points.

23. `svg_2d_plot_series & shape(point_shape shape_);`

Set Data series point marker shape.

24. `point_shape shape();`

Returns: Shape of data point marker(s).

25. `svg_2d_plot_series & size(int size_);`

Set Data series point marker size.

26. `int size();`

Returns: Size of data point marker(s).

27. `svg_2d_plot_series & stroke_color(const svg_color & col_);`

Set Data series point marker stroke color.

28. `int values_count();`

Returns: number of normal values in a data series.

Header <boost/svg_plot/svg_boxplot.hpp>

Create box plots in Scalable Vector Graphic (SVG) format.

Provides `svg_boxplot` data and functions to create plots, and `svg_boxplot_series` to allow data values to be added to the boxplot. Very many functions allow fine control of the appearance and layout of plots and data markers. (Items common to 1D, 2D and boxplot use `axis_plot_frame`).

A convenient way of graphically depicting groups of numerical data through their five-number summaries.

Show 1st quartile, median and 3rd quartile as a box. <http://en.wikipedia.org/wiki/Boxplot>

See Also:

Some Implementations of the Boxplot: Michael Frigge, David C. Hoaglin and Boris Iglewicz *The American Statistician*, Vol. 43, No. 1 (Feb., 1989), pp. 50-54

The Bagplot: A Bivariate Boxplot Peter J. Rousseeuw, Ida Ruts and John W. Tukey *The American Statistician*, Vol. 53, No. 4 (Nov., 1999), pp. 382-387

Jacob Voytko & Paul A. Bristow

```
namespace boost {
    namespace svg {
        class svg_boxplot;
        class svg_boxplot_series;
    }
}
```

Class `svg_boxplot`

`boost::svg::svg_boxplot` — A plot that can display boxplots of several data series.

Holds all info about the plot (but not any data series - see `svg_boxplot_series`)

Synopsis

```
// In header: <boost/svg_plot/svg_boxplot.hpp>

class svg_boxplot {
public:
    // construct/copy/destruct
    svg_boxplot();

    // public member functions
    bool autoscale();
    svg_boxplot & autoscale(bool);
    svg_boxplot & autoscale_check_limits(bool);
    bool autoscale_check_limits();
    svg_boxplot & autoscale_plusminus(double);
    double autoscale_plusminus();
    svg_boxplot & axes_on(bool);
    bool axes_on();
    svg_boxplot & axis_color(const svg_color &);

    svg_color axis_color();
    svg_boxplot & axis_width(double);
    double axis_width();
    svg_boxplot & background_border_color(const svg_color &);

    svg_color background_border_color();
    svg_boxplot & background_border_width(double);
    double background_border_width();
    svg_boxplot & background_color(const svg_color &);

    svg_color background_color();
    svg_boxplot & boost_license_on(bool);
    bool boost_license_on();
    svg_boxplot & box_border(const svg_color &);

    svg_color box_border();
    svg_boxplot & box_fill(const svg_color &);

    svg_color box_fill();
    svg_boxplot & box_width(double);
    double box_width();
    svg_boxplot & confidence(double);
    double confidence();
    svg_boxplot & coord_precision(int);
    int coord_precision();
    svg_boxplot & copyright_date(const std::string &);

    const std::string copyright_date();
    svg_boxplot & copyright_holder(const std::string &);

    const std::string copyright_holder();
    svg_boxplot & data_lines_width(double);
    double data_lines_width();
    svg_boxplot & derived();
    const svg_boxplot & derived() const;
    svg_boxplot & description(const std::string &);

    const std::string & description();
    svg_boxplot & document_title(const std::string &);

    std::string document_title();
    svg_boxplot &
    draw_line(double, double, double, const svg_color & = black);
    svg_boxplot &
    draw_note(double, double, std::string, rotate_style = horizontal,
              align_style = center_align, const svg_color & = black,
              text_style & = no_style);
    svg_boxplot &
    draw_plot_curve(double, double, double, double, double,
                   const svg_color & = black);
```

```

svg_boxplot &
draw_plot_line(double, double, double, double, const svg_color & = black);
svg_boxplot & extreme_outlier_color(const svg_color &);
svg_color extreme_outlier_color();
svg_boxplot & extreme_outlier_fill(const svg_color &);
svg_color extreme_outlier_fill();
svg_boxplot & extreme_outlier_shape(point_shape);
point_shape extreme_outlier_shape();
svg_boxplot & extreme_outlier_size(int);
int extreme_outlier_size();
svg_boxplot & extreme_outlier_values_on(bool);
bool extreme_outlier_values_on();
svg_boxplot & image_border_margin(double);
double image_border_margin();
svg_boxplot & image_border_width(double);
double image_border_width();
unsigned int image_x_size();
svg_boxplot & image_x_size(unsigned int);
unsigned int image_y_size();
svg_boxplot & image_y_size(unsigned int);
svg_boxplot & legend_background_color(const svg_color &);
svg_color legend_background_color();
svg_boxplot & legend_border_color(const svg_color &);
svg_color legend_border_color();
const std::pair< double, double > legend_bottom_right();
bool legend_box_fill_on();
svg_boxplot & legend_color(const svg_color &);
svg_color legend_color();
svg_boxplot & legend_font_family(const std::string &);
const std::string & legend_font_family();
svg_boxplot & legend_font_weight(const std::string &);
const std::string & legend_font_weight();
svg_boxplot & legend_header_font_size(int);
int legend_header_font_size();
svg_boxplot & legend_lines(bool);
bool legend_lines();
svg_boxplot & legend_on(bool);
bool legend_on();
bool legend_outside();
svg_boxplot & legend_place(legend_places);
legend_places legend_place();
svg_boxplot & legend_title(const std::string);
const std::string legend_title();
svg_boxplot & legend_title_font_size(unsigned int);
unsigned int legend_title_font_size();
svg_boxplot & legend_top_left(double, double);
const std::pair< double, double > legend_top_left();
svg_boxplot & legend_width(double);
double legend_width();
svg_boxplot &
license(std::string = "permits", std::string = "permits",
       std::string = "requires", std::string = "permits",
       std::string = "permits");
const std::string license_attribution();
const std::string license_commercialuse();
const std::string license_distribution();
svg_boxplot & license_on(bool);
bool license_on();
const std::string license_reproduction();
svg_boxplot & limit_color(const svg_color &);
svg_color limit_color();
svg_boxplot & limit_fill_color(const svg_color &);
svg_color limit_fill_color();

```

```

svg_boxplot & median_color(const svg_color &);
svg_color median_color();
svg_boxplot & median_values_on(bool);
bool median_values_on();
svg_boxplot & median_width(double);
double median_width();
svg_boxplot & one_sd_color(const svg_color &);
svg_color one_sd_color();
svg_boxplot & outlier_color(const svg_color &);
svg_color outlier_color();
svg_boxplot & outlier_fill(const svg_color &);
svg_color outlier_fill();
svg_boxplot & outlier_shape(point_shape);
point_shape outlier_shape();
svg_boxplot & outlier_size(int);
int outlier_size();
svg_boxplot & outlier_style(plot_point_style &);
plot_point_style & outlier_style();
svg_boxplot & outlier_values_on(bool);
bool outlier_values_on();
template<typename T>
svg_boxplot_series &
plot(const T &, const std::string &, double, svg_style, svg_style,
     svg_style, double, svg_style, svg_style, plot_point_style,
     plot_point_style, int, value_style, text_style);
template<typename T>
svg_boxplot_series & plot(const T &, const std::string & = "");
svg_boxplot & plot_background_color(const svg_color &);
svg_color plot_background_color();
svg_boxplot & plot_border_color(const svg_color &);
svg_color plot_border_color();
svg_boxplot & plot_border_width(double);
double plot_border_width();
svg_boxplot & plot_window_on(bool);
bool plot_window_on();
svg_boxplot & plot_window_x(double, double);
std::pair< double, double > plot_window_x();
double plot_window_x_left();
double plot_window_x_right();
svg_boxplot & plot_window_y(double, double);
std::pair< double, double > plot_window_y();
double plot_window_y_bottom();
double plot_window_y_top();
svg_boxplot & quartile_definition(int);
int quartile_definition();
std::pair< double, double > size();
svg_boxplot & size(unsigned int, unsigned int);
svg_boxplot & three_sd_color(const svg_color &);
svg_color three_sd_color();
svg_boxplot & title(const std::string);
svg_boxplot & title(const std::string &);
std::string title();
svg_boxplot & title_color(const svg_color &);
svg_color title_color();
svg_boxplot & title_font_alignment(align_style);
align_style title_font_alignment();
svg_boxplot & title_font_decoration(const std::string &);
const std::string & title_font_decoration();
svg_boxplot & title_font_family(const std::string &);
const std::string & title_font_family();
svg_boxplot & title_font_rotation(rotate_style);
int title_font_rotation();
svg_boxplot & title_font_size(unsigned int);

```

```

unsigned int title_font_size();
svg_boxplot & title_font_stretch(const std::string &);
const std::string & title_font_stretch();
svg_boxplot & title_font_style(const std::string &);
const std::string & title_font_style();
svg_boxplot & title_font_weight(const std::string &);
const std::string & title_font_weight();
bool title_on();
svg_boxplot & title_on(bool);
svg_boxplot & title_size(unsigned int);
svg_boxplot & two_sd_color(const svg_color &);
svg_color two_sd_color();
svg_boxplot & whisker_length(double);
double whisker_length();
svg_boxplot & write(const std::string &);
svg_boxplot & write(std::ostream &);
svg_boxplot & x_addlimits_color(const svg_color &);
svg_color x_addlimits_color();
svg_boxplot & x_addlimits_on(bool);
bool x_addlimits_on();
double x_auto_max_value();
double x_auto_min_value();
double x_auto_tick_interval();
int x_auto_ticks();
bool x_autoscale();
svg_boxplot & x_autoscale(bool);
svg_boxplot & x_autoscale(std::pair< double, double >);
svg_boxplot & x_autoscale(const T &);
svg_boxplot & x_autoscale(const T &, const T &);
svg_boxplot & x_axis_color(const svg_color &);
svg_color x_axis_color();
svg_boxplot & x_axis_label_color(const svg_color &);
svg_color x_axis_label_color();
svg_boxplot & x_axis_on(bool);
bool x_axis_on();
svg_boxplot & x_axis_position(int);
double x_axis_position();
svg_boxplot & x_axis_vertical(double);
bool x_axis_vertical();
svg_boxplot & x_axis_width(double);
double x_axis_width();
svg_boxplot & x_datetime_color(const svg_color &);
svg_color x_datetime_color();
svg_boxplot & x_datetime_on(bool);
bool x_datetime_on();
svg_boxplot &
x_decor(const std::string &, const std::string & = "",  

       const std::string & = "");  

svg_boxplot & x_df_color(const svg_color &);
svg_color x_df_color();
svg_boxplot & x_df_on(bool);
bool x_df_on();
svg_boxplot & x_id_color(const svg_color &);
svg_color x_id_color();
svg_boxplot & x_id_on(bool);
bool x_id_on();
std::string x_label();
svg_boxplot & x_label(const std::string &);
svg_boxplot & x_label_color(const svg_color &);
svg_color x_label_color();
svg_boxplot & x_label_font_family(const std::string &);
const std::string & x_label_font_family();
svg_boxplot & x_label_font_size(unsigned int);

```

```

unsigned int x_label_font_size();
svg_boxplot & x_label_on(bool);
bool x_label_on();
svg_boxplot & x_label_size(unsigned int);
std::string x_label_text();
svg_boxplot & x_label_units(const std::string &);
std::string x_label_units();
svg_boxplot & x_label_units_on(bool);
bool x_label_units_on();
svg_boxplot & x_label_width(double);
double x_label_width();
svg_boxplot & x_labels_strip_e0s(bool);
svg_boxplot & x_major_grid_color(const svg_color &);
svg_color x_major_grid_color();
svg_boxplot & x_major_grid_on(bool);
bool x_major_grid_on();
svg_boxplot & x_major_grid_width(double);
double x_major_grid_width();
svg_boxplot & x_major_interval(double);
double x_major_interval();
svg_boxplot & x_major_label_rotation(rotate_style);
rotate_style x_major_label_rotation();
int x_major_labels();
svg_boxplot & x_major_labels_on(int);
svg_boxplot & x_major_labels_side(int);
int x_major_labels_side();
svg_boxplot & x_major_tick(double);
double x_major_tick();
svg_boxplot & x_major_tick_color(const svg_color &);
svg_color x_major_tick_color();
svg_boxplot & x_major_tick_length(double);
double x_major_tick_length();
svg_boxplot & x_major_tick_width(double);
double x_major_tick_width();
svg_boxplot & x_max(double);
double x_max();
svg_boxplot & x_min(double);
double x_min();
svg_boxplot & x_min_ticks(int);
int x_min_ticks();
svg_boxplot & x_minor_grid_color(const svg_color &);
svg_color x_minor_grid_color();
svg_boxplot & x_minor_grid_on(bool);
bool x_minor_grid_on();
svg_boxplot & x_minor_grid_width(double);
double x_minor_grid_width();
double x_minor_interval();
svg_boxplot & x_minor_interval(double);
svg_boxplot & x_minor_tick_color(const svg_color &);
svg_color x_minor_tick_color();
svg_boxplot & x_minor_tick_length(double);
double x_minor_tick_length();
svg_boxplot & x_minor_tick_width(double);
double x_minor_tick_width();
svg_boxplot & x_num_minor_ticks(unsigned int);
unsigned int x_num_minor_ticks();
svg_boxplot & x_order_color(const svg_color &);
svg_color x_order_color();
svg_boxplot & x_order_on(bool);
bool x_order_on();
svg_boxplot & x_plusminus_color(const svg_color &);
svg_color x_plusminus_color();
svg_boxplot & x_plusminus_on(bool);

```

```

bool x_plusminus_on();
const std::string x_prefix();
svg_boxplot & x_range(double, double);
std::pair< double, double > x_range();
const std::string x_separator();
svg_boxplot & x_size(unsigned int);
unsigned int x_size();
svg_boxplot & x_steps(int);
int x_steps();
const std::string x_suffix();
svg_boxplot & x_tick_color(const svg_color &);
svg_color x_tick_color();
svg_boxplot & x_tick_length(unsigned int);
double x_tick_length();
svg_boxplot & x_tick_width(unsigned int);
svg_boxplot & x_ticks_down_on(bool);
bool x_ticks_down_on();
svg_boxplot & x_ticks_on_window_or_axis(int);
int x_ticks_on_window_or_axis();
svg_boxplot & x_ticks_up_on(bool);
bool x_ticks_up_on();
svg_boxplot & x_ticks_values_color(const svg_color &);
svg_color x_ticks_values_color();
svg_boxplot & x_ticks_values_font_family(const std::string &);
const std::string & x_ticks_values_font_family();
svg_boxplot & x_ticks_values_font_size(unsigned int);
unsigned int x_ticks_values_font_size();
svg_boxplot & x_ticks_values_ioflags(std::ios_base::fmtflags);
std::ios_base::fmtflags x_ticks_values_ioflags();
svg_boxplot & x_ticks_values_precision(int);
int x_ticks_values_precision();
svg_boxplot & x_tight(double);
double x_tight();
svg_boxplot & x_value_font_size(unsigned int);
unsigned int x_value_font_size();
svg_boxplot & x_value_ioflags(std::ios_base::fmtflags);
std::ios_base::fmtflags x_value_ioflags();
svg_boxplot & x_value_precision(int);
int x_value_precision();
svg_boxplot & x_values_color(const svg_color &);
svg_color x_values_color();
svg_boxplot & x_values_font_family(const std::string &);
const std::string & x_values_font_family();
svg_boxplot & x_values_font_size(unsigned int);
unsigned int x_values_font_size();
svg_boxplot & x_values_ioflags(std::ios_base::fmtflags);
std::ios_base::fmtflags x_values_ioflags();
svg_boxplot & x_values_on(bool);
bool x_values_on();
svg_boxplot & x_values_precision(int);
int x_values_precision();
svg_boxplot & x_values_rotation(rotate_style);
int x_values_rotation();
svg_boxplot & x_with_zero(bool);
bool x_with_zero();
bool y_autoscale();
svg_boxplot & y_autoscale(bool);
svg_boxplot & y_autoscale(double, double);
svg_boxplot & y_autoscale(std::pair< double, double >);
template<typename T> svg_boxplot & y_autoscale(const T &, const T &);
template<typename T> svg_boxplot & y_autoscale(const T &);
svg_boxplot & y_axis_color(const svg_color &);
svg_color y_axis_color();

```

```

svg_boxplot & y_axis_on(bool);
bool y_axis_on();
svg_boxplot & y_axis_position(int);
double y_axis_position();
std::string y_label();
svg_boxplot & y_label(const std::string &);
svg_color y_label_color();
svg_boxplot & y_label_color(const svg_color &);
svg_boxplot & y_label_font_size(unsigned int);
svg_boxplot & y_label_on(bool);
std::string y_label_text();
svg_boxplot & y_label_units(const std::string &);
std::string y_label_units();
bool y_labels_strip_e0s();
svg_boxplot & y_major_interval(double);
svg_boxplot & y_major_labels_on(int);
svg_boxplot & y_major_tick_color(const svg_color &);
svg_boxplot & y_major_tick_length(unsigned int);
svg_boxplot & y_major_tick_width(unsigned int);
double y_minor_interval();
svg_boxplot & y_minor_tick_color(const svg_color &);
svg_boxplot & y_minor_tick_length(unsigned int);
svg_boxplot & y_minor_tick_width(unsigned int);
svg_boxplot & y_num_minor_ticks(unsigned int);
svg_boxplot & y_range(double, double);
std::pair< double, double > y_range();
svg_boxplot & y_size(unsigned int);
unsigned int y_size();
};

```

Description

(axis_plot_frame.hpp contains functions common to 1 and 2-D, and boxplot).

svg_boxplot public construct/copy/destruct

1. `svg_boxplot();`

Default constructor providing all the default colors, style etc,

svg_boxplot public member functions

1. `bool autoscale();`

Returns: true if to use autoscale values autoscaling for X-axis.

2. `svg_boxplot & autoscale(bool b);`

Set true if to use autoscale values for X-axis.

3. `svg_boxplot & autoscale_check_limits(bool b);`

Set true to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed if user can be sure all values are 'inside limits'.

4. `bool autoscale_check_limits();`

Returns: true if to check that values used for autoscaling are within limits.

5. `svg_boxplot & autoscale_plusminus(double);`

Set how many std_dev or standard deviations to allow for ellipses when autoscaling.

6. `double autoscale_plusminus();`

Returns: How many std_dev or standard deviations allowed for ellipses when autoscaling.

7. `svg_boxplot & axes_on(bool is);`

Set true if to draw **both** x and y axes (note plural axes).

8. `bool axes_on();`

Returns: true if to draw **both** x and y axis on.

9. `svg_boxplot & axis_color(const svg_color & color);`

Set color of vertical whisker axis line in box.

Returns: Reference to `svg_boxplot` to make chainable.

10. `svg_color axis_color();`

Returns: Color of vertical whisker axis line in box.

11. `svg_boxplot & axis_width(double l);`

Set width of the box, not the border.

Returns: Reference to `svg_boxplot` to make chainable.

12. `double axis_width();`

Returns: Width of the box, not the border.

13. `svg_boxplot & background_border_color(const svg_color & col);`

Set SVG image background border color.

Returns: Reference to `svg_boxplot` to make chainable.

14. `svg_color background_border_color();`

Returns: Color of the border of the background for the SVG image.

15. `svg_boxplot & background_border_width(double w);`

Set plot background border width.

16. `double background_border_width();`

Returns: Plot background border width.

17. `svg_boxplot & background_color(const svg_color & col);`

Set SVG image background color.

18. `svg_color background_color();`

Returns: Color of the background for the SVG image.

19. `svg_boxplot & boost_license_on(bool l);`

Set true if the Boost license conditions should be included in the SVG document.

20. `bool boost_license_on();`

Returns: true if the Boost license conditions should be included in the SVG document.

21. `svg_boxplot & box_border(const svg_color & color);`

Set color of box border.

Returns: Reference to `svg_boxplot` to make chainable.

22. `svg_color box_border();`

Returns: Color of box border.

23. `svg_boxplot & box_fill(const svg_color & color);`

Set color of box fill (not border).

Returns: Reference to `svg_boxplot` to make chainable.

24. `svg_color box_fill();`

Returns: Color of box fill (not border).

25. `svg_boxplot & box_width(double width);`

Set width of the box. (Width of the box, not the border).

Returns: Reference to `svg_boxplot` to make chainable.

26. `double box_width();`

Returns: width of the box. (Width of the box, not the border).

27. `svg_boxplot & confidence(double);`

Set confidence alpha for display of confidence intervals (default 0.05 for 95%).

28. `double confidence();`

Returns: Confidence alpha for display of confidence intervals (default 0.05 for 95%).

29. `svg_boxplot & coord_precision(int digits);`

Precision of SVG coordinates in decimal digits (default 3). 3 decimal digits precision is sufficient for small images. 4 or 5 decimal digits precision will give higher quality plots, especially for larger images, at the expense of larger .svg files, particularly if there are very many data points.

30. `int coord_precision();`

Returns: precision of SVG coordinates in decimal digits.

31. `svg_boxplot & copyright_date(const std::string d);`

Writes copyright date to the SVG document. and as metadata:<meta name="date" content="2007" />

32. `const std::string copyright_date();`

Returns: SVG document copyright_date.

33. `svg_boxplot & copyright_holder(const std::string d);`

Writes copyright_holder metadata to the SVG document (for header as) /and as metadata:<meta name="copyright" content="Paul A. Bristow" />

34. `const std::string copyright_holder();`

Returns: SVG document copyright holder.

35. `svg_boxplot & data_lines_width(double width);`

Set the width of lines joining data points.

36. `double data_lines_width();`

Returns: the width of lines joining data points.

37. `svg_boxplot & derived();`

Uses Curiously Recurring Template Pattern to allow 1D and 2D to reuse common code. See http://en.wikipedia.org/wiki/Curiously_Recurring_Template_Pattern.

38. `const svg_boxplot & derived() const;`

const version of derived()

39. `svg_boxplot & description(const std::string d);`

Writes description to the document for header as.

`<desc> My Description </desc>.`

40. `const std::string & description();`

Returns: Description of the document for header as`<desc> My description </desc>.`

41. `svg_boxplot & document_title(const std::string d);`

Set document title to the document for header as.

`<title> My Title </title>.`

42. `std::string document_title();`

Returns: Document title to the document for header as`<title> My Title </title>.`

43. `svg_boxplot & draw_line(double x1, double y1, double x2, double y2, const svg_color & col = black);`

Annotate plot with a line from SVG Coordinates X1, Y1 to X2, Y2. (Default color black). Note NOT the data values. See draw_plot_line if want to use user coordinates.

44. `svg_boxplot & draw_note(double x, double y, std::string note, rotate_style rot = horizontal, align_style al = center_align, const svg_color & col = black, text_style & tsty = no_style);`

Annotate plot with a text string (perhaps including Unicode), putting note at SVG Coordinates X, Y.

Defaults color black, rotation horizontal and align = center_align Using center_align is recommended as it will ensure that will center correctly (even if original string is made much longer because it contains Unicode, for example Greek or math symbols, taking about 6 characters per symbol) because the render engine does the centering.

45. `svg_boxplot & draw_plot_curve(double x1, double y1, double x2, double y2, double x3, double y3, const svg_color & col = black);`

Annotate plot with a line from user's Cartesian Coordinates X1, Y1 via X2, Y2 to X3, Y3.

For example, -10, -10, +10, +10, Default color black.

46. `svg_boxplot & draw_plot_line(double x1, double y1, double x2, double y2, const svg_color & col = black);`

Annotate plot with a line from user's Cartesian Coordinates X1, Y1 to X2, Y2.

For example, -10, -10, +10, +10, Default color black.

47. `svg_boxplot & extreme_outlier_color(const svg_color & color);`

Color of extreme outlier.

Returns: Reference to `svg_boxplot` to make chainable.

48. `svg_color extreme_outlier_color();`

Returns: Color of extreme outlier.

49. `svg_boxplot & extreme_outlier_fill(const svg_color & color);`

Set color of extreme outlier fill.

Returns: Reference to `svg_boxplot` to make chainable.

50. `svg_color extreme_outlier_fill();`

Returns: Color of extreme outlier fill.

51. `svg_boxplot & extreme_outlier_shape(point_shape shape);`

Set shape of extreme outlier marker.

Returns: Reference to `svg_boxplot` to make chainable.

52. `point_shape extreme_outlier_shape();`

Returns: Shape of extreme outlier marker.

53. `svg_boxplot & extreme_outlier_size(int size);`

Set size of extreme outlier marker.

Returns: Reference to `svg_boxplot` to make chainable.

54. `int extreme_outlier_size();`

Returns: Size of extreme outlier marker.

55. `svg_boxplot & extreme_outlier_values_on(bool cmd);`

Set true to show extreme outlier values.

Returns: Reference to `svg_boxplot` to make chainable.

56. `bool extreme_outlier_values_on();`

Returns: true if to show extreme outlier value(s).

57. `svg_boxplot & image_border_margin(double w);`

Set the margin around the plot window border (svg units, default pixels).

This prevents the plot window getting too close to other elements of the plot.

58. `double image_border_margin();`

Returns: the margin around the plot window border (svg units, default pixels).

59. `svg_boxplot & image_border_width(double w);`

Set the svg image border width (svg units, default pixels).

60. `double image_border_width();`

Returns: the svg image border width (svg units, default pixels).

61. `unsigned int image_x_size();`

Obselete - deprecated use x_size()

62. `svg_boxplot & image_x_size(unsigned int i);`

Obselete - deprecated - use x_size().

63. `unsigned int image_y_size();`

Obselete - deprecated - use y_size()

64. `svg_boxplot & image_y_size(unsigned int i);`

Obselete - deprecated - use y_size()

65. `svg_boxplot & legend_background_color(const svg_color & col);`

Set the background fill color of the legend box.

66. `svg_color legend_background_color();`

Returns: the background fill color of the legend box.

67. `svg_boxplot & legend_border_color(const svg_color & col);`

Set the border stroke color of the legend box.

68. `svg_color legend_border_color();`

Returns: the border stroke color of the legend box.

69. `const std::pair< double, double > legend_bottom_right();`

Returns: SVG coordinate (default pixels) of bottom right of legend box.

70. `bool legend_box_fill_on();`

Returns: true if legend box has a background fill color.

71. `svg_boxplot & legend_color(const svg_color & col);`

Set the color of the title of the legend.

72. `svg_color legend_color();`

Returns: the color of the title of the legend.

73. `svg_boxplot & legend_font_family(const std::string & family);`

Set the font family for the legend title.

74. `const std::string & legend_font_family();`

Returns: the font family for the legend title.

75. `svg_boxplot & legend_font_weight(const std::string & weight);`

Set the font weight for the legend title.

76. `const std::string & legend_font_weight();`

Returns: Font weight for the legend title.

77. `svg_boxplot & legend_header_font_size(int size);`

Set legend header font size (svg units, default pixels).

78. `int legend_header_font_size();`

Returns: legend header font size (svg units, default pixels).

79. `svg_boxplot & legend_lines(bool is);`

Set true if legend should include samples of the lines joining data points. This allows different series of data points to be distinguished by different color and/or width. This is especially useful to show plots of different functions and/or different parameters in different colors.

80. `bool legend_lines();`

Returns: true if legend should include samples of the lines joining data points.

81. `svg_boxplot & legend_on(bool cmd);`

Set true if a legend is wanted.

82 `bool legend_on();`

Returns: true if a legend is wanted.

83. `bool legend_outside();`

Returns: if the legend should be outside the plot area.

84. `svg_boxplot & legend_place(legend_places l);`

Set the position of the legend..

See Also:

`boost::svg::legend_places`

85. `legend_places legend_place();`

See Also:

`boost::svg::legend_places`

Returns: the position of the legend,

86. `svg_boxplot & legend_title(const std::string title);`

Set the title for the legend.

87. `const std::string legend_title();`

Returns: Title for the legend.

88. `svg_boxplot & legend_title_font_size(unsigned int size);`

Returns: Font family for the legend title.

89. `unsigned int legend_title_font_size();`

Returns: Font size for the legend title (svg units, default pixels).

90. `svg_boxplot & legend_top_left(double x, double y);`

Set position of top left of legend box (svg coordinates, default pixels). (Bottom right is controlled by contents, so the user cannot set it).

91. `const std::pair<double, double> legend_top_left();`

Returns: SVG coordinate (default pixels) of top left of legend box.

92. `svg_boxplot & legend_width(double width);`

Set the width for the legend box.

93. `double legend_width();`

Returns: Width for the legend box.

94. `svg_boxplot & license(std::string repro = "permits", std::string distrib = "permits", std::string attrib = "requires", std::string commercial = "permits", std::string derivative = "permits");`

Set license conditions for reproduction, attribution, commercial use, and derivative works, usually "permits", "requires", or "prohibits", and set license_on == true.

95. `const std::string license_attribution();`

Returns: SVG document attribution license conditions, usually "permits", "requires", or "prohibits".

96. `const std::string license_commercialuse();`

Returns: SVG document commercial use license conditions, usually "permits", "requires", or "prohibits".

97. `const std::string license_distribution();`

Returns: SVG document distribution license conditions, usually "permits", "requires", or "prohibits".

98. `svg_boxplot & license_on(bool l);`

Set if license conditions should be included in the SVG document.

99. `bool license_on();`

Returns: true if license conditions should be included in the SVG document.

100. `const std::string license_reproduction();`

Returns: SVG document reproduction license conditions, usually "permits", "requires", or "prohibits".

101. `svg_boxplot & limit_color(const svg_color &);`

Set the color for 'at limit' point stroke color.

102. `svg_color limit_color();`

Returns: the color for the 'at limit' point stroke color.

103. `svg_boxplot & limit_fill_color(const svg_color &);`

Set the color for 'at limit' point fill color.

104. `svg_color limit_fill_color();`

Returns: the color for the 'at limit' point fill color.

15 `svg_boxplot & median_color(const svg_color & color);`

Set color of median line in box.

Returns: Reference to `svg_boxplot` to make chainable.

16 `svg_color median_color();`

Returns: color of median line in box.

17 `svg_boxplot & median_values_on(bool cmd);`

Set true to show median value(s).

Returns: Reference to `svg_boxplot` to make chainable.

18 `bool median_values_on();`

Returns: true if to show median value(s).

19 `svg_boxplot & median_width(double l);`

Set width of the box (not the border).

Returns: Reference to `svg_boxplot` to make chainable.

10 `double median_width();`

Returns: width of the box (not the border).

11 `svg_boxplot & one_sd_color(const svg_color &);`

Set the color for the one standard deviation (~67% confidence) ellipse fill.

12 `svg_color one_sd_color();`

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

13 `svg_boxplot & outlier_color(const svg_color & color);`

Set color of outlier.

Returns: Reference to `svg_boxplot` to make chainable.

14 `svg_color outlier_color();`

Returns: Color of mild outlier.

15 `svg_boxplot & outlier_fill(const svg_color & color);`

Set color of mild outlier fill.

Returns: Reference to `svg_boxplot` to make chainable.

16 `svg_color outlier_fill();`

Returns: Color of outlier fill.

17 `svg_boxplot & outlier_shape(point_shape shape);`

Set shape of outlier marker.

Returns: Reference to `svg_boxplot` to make chainable.

18 `point_shape outlier_shape();`

Returns: Outlier marker shape.

19 `svg_boxplot & outlier_size(int size);`

Set size of outlier marker.

Returns: Reference to `svg_boxplot` to make chainable.

20 `int outlier_size();`

Returns: Size of outlier marker.

21 `svg_boxplot & outlier_style(plot_point_style & os);`

Set outlier style.

Returns: Reference to `svg_boxplot` to make chainable.

22 `plot_point_style & outlier_style();`

<
Returns: Outlier_style.

23 `svg_boxplot & outlier_values_on(bool cmd);`

Set true to show mild outlier values.

Returns: Reference to `svg_boxplot` to make chainable.

24 `bool outlier_values_on();`

Returns: true if to show mild outlier value(s).

25 `template<typename T>`
`svg_boxplot_series &`
`plot(const T & container, const std::string & title, double bw,`
 `svg_style bs, svg_style ms, svg_style as, double wl, svg_style minws,`
 `svg_style maxws, plot_point_style os, plot_point_style extos,`
 `int q_def, value_style vs, text_style ss);`

Add a data series (the whole container) to boxplot.

Example:

```
myboxplot.plot(myvalues);
```

Parameters:	as	Axis style.
	bs	Box_style
	bw	Box width.
	container	STL Container holding data series values to boxplot (must be convertible to double).
	extos	Extreme outlier style.
	maxws	Max whisker style.
	minws	Min whisker style.
	ms	Median marker style.
	os	Mild outlier style.
	q_def	Quartile definition H&F #.
	ss	Series style - for series title.
	title	Title of data series.
	vs	Style for data values.
	wl	Whisker length.
Template Parameters:	T	STL Container type holding data series.

16 `template<typename T>
svg_boxplot_series &
plot(const T & container, const std::string & title = "");`

Add a container of data series to boxplot.

Plot version copying box'n'whiskers parameters from parent boxplot.

Parameters:	container	A container of data series to boxplot.
	title	Title for boxplot. (Warning given if the default "" is used.)
Template Parameters:	T	data series value type (must be convertible to double).
Returns:		Reference to data series just added.

17 `svg_boxplot & plot_background_color(const svg_color & col);`

Set plot window background color.

Returns: Reference to `svg_boxplot` to make chainable.

18 `svg_color plot_background_color();`

Returns: Color of the background for the plot.

19 `svg_boxplot & plot_border_color(const svg_color & col);`

Set plot window border color.

Returns: Reference to `svg_boxplot` to make chainable.

20 `svg_color plot_border_color();`

Returns: Color of the border of the background for the plot.

13 `svg_boxplot & plot_border_width(double w);`

Set the margin around the plot window border (svg units, default pixels).

This prevents the plot window getting too close to other elements of the plot.

14 `double plot_border_width();`

Returns: the width for the plot window border (svg units, default pixels).

15 `svg_boxplot & plot_window_on(bool cmd);`

Set true if a plot window is wanted (or false if the whole image is to be used).

16 `bool plot_window_on();`

Returns: true if a plot window is wanted (or false if the whole image is to be used).

17 `svg_boxplot & plot_window_x(double min_x, double max_x);`

Set the minimum and maximum (cartesian data units) for the plot window X axis. This is normally calculated from other plot values.

18 `std::pair< double, double > plot_window_x();`

Returns: both the left and right (X axis) of the plot window.

19 `double plot_window_x_left();`

Returns: left of the plot window.

20 `double plot_window_x_right();`

Returns: right of the plot window.

21 `svg_boxplot & plot_window_y(double min_y, double max_y);`

Set the minimum and maximum (cartesian data units) for the plot window Y axis. This is normally calculated from other plot values.

22 `std::pair< double, double > plot_window_y();`

Returns: both the top and bottom (Y axis) of the plot window.

23 `double plot_window_y_bottom();`

Returns: top of the plot window.

24 `double plot_window_y_top();`

Returns: top of the plot window.

13 `svg_boxplot & quartile_definition(int def);`

Set definition of quartile.

Returns: Reference to `svg_boxplot` to make chainable.

14 `int quartile_definition();`

Returns: Definition # of quartile.

15 `std::pair< double, double > size();`

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

16 `svg_boxplot & size(unsigned int x, unsigned int y);`

Set SVG image width (x) and height (y).

Returns: Reference to `svg_boxplot` to make chainable.

17 `svg_boxplot & three_sd_color(const svg_color &);`

Set the color for three standard deviation (~99% confidence) ellipse fill.

18 `svg_color three_sd_color();`

Returns: Color for three standard deviation (~99% confidence) ellipse fill.

19 `svg_boxplot & title(const std::string title);`

Set a title for plot. The string may include Unicode for greek letter and symbols. **example:** A title that includes a greek omega and degree symbols:

```
my_plot.title("Plot of &#xA9; function (&#x00B0;C)");
```

Unicode symbols are at <http://unicode.org/charts/symbols.html>.

20 `svg_boxplot & title(const std::string & str);`

Set title text for plot.

Returns: Reference to `svg_boxplot` to make chainable.

21 `std::string title();`

Returns: title of the plot.

22 `svg_boxplot & title_color(const svg_color & col);`

Set boxplot title color.

Returns: Reference to `svg_boxplot` to make chainable.

13 `svg_color title_color();`

Returns: color of the title.

14 `svg_boxplot & title_font_alignment(align_style alignment);`

Set the alignment for the title.

15 `align_style title_font_alignment();`

Returns: the alignment for the title.

16 `svg_boxplot & title_font_decoration(const std::string & decoration);`

Set the font decoration for the title (default normal, or underline, overline or strike-thru).

17 `const std::string & title_font_decoration();`

Returns: Font decoration for the title (default normal, or underline, overline or strike-thru).

18 `svg_boxplot & title_font_family(const std::string & family);`

Set the font family for the title (for example: .title_font_family("Lucida Sans Unicode");.

19 `const std::string & title_font_family();`

Returns: Font family for the title.

20 `svg_boxplot & title_font_rotation(rotate_style rotate);`

Set the rotation for the title font (degrees, 0 to 360 in steps using rotate_style, for example horizontal, uphill...

21 `int title_font_rotation();`

Returns: the rotation for the title font (degrees).

22 `svg_boxplot & title_font_size(unsigned int i);`

Sets the font size for the title (SVG units, default pixels).

23 `unsigned int title_font_size();`

Returns: Font size for the title (SVG units, default pixels).

24 `svg_boxplot & title_font_stretch(const std::string & stretch);`

Set the font stretch for the title (default normal), wider or narrow.

25 `const std::string & title_font_stretch();`

Returns: Font stretch for the title.

16 `svg_boxplot & title_font_style(const std::string & style);`

Set the font style for the title (default normal).

17 `const std::string & title_font_style();`

Returns: Font style for the title (default normal).

18 `svg_boxplot & title_font_weight(const std::string & weight);`

Set the font weight for the title (default normal).

19 `const std::string & title_font_weight();`

Returns: Font weight for the title.

20 `bool title_on();`

Returns: true if will show a title for the plot.

21 `svg_boxplot & title_on(bool cmd);`

Write SVG boxplot to ostream.

Set true to show whole boxplot title.

Returns: Reference to `svg_boxplot` to make chainable.

22 `svg_boxplot & title_size(unsigned int size);`

Set font size for title text.

Returns: Reference to `svg_boxplot` to make chainable.

23 `svg_boxplot & two_sd_color(const svg_color &);`

Set the color for two standard deviation (~95% confidence) ellipse fill.

24 `svg_color two_sd_color();`

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

25 `svg_boxplot & whisker_length(double width);`

Set the length of the whisker.

Returns: Reference to `svg_boxplot` to make chainable.

26 `double whisker_length();`

Get whisker length (width of the box, not the margin).

<
Returns: Length of whisker.

17 `svg_boxplot & write(const std::string & file);`

Write SVG image to file.

18 `svg_boxplot & write(std::ostream & s_out);`

Write SVG boxplot to file.

Write SVG image to ostream.

Returns: Reference to `svg_boxplot` to make chainable.

19 `svg_boxplot & x_addlimits_color(const svg_color & col);`

Set the color of X confidence limits of value, for example, the color in "<1.23, 1.45>".

20 `svg_color x_addlimits_color();`

Returns: the color of X confidence limits of value, for example, the color of "<1.23, 1.45>".

21 `svg_boxplot & x_addlimits_on(bool b);`

Set if to append confidence limits to data point X values near data points markers.

22 `bool x_addlimits_on();`

Returns: true if to append confidence limits estimate to data point X values near data points markers.

23 `double x_auto_max_value();`

Returns: X-axis maximum value computed by autoscale.

24 `double x_auto_min_value();`

Returns: X-axis minimum value computed by autoscale.

25 `double x_auto_tick_interval();`

Returns: the X-axis major tick interval computed by autoscale.

26 `int x_auto_ticks();`

Returns: the X-axis number of major ticks computed by autoscale.

27 `bool x_autoscale();`

Returns: true if to use autoscale value for X-axis.

18 `svg_boxplot & x_autoscale(bool b);`

Set true if to use autoscale values for X-axis.

19 `svg_boxplot & x_autoscale(std::pair< double, double > p);`

autoscale X axis using a pair of doubles.

20 `svg_boxplot & x_autoscale(const T & container);`

<

21 `svg_boxplot & x_autoscale(const T & begin, const T & end);`

<

22 `svg_boxplot & x_axis_color(const svg_color & col);`

Set the color of the X-axis line.

23 `svg_color x_axis_color();`

Returns: the color of the X-axis line.

24 `svg_boxplot & x_axis_label_color(const svg_color & col);`

Set X axis label color, for example, red.

25 `svg_color x_axis_label_color();`

Returns: X axis label color. <X-axis ticks values label style.

26 `svg_boxplot & x_axis_on(bool is);`

If set true, draw a horizontal X-axis line.

27 `bool x_axis_on();`

Returns: true if will draw a horizontal X-axis line.

28 `svg_boxplot & x_axis_position(int pos);`

Set position of the horizontal X-axis line (on the border).

But controlled by the intersection with Y-axis, so this only changes the default position from bottom to top, but will be changed if X-axis intersects the Y-axis (that is, if Y-axis includes zero).

Returns: Reference to `svg_boxplot` to make chainable.

29 `double x_axis_position();`

Returns: position of the horizontal X-axis line (on the border).

20 `svg_boxplot & x_axis_vertical(double fraction);`

Set vertical position of X-axis for 1D as fraction of plot window.

21 `bool x_axis_vertical();`

Returns: vertical position of X-axis for 1D as fraction of plot window.

22 `svg_boxplot & x_axis_width(double width);`

Set the width of X-axis lines.

23 `double x_axis_width();`

Returns: the width of X-axis lines.

24 `svg_boxplot & x_datetime_color(const svg_color & col);`

Set the color of X date time , for example, the color of text in "".

25 `svg_color x_datetime_color();`

Returns: the color of X date time, for example, the color of text in "".

26 `svg_boxplot & x_datetime_on(bool b);`

Set true if to append date time to data point X values near data points markers.

27 `bool x_datetime_on();`

Returns: true if to append an date time to data point X values near data points markers.

28 `svg_boxplot & x_decor(const std::string & pre, const std::string & sep = "", const std::string & suf = "");`

Set prefix, separator and suffix together for x_ values. Note if you want a space, you must use a Unicode space " ", for example, ", " rather than ASCII space", ". If 1st char in separator == , then Y values and info will be on a newline below.

29 `svg_boxplot & x_df_color(const svg_color & col);`

Set the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

30 `svg_color x_df_color();`

Returns: the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

31 `svg_boxplot & x_df_on(bool b);`

Set true if to append a degrees of freedom estimate to data point X values near data points markers.

22 `bool x_df_on();`

Returns: true if to append a degrees of freedom estimate to data point X values near data points markers.

23 `svg_boxplot & x_id_color(const svg_color & col);`

Set the color of X id or name, for example, the color of text in "my_id".

24 `svg_color x_id_color();`

Returns: the color of X X id or name, for example, the color of text in "my_id".

25 `svg_boxplot & x_id_on(bool b);`

Set true if to append append an ID or name to data point X values near data points markers.

26 `bool x_id_on();`

Returns: true if to append an ID or name to data point X values near data points markers.

27 `std::string x_label();`

Returns: the text to label the X-axis.

28 `svg_boxplot & x_label(const std::string & str);`

Set label for X axis (can also append optional units).

See Also:

`x_label_units_on` and

`x_label_units`

Returns: Reference to `svg_boxplot` to make chainable.

29 `svg_boxplot & x_label_color(const svg_color & col);`

Set the font color for the X axis label.

Returns: Reference to `svg_boxplot` to make chainable.

30 `svg_color x_label_color();`

Returns: Color of the X axis label.

31 `svg_boxplot & x_label_font_family(const std::string & family);`

Set X tick value label font family. Available fonts depend on the program rendering the SVG XML, usually a browser. The default font (usually "default_font") is used if a render program does not provide the font specified. These are probably usable:

```
"arial", "impact", "courier", "lucida console", "Lucida sans unicode", "verdana", "calibri", "century",
"lucida calligraphy", "tahoma", "vivaldi", "informal roman", "lucida handwriting", "lucida bright", "helvetica"
```

22 `const std::string & x_label_font_family();`

Returns: X tick value label font family.

23 `svg_boxplot & x_label_font_size(unsigned int i);`

Set X axis label font size (svg units, default pixels).

24 `unsigned int x_label_font_size();`

Returns: X axis label font size (svg units, default pixels).

25 `svg_boxplot & x_label_on(bool cmd);`

true if to include title in plot.

Set true if X axis name or label, for example: "length of thing".

Returns: Reference to `svg_boxplot` to make chainable.

26 `bool x_label_on();`

Returns: label for the X axis.

27 `svg_boxplot & x_label_size(unsigned int size);`

Set the font size for the X axis label.

Returns: Reference to `svg_boxplot` to make chainable.

28 `std::string x_label_text();`

Returns: Text of label for X axis.

29 `svg_boxplot & x_label_units(const std::string & str);`

Set the text to add units to the X-axis label.

30 `std::string x_label_units();`

Returns: the text to add units to the X-axis label. <The label will only be shown if `x_label_on()` == true.

31 `svg_boxplot & x_label_units_on(bool cmd);`

Set true if want X axis label to include units (as well as label like "length"). <.

See Also:

x_label_units which also sets true.

22 `bool x_label_units_on();`

Set true if want X axis label to include units (as well as label like "length").

23 `svg_boxplot & x_label_width(double width);`

Set the width (boldness) of X-axis label (including any units). (not recommended until browsers implement better).

24 `double x_label_width();`

Returns: the width (boldness) of X-axis label (including any units).

25 `svg_boxplot & x_labels_strip_e0s(bool cmd);`

Set if to strip redundant zeros, signs and exponents, for example, reducing "1.2e+000" to "1.2" This markedly reduces visual clutter, and is the default.

26 `svg_boxplot & x_major_grid_color(const svg_color & col);`

Set the color of X-axis major grid lines.

27 `svg_color x_major_grid_color();`

Set the color of X-axis major grid lines.

28 `svg_boxplot & x_major_grid_on(bool is);`

If set true, will include a major X-axis grid.

29 `bool x_major_grid_on();`

Returns: true if will include a major X-axis grid.

30 `svg_boxplot & x_major_grid_width(double w);`

Set the width of X-axis major grid lines.

31 `double x_major_grid_width();`

Returns: the color of X-axis major grid lines.

32 `svg_boxplot & x_major_interval(double inter);`

Set the interval between X-axis major ticks.

33 `double x_major_interval();`

Returns: the interval between X-axis major ticks.

24 `svg_boxplot & x_major_label_rotation(rotate_style rot);`

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

`rotate_style`

25 `rotate_style x_major_label_rotation();`

See Also:

`rotate_style`

Returns: rotation for X ticks major value labels.

26 `int x_major_labels();`

Returns: which side of the X axis for labels.

27 `svg_boxplot & x_major_labels_on(int cmd);`

Set direction of X major labels.

0 means to down (default), 0 (false) means none, > 0 means to top.

Returns: Reference to `svg_boxplot` to make chainable.

28 `svg_boxplot & x_major_labels_side(int side);`

Position of labels for X major ticks on horizontal X axis line.

Parameters: `side` > 0 X tick value labels to left of Y axis line (default), 0 (false) no major X tick value labels on Y axis, 0 X tick labels to right of Y axis line.

29 `int x_major_labels_side();`

Returns: the side for X ticks major value labels.

30 `svg_boxplot & x_major_tick(double d);`

Set interval (Cartesian units) between major ticks.

31 `double x_major_tick();`

Returns: interval (Cartesian units) between major ticks.

32 `svg_boxplot & x_major_tick_color(const svg_color & col);`

Set the color of X-axis major ticks.

33 `svg_color x_major_tick_color();`

Returns: the color of X-axis major ticks.

24 `svg_boxplot & x_major_tick_length(double length);`

Set length of X major ticks (SVG units, default pixels).

25 `double x_major_tick_length();`

Set length of X major ticks (SVG units, default pixels).

26 `svg_boxplot & x_major_tick_width(double width);`

Set width of X major ticks (SVG units, default pixels).

27 `double x_major_tick_width();`

Returns: Width of major ticks on the X axis.

28 `svg_boxplot & x_max(double x);`

Set the maximum value on the X-axis.

29 `double x_max();`

autoscale set & get parameters, <Note: all these *MUST* precede x_autoscale(data) call.

Returns: the maximum value on the X-axis.

30 `svg_boxplot & x_min(double min_x);`

Set the minimum value on the X-axis.

31 `double x_min();`

Returns: the minimum value on the X-axis.

32 `svg_boxplot & x_min_ticks(int min_ticks);`

Set X-axis autoscale to include at least minimum number of ticks (default = 6).

33 `int x_min_ticks();`

Returns: X-axis autoscale minimum number of ticks.

34 `svg_boxplot & x_minor_grid_color(const svg_color & col);`

Set the color of X-axis minor grid lines.

35 `svg_color x_minor_grid_color();`

Returns: the color of X-axis minor grid lines.

26 `svg_boxplot & x_minor_grid_on(bool is);`

If set true, will include a minor X-axis grid.

27 `bool x_minor_grid_on();`

Returns: true if will include a major X-axis grid.

28 `svg_boxplot & x_minor_grid_width(double w);`

Set the width of X-axis minor grid lines.

29 `double x_minor_grid_width();`

Returns: the width of X-axis minor grid lines.

30 `double x_minor_interval();`

Returns: interval between X minor ticks.

31 `svg_boxplot & x_minor_interval(double interval);`

Set interval between X-axis minor ticks.

32 `svg_boxplot & x_minor_tick_color(const svg_color & col);`

Set the color of X-axis minor ticks.

33 `svg_color x_minor_tick_color();`

Returns: the color of X-axis minor ticks.

34 `svg_boxplot & x_minor_tick_length(double length);`

Set length of X minor ticks (SVG units, default pixels).

35 `double x_minor_tick_length();`

Returns: length of X minor ticks (SVG units, default pixels).

36 `svg_boxplot & x_minor_tick_width(double width);`

Set width of X minor ticks (SVG units, default pixels).

37 `double x_minor_tick_width();`

Returns: width of X minor ticks (SVG units, default pixels).

38 `svg_boxplot & x_num_minor_ticks(unsigned int num);`

Set number of X-axis minor ticks between major ticks.

29 `unsigned int x_num_minor_ticks();`

Returns: number of X-axis minor ticks between major ticks.

30 `svg_boxplot & x_order_color(const svg_color & col);`

Set the color of X order #, for example, the color of #42.

31 `svg_color x_order_color();`

Returns: the color of X order #, for example, the color of #42.

32 `svg_boxplot & x_order_on(bool b);`

Set true if to append append an order # to data point X values near data points markers.

33 `bool x_order_on();`

Returns: true if to append an order # to data point X values near data points markers.

34 `svg_boxplot & x_plusminus_color(const svg_color & col);`

Set the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

35 `svg_color x_plusminus_color();`

Returns: the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

36 `svg_boxplot & x_plusminus_on(bool b);`

Set if to append std_dev estimate to data point X values near data points markers.

37 `bool x_plusminus_on();`

Returns: true if to append std_dev estimate to data point X values near data points markers.

38 `const std::string x_prefix();`

Returns: the prefix.

39 `svg_boxplot & x_range(double min_x, double max_x);`

Set the range of values on the X-axis. The minimum and maximum values must be finite and not too near to the minima or maxima that can be represented by floating point double, std::numeric_limits<double>::min() or std::numeric_limits<double>::max(), and the range must not be too small.

40 `std::pair< double, double > x_range();`

Returns: the range of values on the X-axis.

21 `const std::string x_separator();`

Returns: the separator, perhaps including Unicode.

22 `svg_boxplot & x_size(unsigned int i);`

Set SVG image X-axis size (SVG units, default pixels).

23 `unsigned int x_size();`

Returns: width of the SVG image.

24 `svg_boxplot & x_steps(int steps);`

Set autoscale to set ticks in steps multiples of:

2,4,6,8,10, if 2

or 1,5,10 if 5

or 2,5,10 if 10.

default = 0 (none).



Note

: Must precede `x_autoscale(data)` call).

25 `int x_steps();`

Returns: autoscale to set ticks in steps.

26 `const std::string x_suffix();`

Returns: the suffix (only used if separator != "")

27 `svg_boxplot & x_tick_color(const svg_color & col);`

Returns: Y major ticks color.

Reference to `svg_boxplot` to make chainable.

28 `svg_color x_tick_color();`

Returns: Color of ticks on the X axis.

29 `svg_boxplot & x_tick_length(unsigned int length);`

Set the length of major ticks on the X axis.

Returns: Reference to `svg_boxplot` to make chainable.

30 `double x_tick_length();`

Returns: Length of major ticks on the X axis.

31 `svg_boxplot & x_tick_width(unsigned int width);`

Set the width of major ticks on the X axis.

Returns: Reference to `svg_boxplot` to make chainable.

32 `svg_boxplot & x_ticks_down_on(bool cmd);`

Set true if Y major ticks should mark upwards.

33 `bool x_ticks_down_on();`

Returns: true if Y major ticks should mark upwards.

34 `svg_boxplot & x_ticks_on_window_or_axis(int side);`

Set position of X ticks on window or axis.

Parameters: side -1 X ticks on bottom of plot window, 0 X ticks on X-axis horizontal line, +1 X ticks top of plot window.

35 `int x_ticks_on_window_or_axis();`

Returns: true if X axis ticks wanted on the window (rather than on axis).
 <-1 bottom of plot window, 0 on horizontal X axis , +1 top of plot window.

36 `svg_boxplot & x_ticks_up_on(bool cmd);`

Set true if X major ticks should mark upwards.

37 `bool x_ticks_up_on();`

Returns: true if X major ticks should mark upwards.

38 `svg_boxplot & x_ticks_values_color(const svg_color & col);`

Set X axis tick value label color.

39 `svg_color x_ticks_values_color();`

Returns: X-axis ticks value label color.

40 `svg_boxplot & x_ticks_values_font_family(const std::string & family);`

Set X ticks value label font family. Available fonts depend on the program rendering the SVG XML, usually a browser. The default font (usually "default_font") is used if a render program does not provide the font specified. These are probably usable:

```
"arial", "impact", "courier", "lucida console", "Lucida sans unicode", "verdana", "calibri", "century",
"lucida calligraphy", "tahoma", "vivaldi", "informal roman", "lucida handwriting", "lucida bright", "helvetica"
```

31 `const std::string & x_ticks_values_font_family();`

Returns: X ticks value label font family.

32 `svg_boxplot & x_ticks_values_font_size(unsigned int i);`

Set X ticks value label font size (svg units, default pixels).

33 `unsigned int x_ticks_values_font_size();`

Set X ticks value label font size (svg units, default pixels).

34 `svg_boxplot & x_ticks_values_ioflags(std::ios_base::fmtflags f);`

Set iostream format flags of data point X values near data points markers.

35 `std::ios_base::fmtflags x_ticks_values_ioflags();`

Returns: iostream format flags of data point X values near data points markers.

36 `svg_boxplot & x_ticks_values_precision(int p);`

Set iostream decimal digits precision of data point X values near data points markers.

37 `int x_ticks_values_precision();`

Returns: iostream decimal digits precision of data point X values near data points markers.

38 `svg_boxplot & x_tight(double tight);`

Set tolerance to autoscale to permit data points slightly outside both end ticks.

39 `double x_tight();`

Returns: tolerance given to autoscale to permit data points slightly outside both end ticks. <Get results of autoscaling.

40 `svg_boxplot & x_value_font_size(unsigned int i);`

Set X tick value label font size (svg units, default pixels).

41 `unsigned int x_value_font_size();`

Returns: X tick value label font size (svg units, default pixels).

42 `svg_boxplot & x_value_ioflags(std::ios_base::fmtflags flags);`

Set iostream std::ios::fmtflags for X value label (default decimal == 0X201). Mainly useful for changing to scientific, fixed or hexadecimal format. For example:

`myplot.x_value_ioflags(std::ios::dec | std::ios::scientific)`

33 `std::ios_base::fmtflags x_value_ioflags();`

Returns: stream std::ios::fmtflags for control of format of X value labels.

34 `svg_boxplot & x_value_precision(int digits);`

Set precision of X-tick label values in decimal digits (default 3). 3 decimal digits precision is sufficient for small images. 4 or 5 decimal digits precision will give more cluttered plots. If the range of labels is very small, then more digits will be essential.

35 `int x_value_precision();`

Returns: Precision of X-tick label values in decimal digits

36 `svg_boxplot & x_values_color(const svg_color & col);`

Set the color of data point X values near data points markers.

37 `svg_color x_values_color();`

Returns: the color of data point X values near data points markers.

38 `svg_boxplot & x_values_font_family(const std::string & family);`

Set font family of data point X values near data points markers.

39 `const std::string & x_values_font_family();`

Returns: font family of data point X values near data points markers.

40 `svg_boxplot & x_values_font_size(unsigned int i);`

Set font size of data point X values near data points markers.

41 `unsigned int x_values_font_size();`

Returns: font size of data point X values near data points markers.

42 `svg_boxplot & x_values_ioflags(std::ios_base::fmtflags f);`

Set iostream format flags of data point X values near data points markers.

43 `std::ios_base::fmtflags x_values_ioflags();`

Returns: iostream format flags of data point X values near data points markers.

44 `svg_boxplot & x_values_on(bool b);`

Set true to show data point values near data points markers.

45 `bool x_values_on();`

Returns: true if to show data point values near data points markers.

36 `svg_boxplot & x_values_precision(int p);`

Set iostream decimal digits precision of data point X values near data points markers.

37 `int x_values_precision();`

Returns: iostream decimal digits precision of data point X values near data points markers.

38 `svg_boxplot & x_values_rotation(rotate_style rotate);`

Returns: the rotation (rotate_style) of data point X values near data points markers.

39 `int x_values_rotation();`

Set the rotation (rotate_style) of data point X values near data points markers.

40 `svg_boxplot & x_with_zero(bool b);`

Set X-axis autoscale to include zero (default = false).

41 `bool x_with_zero();`

Returns: true if X-axis autoscale to include zero (default = false).

42 `bool y_autoscale();`

Returns: true if Y-axis to use autoscaling.

43 `svg_boxplot & y_autoscale(bool b);`

Set true if Y axis is to use autoscale.

Returns: Reference to `svg_boxplot` to make chainable.

44 `svg_boxplot & y_autoscale(double min, double max);`

Autoscale Y-axis using minimum and maximum provided as two doubles.

Returns: Reference to `svg_boxplot` to make chainable.

45 `svg_boxplot & y_autoscale(std::pair< double, double > p);`

Set Y min & max values (as a std::pair) to use for autoscaling Y-axis.

Returns: Reference to `svg_boxplot` to make chainable.

46 `template<typename T> svg_boxplot & y_autoscale(const T & begin, const T & end);`

Autoscale using iterators into a container (allowing only a part of container values to be used to calculate minimum and maximum Y-axis).

Parameters: `begin` First element to use to calculate autoscaled values.
`end` Last element to use to calculate autoscaled values.
 Template Parameters: `T` an STL container: array, vector ...
 Returns: Reference to [svg_boxplot](#) to make chainable.

37 `template<typename T> svg_boxplot & y_autoscale(const T & container);`

Autoscale using a whole container to calculate minimum and maximum autoscaled Y-axis values.

Template Parameters: `T` an STL container: array, vector ...
 Returns: Reference to [svg_boxplot](#) to make chainable.

38 `svg_boxplot & y_axis_color(const svg_color & col);`

Set the color of the Y-axis line.

39 `svg_color y_axis_color();`

Returns: the color of the Y-axis line.

40 `svg_boxplot & y_axis_on(bool is);`

If set true, draw a vertical Y-axis line.

41 `bool y_axis_on();`

Returns: true if will draw a horizontal X-axis line.

42 `svg_boxplot & y_axis_position(int pos);`

Set position of the vertical Y-axis line (on the border). But controlled by the intersection with X-axis, so this only changes the default position from bottom to top, but will be changed if X-axis intersects the X-axis (that is if X-axis includes zero).

Returns: Reference to [svg_boxplot](#) to make chainable.

43 `double y_axis_position();`

Returns: Position of the vertical Y-axis line (on the border).

44 `std::string y_label();`

Returns: the text for the Y-axis label. < The label will only be shown if `y_label_on() == true`.

45 `svg_boxplot & y_label(const std::string & str);`

Set Y axis label.

Returns: Reference to [svg_boxplot](#) to make chainable.

46 `svg_color y_label_color();`

Returns: the color of Y-axis label (including any units).

37 `svg_boxplot & y_label_color(const svg_color & col);`

Set font color for Y axis label.

Returns: Reference to `svg_boxplot` to make chainable.

38 `svg_boxplot & y_label_font_size(unsigned int size);`

Set font size for Y axis label.

Returns: Reference to `svg_boxplot` to make chainable.

39 `svg_boxplot & y_label_on(bool cmd);`

Set true if Y axis name or label, for example: "width of thing".

Returns: Reference to `svg_boxplot` to make chainable.

40 `std::string y_label_text();`

Returns: Text of label for Y axis.

41 `svg_boxplot & y_label_units(const std::string & str);`

Set the text to add units to the Y-axis label.

42 `std::string y_label_units();`

Returns: the text to add units to the X-axis label.

43 `bool y_labels_strip_e0s();`

Returns: if to strip redundant zeros, signs and exponents, for example, reducing "1.2e+000" to "1.2"

44 `svg_boxplot & y_major_interval(double inter);`

Set the interval between major ticks on the Y axis.

45 `svg_boxplot & y_major_labels_on(int cmd);`

Set direction of Y major labels. < 0 means to left (default), 0 (false) means none, > 0 means to right.

Returns: Reference to `svg_boxplot` to make chainable.

46 `svg_boxplot & y_major_tick_color(const svg_color & col);`

Set Y major ticks color.

Returns: Reference to `svg_boxplot` to make chainable.

47 `svg_boxplot & y_major_tick_length(unsigned int length);`

Set the length of major ticks on the Y axis.

Returns: Reference to `svg_boxplot` to make chainable.

38 `svg_boxplot & y_major_tick_width(unsigned int width);`

Set the width of major ticks on the Y axis.

Returns: Reference to `svg_boxplot` to make chainable.

39 `double y_minor_interval();`

Returns: interval between Y minor ticks.

30 `svg_boxplot & y_minor_tick_color(const svg_color & col);`

Set Y minor ticks color.

Returns: Reference to `svg_boxplot` to make chainable.

31 `svg_boxplot & y_minor_tick_length(unsigned int length);`

Set the length of minor ticks on the Y axis.

Returns: Reference to `svg_boxplot` to make chainable.

32 `svg_boxplot & y_minor_tick_width(unsigned int width);`

Set the width of minor ticks on the Y axis.

Returns: Reference to `svg_boxplot` to make chainable.

33 `svg_boxplot & y_num_minor_ticks(unsigned int num);`

Set the number of minor ticks between major ticks on the Y axis. For example, 1 gives alternating major and minor ticks, 4 is more useful giving major ticks at 1, 5, 10, 15...

9 gives major ticks at 10, 20, 30...

Returns: Reference to `svg_boxplot` to make chainable.

34 `svg_boxplot & y_range(double min_y, double max_y);`

Set range of Y values for Y axis (and do not use autoscale).

Returns: Reference to `svg_boxplot` to make chainable.

35 `std::pair< double, double > y_range();`

Set `y_range` using a pair of doubles (and do not use autoscale).

36 `svg_boxplot & y_size(unsigned int i);`

Set SVG image Y-axis size (SVG units, default pixels).

37. `unsigned int y_size();`

Returns: height of the SVG image.

Class `svg_boxplot_series`

`boost::svg::svg_boxplot_series`

Synopsis

```
// In header: <boost/svg_plot/svg_boxplot.hpp>

class svg_boxplot_series {
public:
    // construct/copy/destruct
    template<typename T>
    svg_boxplot_series(T, T, const std::string &, double, svg_style,
                      svg_style, svg_style, double, svg_style, svg_style,
                      plot_point_style, plot_point_style, int, value_style,
                      text_style);

    // public member functions
    svg_boxplot_series & axis_color(const svg_color &);
    svg_color axis_color();
    svg_boxplot_series & axis_width(double);
    double axis_width();
    svg_boxplot_series & box_border(const svg_color &);
    svg_color box_border();
    svg_boxplot_series & box_fill(const svg_color &);
    svg_color box_fill();
    svg_style & box_style();
    svg_boxplot_series & box_style(svg_style &);
    svg_boxplot_series & box_width(double);
    double box_width();
    void calculate_quantiles();
    svg_boxplot_series & extreme_outlier_color(const svg_color &);
    svg_color extreme_outlier_color();
    svg_boxplot_series & extreme_outlier_fill(const svg_color &);
    svg_color extreme_outlier_fill();
    svg_boxplot_series & extreme_outlier_shape(point_shape);
    point_shape extreme_outlier_shape();
    svg_boxplot_series & extreme_outlier_size(int);
    int extreme_outlier_size();
    svg_boxplot_series & max_whisker_color(const svg_color &);
    svg_color max_whisker_color();
    svg_boxplot_series & max_whisker_width(double);
    double max_whisker_width();
    svg_boxplot_series & median_color(const svg_color &);
    svg_color median_color();
    svg_style & median_style();
    svg_boxplot_series & median_style(svg_style &);
    svg_boxplot_series & median_width(double);
    double median_width();
    svg_boxplot_series & min_whisker_color(const svg_color &);
    svg_color min_whisker_color();
    svg_boxplot_series & min_whisker_width(double);
    double min_whisker_width();
    svg_boxplot_series & outlier_color(const svg_color &);
    svg_color outlier_color();
```

```

svg_boxplot_series & outlier_fill(const svg_color &);
svg_color outlier_fill();
svg_boxplot_series & outlier_shape(point_shape);
point_shape outlier_shape();
svg_boxplot_series & outlier_size(int);
int outlier_size();
plot_point_style & outlier_style();
svg_boxplot_series & outlier_style(plot_point_style &);
svg_boxplot_series & quartile_definition(int);
int quartile_definition();
svg_boxplot_series & title(const std::string &);
const std::string title();
svg_boxplot_series & whisker_length(double);
double whisker_length();
};

```

Description

Information about a series of data values to be displayed as a Box Plot. A Box Plot that can contain several boxplot data series. Median, whiskers and outliers are computed for each series.

See Also:

<http://en.wikipedia.org/wiki/Boxplot>

svg_boxplot_series public construct/copy/destruct

1. `template<typename T>`
`svg_boxplot_series(T begin, T end, const std::string & title, double bw,`
`svg_style bs, svg_style ms, svg_style as, double wl,`
`svg_style minws, svg_style maxws, plot_point_style os,`
`plot_point_style extos, int q_def, value_style vs,`
`text_style ss);`

Constructor, providing default values for all data members.

.

Default Constructor sorts a copy of container, and uses the copy for fast lookup of quartile values.

Template Parameters: `T` An STL data container type, typically `double` or `uncertain`. All other parameters can also be added using chainable functions.

svg_boxplot_series public member functions

1. `svg_boxplot_series & axis_color(const svg_color & color);`

Set color of axis line.

Returns: Reference to `svg_boxplot_series` to make chainable.

2. `svg_color axis_color();`

Returns: Color of axis line.

3. `svg_boxplot_series & axis_width(double l);`

Set width of axis line.

Returns: Reference to `svg_boxplot_series` to make chainable.

4. `double axis_width();`

Returns: width of axis line.

5. `svg_boxplot_series & box_border(const svg_color & color);`

Set color of box border.

Returns: Reference to `svg_boxplot_series` to make chainable.

6. `svg_color box_border();`

Returns: color of box border.

7. `svg_boxplot_series & box_fill(const svg_color & color);`

Set color of box fill (not border).

Returns: Reference to `svg_boxplot_series` to make chainable.

8. `svg_color box_fill();`

Returns: color of box fill.

9. `svg_style & box_style();`

Returns: box style.

10. `svg_boxplot_series & box_style(svg_style & bs);`

Set entire box style.

Returns: Reference to `svg_boxplot_series` to make chainable.

11. `svg_boxplot_series & box_width(double l);`

Set width of the box.

Returns: Reference to `svg_boxplot_series` to make chainable.

12. `double box_width();`

Returns: width of the box.

13. `void calculate_quantiles();`

Divide sorted data set into four equal parts, called quartiles, so each part represent 1/4th of the sampled population.

Michael Frigge, David C. Hoaglin and Boris Iglewicz
The American Statistician, Vol. 43, No. 1 (Feb., 1989), pp. 50-54

Tukey, J. W. Exploratory Data Analysis, Addison Wesley (1977, p 33)
 "Some Implementations of the Boxplot"

```
x[1] .. x[n] == series[0] ... series[n - 1] q1_ = (1 - g) * x[j] + g * [j+1];
```

Fences (beyond which lie outliers) are at $q1 - k * (q3 - q1)$ and $q3 + k * (q3 - q1)$ commonly $k = 1.5$, but can be 2.
 Extreme outlier usually set at $k = 3$.

14. `svg_boxplot_series & extreme_outlier_color(const svg_color & color);`

Set fill color of extreme outlier line in box.

Returns: Reference to `svg_boxplot_series` to make chainable.

15. `svg_color extreme_outlier_color();`

Returns: fill color of extreme outlier line in box.

16. `svg_boxplot_series & extreme_outlier_fill(const svg_color & color);`

Set fill color of extreme outlier line in box.

Returns: Reference to `svg_boxplot_series` to make chainable.

17. `svg_color extreme_outlier_fill();`

Returns: fill color of extreme outlier line in box.

18. `svg_boxplot_series & extreme_outlier_shape(point_shape shape);`

Set shape of extreme outlier marker.

Returns: Reference to `svg_boxplot_series` to make chainable.

19. `point_shape extreme_outlier_shape();`

Returns: shape of extreme outlier marker.

20. `svg_boxplot_series & extreme_outlier_size(int size);`

Set size of extreme outlier marker.

Returns: Reference to `svg_boxplot_series` to make chainable..

21. `int extreme_outlier_size();`

Returns: Size of extreme outlier marker.

22. `svg_boxplot_series & max_whisker_color(const svg_color & col);`

Set color of minimum whisker.

Returns: Reference to `svg_boxplot_series` to make chainable.

23. `svg_color max_whisker_color();`

Returns: color of maximum whisker.

24. `svg_boxplot_series & max_whisker_width(double l);`

Returns: line width of maximum whisker.

Reference to `svg_boxplot_series` to make chainable.

25. `double max_whisker_width();`

Returns: line width of maximum whisker.

26. `svg_boxplot_series & median_color(const svg_color & color);`

Set color of median line in box.

Returns: Reference to `svg_boxplot_series` to make chainable.

27. `svg_color median_color();`

Returns: color of median line in box.

28. `svg_style & median_style();`

Returns: median style.

29. `svg_boxplot_series & median_style(svg_style & ms);`

Set entire median style.

Returns: Reference to `svg_boxplot_series` to make chainable.

30. `svg_boxplot_series & median_width(double l);`

Set width of median line in box.

Returns: Reference to `svg_boxplot_series` to make chainable..

31. `double median_width();`

Returns: Width of median line in box.

32. `svg_boxplot_series & min_whisker_color(const svg_color & col);`

Set color of minimum whisker.

Returns: Reference to `svg_boxplot_series` to make chainable.

33. `svg_color min_whisker_color();`

Returns: color of minimum whisker.

34. `svg_boxplot_series & min_whisker_width(double l);`

Set line width of minimum whisker.

Returns: Reference to `svg_boxplot_series` to make chainable..

35. `double min_whisker_width();`

Returns: line width of minimum whisker.

36. `svg_boxplot_series & outlier_color(const svg_color & color);`

Set color of outlier line in box.

Returns: Reference to `svg_boxplot_series` to make chainable.

37. `svg_color outlier_color();`

Returns: color of outlier line in box.

38. `svg_boxplot_series & outlier_fill(const svg_color & color);`

Set fill color of mild outlier line in box.

Returns: Reference to `svg_boxplot_series` to make chainable.

39. `svg_color outlier_fill();`

Returns: fill color of mild outlier line in box.

40. `svg_boxplot_series & outlier_shape(point_shape shape);`

Set shape of outlier marker.

Returns: Reference to `svg_boxplot_series` to make chainable.

41. `point_shape outlier_shape();`

Returns: shape of outlier marker.

42. `svg_boxplot_series & outlier_size(int size);`

Set size of outlier marker.

Returns: Reference to `svg_boxplot_series` to make chainable.

43. `int outlier_size();`

Returns: Size of outlier marker.

44. `plot_point_style` & `outlier_style()`;

Get current outlier style.

Returns: `outlier_style`.

45. `svg_boxplot_series` & `outlier_style(plot_point_style & os)`;

Set entire outlier style..

Set entire outlier style.

Returns: Reference to `svg_boxplot_series` to make chainable.

46. `svg_boxplot_series` & `quartile_definition(int def)`;

set Choice of H&F quartile definition.

Returns: Reference to `svg_boxplot_series` to make chainable.

47. `int quartile_definition()`;

Returns: Choice of H&F quartile definition.

48. `svg_boxplot_series` & `title(const std::string & t)`;

Set title of a **data** series.

Returns: Reference to `svg_boxplot_series` to make chainable.

49. `const std::string title()`;

Obtain current series title.

Returns: title of a **data** series.

50. `svg_boxplot_series` & `whisker_length(double l)`;

Set minimum and maximum whisker length.

Returns: Reference to `svg_boxplot_series` to make chainable.

51. `double whisker_length()`;

Get current whisker length (Applies to BOTH min and max whisker).

Returns: Both minimum and maximum whisker length.

Header <[boost/svg_plot/svg_color.hpp](#)>

SVG standard names of colors, and functions to create and output colors.

9 Feb 2009

Jacob Voytko & Paul A. Bristow

```

namespace boost {
namespace svg {
class svg_color;

enum svg_color_constant;

svg_color color_array; // SVG standard colors.,
void constant_to_rgb(svg_color_constant, unsigned char &, unsigned char &,
                     unsigned char &);

svg_color constant_to_rgb(svg_color_constant);
bool is_blank(const svg_color &);

bool operator!=(const svg_color &, const svg_color &);

std::ostream & operator<<(std::ostream &, const svg_color &);

bool operator==(const svg_color &, const svg_color &);

}
}

```

Class `svg_color`

`boost::svg::svg_color` — SVG standard colors, see also `enum svg_color_constant`.

Synopsis

```

// In header: <boost/svg_plot/svg_color.hpp>

class svg_color {
public:
    // construct/copy/destruct
    svg_color(int, int, int);
    svg_color(bool);
    svg_color(svg_color_constant);

    // public member functions
    unsigned int blue() const;
    unsigned int green() const;
    bool is_blank() const;
    bool operator!=(const svg_color &);

    bool operator==(const svg_color &);

    unsigned int red() const;
    void write(std::ostream &);

    // public data members
    unsigned char b_; // blue unsigned char provides range [0 to 255].
    unsigned char g_; // green unsigned char provides range [0 to 255].
    bool is_blank_; // true means "Not to be displayed" a 'pseudo-color'. If is_blank_ == true should write output □
    to SVG XML file as "none".
    unsigned char r_; // red unsigned char provides range [0 to 255];
};


```

Description

`svg_color` is the struct that contains information about RGB colors. For the constructor, the SVG standard specifies that numbers outside the normal rgb range are to be accepted, but are constrained to acceptable range of integer values [0, 255].

svg_color public construct/copy/destruct

1. `svg_color(int red, int green, int blue);`

Construct an SVG color from RGB values.

Constrain rgb to [0 .. 255]. Default is to construct a 'pseudo-color' blank.

2. `svg_color(bool is);`

Constructor from bool permits `svg_color` my_blank(false) as a (non-)color.

with same effect as `svg_color` my_blank(blank); `svg_color(true)` means default (black?) `svg_color(false)` means blank. For example: `plot.area_fill(false)` will be a blank == no fill. `plot.area_fill(true)` will be a default(black) fill.

3. `svg_color(svg_color_constant col);`

Set a color (including blank) using the SVG 'standard' colors defined in enum `boost::svg::svg_color_constant`

svg_color public member functions

1. `unsigned int blue() const;`

return blue component of color [0, 255]

2. `unsigned int green() const;`

return green component of color [0, 255]

3. `bool is_blank() const;`

Returns: true if color is blank.

4. `bool operator!=(const svg_color & rhs);`

Compare colors (for not equal).

5. `bool operator==(const svg_color & rhs);`

Compare colors (for equal).

6. `unsigned int red() const;`

return red component of color [0, 255]

7. `void write(std::ostream & os);`

Write to ostream a color in svg format.

Usage: `my_color.write(cout);` Outputs: `rgb(127,255,212)`

Note lower case (whereas operator<< uses uppercase).

Type `svg_color_constant`

`boost::svg::svg_color_constant` — Colors that have SVG standard special names.

Synopsis

// In header: <[boost/svg_plot/svg_color.hpp](#)>

```
enum svg_color_constant { aliceblue, antiquewhite, aqua, aquamarine, azure,
    beige, bisque, black, blanchedalmond, blue,
    blueviolet, brown, burlywood, cadetblue, chartreuse,
    chocolate, coral, cornflowerblue, cornsilk, crimson,
    cyan, darkblue, darkcyan, darkgoldenrod, darkgray,
    darkgreen, darkgrey, darkkhaki, darkmagenta,
    darkolivedgreen, darkorange, darkorchid, darkred,
    darksalmon, darkseagreen, darkslateblue,
    darkslategray, darkslategrey, darkturquoise,
    darkviolet, deeppink, deepskyblue, dimgray, dimgrey,
    dodgerblue, firebrick, floralwhite, forestgreen,
    fuchsia, gainsboro, ghostwhite, gold, goldenrod,
    gray, grey, green, greenyellow, honeydew, hotpink,
    indianred, indigo, ivory, khaki, lavender,
    lavenderblush, lawngreen, lemonchiffon, lightblue,
    lightcoral, lightcyan, lightgoldenrodyellow,
    lightgray, lightgreen, lightgrey, lightpink,
    lightsalmon, lightseagreen, lightskyblue,
    lightslategray, lightslategrey, lightsteelblue,
    lightyellow, lime, limegreen, linen, magenta,
    maroon, mediumaquamarine, mediumblue, mediumorchid,
    mediumpurple, mediumseagreen, mediumslateblue,
    mediumspringgreen, mediumturquoise, mediumvioletred,
    midnightblue, mintcream, mistyrose, moccasin,
    navajowhite, navy, oldlace, olive, olivedrab,
    orange, orangered, orchid, palegoldenrod, palegreen,
    paleturquoise, palevioletred, papayawhip, peachpuff,
    peru, pink, plum, powderblue, purple, red,
    rosybrown, royalblue, saddlebrown, salmon,
    sandybrown, seagreen, seashell, sienna, silver,
    skyblue, slateblue, slategray, slategrey, snow,
    springgreen, steelblue, tanned, teal, thistle,
    tomato, turquoise, violet, wheat, white, whitesmoke,
    yellow, yellowgreen, blank };
```

Description

SVG standard names for some colors. See <http://www.w3.org/TR/SVG/types.html#ColorKeywords>.

The reason that the underscore separator convention does not match the normal Boost format is that these names that are specified by the SVG standard. <http://www.w3.org/TR/SVG/types.html#ColorKeywords> color "tan" is also renamed to "tanned" to avoid clash with global function name tan in math.h.

Global color_array

`boost::svg::color_array` — SVG standard colors.,

Synopsis

// In header: <[boost/svg_plot/svg_color.hpp](#)>

```
svg_color color_array;
```

Description

See Also:

`svg_color_constant`

Function `constant_to_rgb`

`boost::svg::constant_to_rgb`

Synopsis

```
// In header: <boost/svg_plot/svg_color.hpp>

void constant_to_rgb(svg_color_constant c, unsigned char &r,
                     unsigned char &g, unsigned char &b);
```

Description

Convert a named SVG standard color, see enum `boost::svg::svg_color_constant` to update three variables (r, g, b) holding red, green and blue values. Asserts that c NOT the blank color.

Convert a named SVG standard color, see enum `boost::svg::svg_color_constant` to update three variables (r, g, b) holding red, green and blue values. Asserts that c NOT the blank color.

Function `constant_to_rgb`

`boost::svg::constant_to_rgb`

Synopsis

```
// In header: <boost/svg_plot/svg_color.hpp>

svg_color constant_to_rgb(svg_color_constant c);
```

Description

Convert a svg color constant enum `boost::svg::svg_color_constant` to a `svg_color`.

Returns: `svg_color` Example: `constant_to_rgb(4)` or `constant_to_rgb(aquamarine)` gives `svg_color(127, 255, 212)` // aquamarine.

Function `is_blank`

`boost::svg::is_blank`

Synopsis

```
// In header: <boost/svg_plot/svg_color.hpp>

bool is_blank(const svg_color &col);
```

Description

Returns: true if color is blank.

Function operator!=

boost::svg::operator!=

Synopsis

```
// In header: <boost/svg_plot/svg_color.hpp>

bool operator!=(const svg_color & lhs, const svg_color & rhs);
```

Description

Compare colors (for not equal).

Function operator<<

boost::svg::operator<<

Synopsis

```
// In header: <boost/svg_plot/svg_color.hpp>

std::ostream & operator<<(std::ostream &, const svg_color &);
```

Description

Output color to stream as RGB. See boost::svg::svg_color_constant

for example: "RGB(138, 43 , 226)" for blueviolet. This comment does not appear - for reasons entirely unclear.

Usage: `svg_color my_color(127, 255, 212); cout << "my_color" << my_color << endl;` Outputs: my_color RGB(127,255,212) `cout << "magenta" << svg_color(magenta) << endl;` but caution! `cout << magenta << endl;` outputs 85 because magenta is an enum
boost::svg::svg_color_constant !

Output color to stream as RGB. See boost::svg::svg_color_constant

for example: "RGB(138, 43 , 226)" for blueviolet. This comment does not appear - for reasons entirely unclear.

Usage: `svg_color my_color(127, 255, 212); cout << "my_color" << my_color << endl;` Outputs: my_color RGB(127,255,212) `cout << "magenta" << svg_color(magenta) << endl;` but caution! `cout << magenta << endl;` outputs 85 because magenta is an enum
boost::svg::svg_color_constant !

Function operator==

boost::svg::operator==

Synopsis

```
// In header: <boost/svg_plot/svg_color.hpp>

bool operator==(const svg_color &lhs, const svg_color &rhs);
```

Description

Compare colors (for equal).

Header <boost/svg_plot/svg_style.hpp>

Styles for SVG specifying font, sizes, shape, color etc for text, values, lines, axes etc.

SVG style information is fill, stroke, width, line & bezier curve. This module provides struct plot_point_style & struct plot_line_style and class svg_style holding the styles. See <http://www.w3.org/TR/SVG11/styling.html>

Mar 2009

Jacob Voytko and Paul A. Bristow

```

namespace boost {
namespace svg {
    class axis_line_style;
    class bar_style;
    class box_style;
    class histogram_style;
    class plot_line_style;
    class plot_point_style;
    class svg_style;
    class text_style;
    class ticks_labels_style;
    class value_style;

    // Options for bar to draw bar charts.
    enum bar_option { y_block == -2, y_stick == -1, no_bar == 0,
                      x_stick == +1, x_block == +2 };

    // dimension of plot. (Used so that an axis knows what type it is, or none == N).
    enum dim { N == 0, X == 1, Y == 2 };

    // options for histograms.
    enum histogram_option { no_histogram == 0, column == +1 };

    // The place for ticks value labels on the axis.
    enum place { left_side == -1, on_axis == 0, right_side == +1,
                 bottom_side == -1, top_side == +1 };

    // used for marking a data point.
    enum point_shape { none == 0, circlet, square, point, egg, unc_ellipse,
                       vertical_line, horizontal_line, vertical_tick,
                       horizontal_tick, cone, triangle, star, lozenge,
                       diamond, heart, club, spade, asterisk, cross, symbol };

    // Rotation of text (in degrees clockwise from horizontal).
    enum rotate_style { horizontal == 0, slopeup == -30, uphill == -45,
                        steepup == -60, upward == -90, backup == -135,
                        leftward == -180, rightward == 360,
                        slopedownhill == 30, downhill == 45,
                        steepdown == 60, downward == 90, backdown == 135,
                        upsidedown == 180 };

    text_style no_style;    // Text style that uses all constructor defaults.
    static const double wh; // font text width/height ratio.
    const char * default_font("Lucida Sans Unicode");

    // plot_point_style that uses all the defaults.
    plot_point_style default_plot_point_style();
    bool operator!=(const text_style &, const text_style &);
    std::ostream & operator<<(std::ostream &, const svg_style &);
    std::ostream & operator<<(std::ostream &, const text_style &);
    std::ostream & operator<<(std::ostream &, plot_point_style);
    std::ostream & operator<<(std::ostream &, plot_line_style);
    bool operator==(const text_style &, const text_style &);
    double string_svg_length(const std::string &, const text_style &);

    const std::string strip_e0s(std::string);
}
}

```

Class axis_line_style

`boost::svg::axis_line_style` — Style of the X or Y-axes lines.

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

class axis_line_style {
public:
    // construct/copy/destruct
    axis_line_style(dim = X, double = -10., double = +10.,
                    const svg_color = black, double = 1, int = 0, bool = true,
                    bool = false, bool = true, double = -1);

    // public member functions
    bool axis_line_on() const;
    axis_line_style & color(const svg_color &);
    svg_color color();
    bool label_on() const;
    axis_line_style & label_on(bool);
    bool label_units_on() const;
    axis_line_style & label_units_on(bool);
    axis_line_style & position(int);
    double position();
    axis_line_style & width(double);
    double width();

    // public data members
    double axis_; // Depending on value of dim, either X-axis (y = 0) transformed into SVG Y coordinates or Y-axis
is (x = 0) transformed into SVG X coordinates (-1 if not calculated yet).
    bool axis_line_on_; // Draw an X horizontal or Y vertical axis line.
    int axis_position_;
    double axis_width_; // Axis line width.
    svg_color color_; // Axis line (stroke) color.
    dim dim_; // None, X or Y.
    bool label_on_; // Label axis with text - example: "length".
    bool label_units_on_; // Label axis units, example: "cm".
    double max_; // maximum Y value (Cartesian units).
    double min_; // minimum X value (Cartesian units).
};

};
```

Description

(But NOT the ticks and value labels because different styles for X and Y-axes are possible).

axis_line_style public construct/copy/destruct

1.

```
axis_line_style(dim d = X, double min = -10., double max = +10.,
                  const svg_color col = black, double width = 1,
                  int axis_position = 0, bool label_on = true,
                  bool label_units_on = false, bool axis_lines_on = true,
                  double axis = -1);
```

Constructor that provides defaults all axis style items.

axis_line_style public member functions

1.

```
bool axis_line_on() const;
```

If returns true, then either an X or a Y axis line to be drawn.

2. `axis_line_style & color(const svg_color & color);`

Set color of an axis line.

Returns: `plot_line_style&` to make chainable.

3. `svg_color color();`

Returns: color of an axis line.

4. `bool label_on() const;`

If returns true, then axis to be labelled, for example "X axis".

5. `axis_line_style & label_on(bool is);`

If set true, then axis to be labelled with the label, for example "X axis" (but default "").

Returns: `plot_line_style&` to make chainable.

6. `bool label_units_on() const;`

If returns true, then axis to be labelled with unit, for example "(mm)"

7. `axis_line_style & label_units_on(bool is);`

If set true, then axis to be labelled with the units label, for example "(mm)" (but default "").

Returns: `plot_line_style&` to make chainable.

8. `axis_line_style & position(int pos);`

How the axes intersect.

Returns: `plot_line_style&` to make chainable.

9. `double position();`

Returns: How the axes intersect.

`enum x_axis_intersect {bottom = -1, x_intersects_y = 0, top = +1}; enum y_axis_intersect {left = -1, y_intersects_x = 0, right = +1};` If axes look like an L, then is bottom left. If a T then y intersects and X is at bottom.

10. `axis_line_style & width(double w);`

Set width of an axis line.

Returns: `plot_line_style&` to make chainable.

11. `double width();`

Returns: width of an axis line.

axis_line_style public public data members

1. `int axis_position_;`

How the axes intersect with values as below:

`enum x_axis_intersect {bottom = -1, x_intersects_y = 0, top = +1}; enum y_axis_intersect {left = -1, y_intersects_x = 0, right = +1};` If axes look like an L, then is bottom left. If a T then y intersects and X is at bottom.

Class bar_style

`boost::svg::bar_style` — Style (color, width, fill) of histogram bars.

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

class bar_style {
public:
    // construct/copy/destruct
    bar_style(const svg_color &= black, const svg_color &= true, double = 2,
              bar_option = no_bar);

    // public member functions
    bar_style & area_fill(const svg_color &);
    svg_color & area_fill();
    bar_style & bar_opt(bar_option);
    double bar_opt();
    bar_style & color(const svg_color &);
    svg_color & color();
    bar_style & width(double);
    double width();

    // public data members
    svg_color area_fill_; // Fill color from line to axis.
    bar_option bar_option_; // stick or bar.
    svg_color color_; // Color of line (stroke) (no fill color for lines).
    double width_; // Width of bar, not enclosing line width.
};
```

Description**bar_style public construct/copy/destruct**

1. `bar_style(const svg_color & col = black, const svg_color & acol = true,
 double width = 2, bar_option opt = no_bar);`

Construct with defaults for all member variables.

Constructor, setting defaults for all member variables.

bar_style public member functions

1. `bar_style & area_fill(const svg_color & f);`

Returns: `box_style&` to make chainable.

2. `svg_color & area_fill();`

Returns: bar rectangle fill color.

3. `bar_style & bar_opt(bar_option option);`

Returns: `box_style`& to make chainable.

4. `double bar_opt();`

Returns: If to use stick or bar for histograms.

5. `bar_style & color(const svg_color & f);`

Returns: `box_style`& to make chainable.

6. `svg_color & color();`

Returns: Color of bar line or enclosing line.

7. `bar_style & width(double w);`

Returns: `box_style`& to make chainable.

8. `double width();`

Returns: Width of bar, not enclosing line width.

Class `box_style`

`boost::svg::box_style` — a rectangular box. (Used for boxplot image and plot window).

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

class box_style {
public:
    // construct/copy/destruct
    box_style(const svg_color & = black, const svg_color & = white, double = 1,
              double = 4., bool = true, bool = false);

    // public member functions
    bool border_on() const;
    box_style & border_on(bool);
    box_style & fill(const svg_color &);
    svg_color fill();
    bool fill_on() const;
    box_style & fill_on(bool);
    box_style & margin(double);
    double margin();
    box_style & stroke(const svg_color &);
    svg_color stroke();
    box_style & width(double);
    double width();

    // public data members
    bool border_on_; // Display the border of the box.
    svg_color fill_; // Box fill color.
    bool fill_on_; // Color fill the box.
    double margin_; // Marginal (pixels) space around the box (inside or out).
    svg_color stroke_; // Box line (stroke) color.
    double width_; // plot border rectangle width.
};


```

Description

box_style public construct/copy/destruct

1. `box_style(const svg_color & scolor = black, const svg_color & fcolor = white,
 double width = 1, double margin = 4., bool border_on = true,
 bool fill_on = false);`

Constructor to set parameters but provides defaults for all variables.

box_style public member functions

1. `bool border_on() const;`

Returns: If the box border should be shown.

2. `box_style & border_on(bool is);`

Set true if the box border should be shown.

Returns: `box_style&` to make chainable.

3. `box_style & fill(const svg_color & color);`

Set fill color for box.

Returns: `box_style&` to make chainable.

4. `svg_color fill();`

Returns: Fill color for box.

5. `bool fill_on() const;`

Returns: if the box should be filled.

6. `box_style & fill_on(bool is);`

Set true if the box should be filled.

Returns: `box_style&` to make chainable.

7. `box_style & margin(double w);`

Set marginal (default pixels) space around the box (inside or out).

Returns: `box_style&` to make chainable.

8. `double margin();`

Returns: marginal (default pixels) space around the box (inside or out).

9. `box_style & stroke(const svg_color & color);`

Set (stroke) color for box outline.

Returns: `box_style&` to make chainable.

10. `svg_color stroke();`

Returns: (stroke) color for box outline.

11. `box_style & width(double w);`

Set width for box.

Returns: `box_style&` to make chainable.

12. `double width();`

Returns: width for box.

Class `histogram_style`

`boost::svg::histogram_style` — Histogram options.

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

class histogram_style {
public:
    // construct/copy/destruct
    histogram_style(histogram_option = no_histogram);

    // public member functions
    histogram_style & histogram(histogram_option);
    double histogram();

    // public data members
    histogram_option histogram_option_; // default bar, no_histogram or column.
};


```

Description

histogram_style public construct/copy/destruct

1. `histogram_style(histogram_option opt = no_histogram);`

Set any histogram option.

Constructor providing defaults for all private data. Line width and area-fill are taken from the [plot_line_style](#) style.

histogram_style public member functions

1. `histogram_style & histogram(histogram_option opt);`

Set any histogram option.

Histogram to be shown as sticks or bars.

Returns: `box_style&` to make chainable.

2. `double histogram();`

<

Returns: Histogram option.

Returns: Histogram option.

Class **plot_line_style**

`boost::svg::plot_line_style` — line joining data series values.

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

class plot_line_style {
public:
    // construct/copy/destruct
    plot_line_style(const svg_color & = black, const svg_color & = blank,
                    double = 2, bool = true, bool = false);

    // public member functions
    plot_line_style & area_fill(const svg_color &);
    svg_color & area_fill();
    bool bezier_on() const;
    plot_line_style & bezier_on(bool);
    plot_line_style & color(const svg_color &);
    svg_color & color();
    bool line_on() const;
    plot_line_style & line_on(bool);
    plot_line_style & width(double);
    double width();

    // public data members
    svg_color area_fill_; // Fill color from line to axis. == false means color.is_blank = true, or = blank.
    bool bezier_on_; // If true, data points will be joined by bezier curved line(s).
    bool line_on_; // If true, data points will be joined by straight line(s).
    svg_color stroke_color_; // Stroke color of line. (no fill color for lines)
    double width_; // Width of line joining data series values.
};

};
```

Description

plot_line_style public construct/copy/destruct

1. `plot_line_style(const svg_color & col = black,
 const svg_color & fill_col = blank, double width = 2,
 bool line_on = true, bool bezier_on = false);`

Constructor to set plot line style, but providing defaults for all member data.

plot_line_style public member functions

1. `plot_line_style & area_fill(const svg_color & f);`

Set if area under line joining data points is to be color filled.

Returns: `plot_line_style&` to make chainable.

2. `svg_color & area_fill();`

Returns: if area under line joining data points is to be color filled.

3. `bool bezier_on() const;`

Returns: true if bezier curved line(s) are to join data points.

4. `plot_line_style & bezier_on(bool is);`

Set true if bezier curved line(s) are to join data points.

Returns: `plot_line_style&` to make chainable.

5. `plot_line_style & color(const svg_color & f);`

Set color of line(s) joining data points.

Returns: `plot_line_style&` to make chainable.

6. `svg_color & color();`

Returns: color of line(s) joining data points.

7. `bool line_on() const;`

Returns: True if line(s) will join data points.

8. `plot_line_style & line_on(bool is);`

Set true if line(s) are to join data points.

Returns: `plot_line_style&` to make chainable.

9. `plot_line_style & width(double w);`

Set width of line(s) joining data points.

Returns: `plot_line_style&` to make chainable.

10. `double width();`

Returns: width of line(s) joining data points.

Class `plot_point_style`

`boost::svg::plot_point_style` — Shape, color, of data point markers.

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

class plot_point_style {
public:
    // construct/copy/destruct
    plot_point_style(const svg_color &= black, const svg_color &= blank,
                     int = 5, point_shape = circlet, const std::string &= "X");

    // public member functions
    plot_point_style & fill_color(const svg_color &);
    svg_color & fill_color();
    plot_point_style & shape(point_shape);
    point_shape shape();
    plot_point_style & size(int);
    int size();
    plot_point_style & stroke_color(const svg_color &);
    svg_color & stroke_color();
    plot_point_style & style(text_style);
    text_style & style() const;
    plot_point_style & symbols(const std::string &);

    // public data members
    svg_color fill_color_; // Fill color of the centre of the shape.
    point_shape shape_; // shape: round, square, point...
    bool show_x_value_; // If true, show the X value like "1.2" near the point. (If both true, then show both X and □ Y as a pair like "1.2, 3.4".)
    bool show_y_value_; // If true, show the Y value like "3.4" near the point. (If both true, then show both X and □ Y as a pair like "1.2, 3.4".)
    int size_; // Diameter of circle, height of square, font_size ...
    svg_color stroke_color_; // Color of circumference of shape.
    std::string symbols_; // Unicode symbol(s) (letters, digits, squiggles etc).

Caution: not all Unicode symbols are output by all browsers!

Example: U2721 is Star of David or hexagram, see http://en.wikipedia.org/wiki/Hexagram, symbols("✡") □
Positioning of symbols (especially > 1 symbols) may be imprecise.
    text_style symbols_style_; // font, size, decoration of symbols.
};


```

Description

(optional value & uncertainty) not implemented yet.

plot_point_style public construct/copy/destruct

1. `plot_point_style(const svg_color & stroke = black,
 const svg_color & fill = blank, int size = 5,
 point_shape shape = circlet,
 const std::string & symbols = "X");`

Unicode symbol(s) (letters, digits, squiggles etc).

plot_point_style public member functions

1. `plot_point_style & fill_color(const svg_color & f);`

Set fill color of shape or symbol used to mark data value plot point(s).

Returns: [plot_point_style](#)& to make chainable.

2. [svg_color](#) & [fill_color](#)();

Returns: fill color of shape or symbol used to mark data value plot point(s).

3. [plot_point_style](#) & [shape\(point_shape s\)](#);

Set shape used to mark data value plot point(s).

Returns: [plot_point_style](#)& to make chainable.

4. [point_shape shape](#)();

Returns: shape used to mark data value plot point(s).

5. [plot_point_style](#) & [size\(int i\)](#);

Set size of shape or symbol used to mark data value plot point(s).

< Diameter of circle, height of square, font_size ...

Returns: [plot_point_style](#)& to make chainable.

6. [int size](#)();

Returns: size of shape or symbol used to mark data value plot point(s).

7. [plot_point_style](#) & [stroke_color\(const svg_color & f\)](#);

Set stroke color of shape or symbol used to mark data value plot point(s).

Returns: [plot_point_style](#)& to make chainable.

8. [svg_color](#) & [stroke_color](#)();

Returns: stroke color of shape or symbol used to mark data value plot point(s).

9. [plot_point_style](#) & [style\(text_style ts\)](#);

Returns: [plot_point_style](#)& to make chainable.

10. [text_style](#) & [style\(\)](#) const;

Returns: [text_style](#)& To allow control of symbol font, size, decoration etc.

11. [plot_point_style](#) & [symbols\(const std::string s\)](#);

Override default symbol "X" - only effective if .shape(symbol) used.

Returns: [plot_point_style](#)& to make chainable.

12 std::string & symbols();

Returns: plot point marking symbol (only effective if .shape(symbol) used).

Class `svg_style`

`boost::svg::svg_style` — basic SVG stroke, fill colors and width, and their switches.

Synopsis

// In header: <[boost/svg_plot/svg_style.hpp](#)>

```
class svg_style {
public:
    // construct/copy/destruct
    svg_style();
    svg_style(const svg_color &, const svg_color &, unsigned int);

    // public member functions
    svg_style & fill_color(const svg_color &);
    svg_color fill_color() const;
    svg_style & fill_on(bool);
    bool fill_on() const;
    bool operator!=(svg_style &);
    bool operator==(svg_style &);
    svg_style & stroke_color(const svg_color &);
    svg_color stroke_color() const;
    svg_style & stroke_on(bool);
    bool stroke_on() const;
    svg_style & stroke_width(double);
    double stroke_width() const;
    svg_style & width_on(bool);
    bool width_on() const;
    void write(std::ostream &);
```

};

Description

This is the style information for any group (g) tag. This could be expanded to include more data from the SVG standard.

There are some strange effects for text on some browsers (Firefox especially) when only stroke is specified. fill is interpreted as black, and the font outline is fuzzy and bolder. `<g id="title" stroke="rgb(255,0,0)">` .. is red border and black fill. (because created as a graphic not a builtin font?) `<g id="title" fill="rgb(255,0,0)">` .. is red sharp font. `<g id="title" stroke="rgb(255,0,0)" fill="rgb(255,0,0)">` red and red fill also fuzzy. So for text, only specific the fill unless a different outline is really wanted. Defaults for text provide a built-in glyph, for example for title: `<g id="title"> <text x="250" y="36" text-anchor="middle" font-size="18" font-family="Verdana"> Plot of data </text> </g>` and this is not a graphic.

`svg_style` public construct/copy/destruct

1. `svg_style();`

2. `svg_style(const svg_color & stroke, const svg_color & fill, unsigned int width);`

Construct `svg_style` with specified fill and stroke colors, and width.

svg_style public member functions

1. `svg_style & fill_color(const svg_color & col);`

Set fill color (and set fill on true, unless color is blank).

Returns: `svg_style&` to make chainable.

2. `svg_color fill_color() const;`

Returns: SVG fill color.

3. `svg_style & fill_on(bool is);`

Set fill is wanted.

Returns: `svg_style&` to make chainable.

4. `bool fill_on() const;`

Returns: true if fill wanted.

5. `bool operator!=(svg_style & s);`

Compare `svg_styles` (for inequality).

6. `bool operator==(svg_style & s);`

Compare `svg_styles`.

7. `svg_style & stroke_color(const svg_color & col);`

Set stroke color (and set stroke on).

Returns: `svg_style&` to make chainable.

8. `svg_color stroke_color() const;`

Returns: SVG stroke color.

9. `svg_style & stroke_on(bool is);`

Set true if SVG stroke is wanted.

Returns: `svg_style&` to make chainable.

10. `bool stroke_on() const;`

Returns: true if SVG stroke is on.

11. `svg_style & stroke_width(double width);`

Set stroke width (and set width on).

Returns: `svg_style&` to make chainable.

12. `double stroke_width() const;`

Returns: SVG stroke width.

13. `svg_style & width_on(bool is);`

Set true to use SVG stroke width.

Returns: `svg_style&` to make chainable.

14. `bool width_on() const;`

Returns: true if to use SVG stroke width.

15. `void write(std::ostream & os);`

Write any stroke, fill colors and/or width info to SVG XML document.

Example output: <g id="yMinorTicks" stroke="rgb(0,0,0)" stroke-width="1">

Class `text_style`

`boost::svg::text_style` — Font family, font size, weight, style, stretch & decoration.

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

class text_style {
public:
    // construct/copy/destruct
    text_style(int = 12, const std::string & = default_font,
               const std::string & = "", const std::string & = "",
               const std::string & = "", const std::string & = "");

    // public member functions
    text_style & font_decoration(const std::string &);
    const std::string & font_decoration() const;
    text_style & font_family(const std::string &);
    const std::string & font_family() const;
    text_style & font_size(unsigned int);
    int font_size() const;
    text_style & font_stretch(const std::string &);
    const std::string & font_stretch() const;
    text_style & font_style(const std::string &);
    const std::string & font_style() const;
    text_style & font_weight(const std::string &);
    const std::string & font_weight() const;
    bool operator!=(const text_style &);
    bool operator==(const text_style &);

    // public data members
    std::string decoration_; // Font decoration, examples: "underline" | "overline" | "line-through".
    std::string font_family_; // Font family, examples: "Arial", "Times New Roman", "Verdana", "Lucida Sans Uni
                           // code".
    int font_size_; // Font size (SVG units, default pixels).
    std::string stretch_; // Font stretch, examples: normal | wider | narrower.
    std::string style_; // Font weight, examples: normal | bold | italic | oblique.
    std::string weight_; // Font style, examples: "bold", "normal".
};


```

Description

text_style public construct/copy/destruct

1. `text_style(int size = 12, const std::string & font = default_font,
 const std::string & weight = "", const std::string & style = "",
 const std::string & stretch = "",
 const std::string & decoration = "");`

Examples: "underline" | "overline" | "line-through".

Default constructor only sets font size = 20, and leaves other font details as SVG defaults.

text_style public member functions

1. `text_style & font_decoration(const std::string & s);`

Set font decoration. Examples: "underline" | "overline" | "line-through"
http://www.croczilla.com/~alex/conformance_suite/svg/text-deco-01-b.svg tests line-through and underline. But implementation varies.

Returns: reference to `text_style` to make chainable.

2. `const std::string & font_decoration() const;`

Returns: font decoration.

3. `text_style & font_family(const std::string & s);`

Set font family, for example: "Arial", "Times New Roman", "Verdana", "Lucida Sans Unicode".

Default for browser is sans with Firefox & IE but serif with Opera.

See also browser conformance test at

http://www.croczilla.com/~alex/conformance_suite/svg/text-fonts-01-t.svg

which tests three styles of font, serif, sans-serif and mono-spaced.

```
<text font-family="Georgia, 'Minion Web', 'Times New Roman', Times, 'MS PMincho', Heisei-Mincho, serif " x="20" y="80">A
serifed face</text>\n <text font-family="Arial, 'Arial Unicode', 'Myriad Web', Geneva, 'Lucida Sans Unicode', 'MS PGothic',
Osaka, sans-serif " x="20" y="160">A sans-serif face</text>\n <text font-family="Lucida Console', 'Courier New', Courier,
Monaco, 'MS Gothic', Osaka-Mono, monospace" x="20" y="240">A mono (iW) face</text>
```

Returns: reference to `text_style` to make chainable.

4. `const std::string & font_family() const;`

Returns: font family as string.

5. `text_style & font_size(unsigned int i);`

Set font size (svg units usually pixels) default 10. \return `text_style&` to make chainable.

Returns: reference to `text_style` to make chainable.

6. `int font_size() const;`

Returns: font size (svg units, usually pixels).

7. `text_style & font_stretch(const std::string & s);`

Examples: "wider" but implementation by browsers varies. `font-stretch: normal | wider | narrower ...`

Returns: reference to `text_style` to make chainable.

8. `const std::string & font_stretch() const;`

Returns: font stretch, for example: `normal | wider | narrower`.

9. `text_style & font_style(const std::string & s);`

Set font style. Example: `my_text_style.font_style("italic");`

See also browser conformance tests:

http://www.croczilla.com/~alex/conformance_suite/svg/text-fonts-02-t.svg

Returns: reference to `text_style` to make chainable.

10. `const std::string & font_style() const;`

`font-style: normal | bold | italic | oblique.` Example "normal" is default.

Returns: font style.

11. `text_style & font_weight(const std::string & s);`

svg `font-weight: normal | bold | bolder | lighter | 100 | 200 .. 900` Examples: "bold", "normal"
http://www.croczilla.com/~alex/conformance_suite/svg/text-fonts-02-t.svg tests conformance. Only two weights, "bold", "normal", are supported by Firefox, Opera, Inkscape.

Returns: reference to `text_style` to make chainable.

12. `const std::string & font_weight() const;`

Set font weight. Example: `my_text_style.font_style("bold");`

See also browser conformance tests:

http://www.croczilla.com/~alex/conformance_suite/svg/text-fonts-02-t.svg

13. `bool operator!=(const text_style & ts);`

Compare `text_style` for inequality (needed for testing).

14. `bool operator==(const text_style & ts);`

Compare `text_style` for equality (needed for testing).

Class ticks_labels_style

`boost::svg::ticks_labels_style` — Style of the X and Y axes ticks, grids and their tick value labels.

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

class ticks_labels_style {
public:
    // construct/copy/destruct
    ticks_labels_style(dim = X, const text_style & = no_style, double = 10.,
                       double = -10., double = 2., unsigned int = 4);

    // public member functions
    double label_length(double);
    double longest_label();
    int major_value_labels_side() const;
    ticks_labels_style & major_value_labels_side(int);
    bool use_down_ticks() const;
    ticks_labels_style & use_down_ticks(bool);
    bool use_up_ticks() const;
    ticks_labels_style & use_up_ticks(bool);

    // public data members
    dim dim_; // X, Y, or None.
    bool down_ticks_on_; // Draw ticks down from horizontal X-axis line.
    double label_max_length_; // width (in SVG units, pixels) of longest value label text on axis.
    double label_max_space_; // Space (SVG units, pixels) needed for value label adjusted for rotation.
    rotate_style label_rotation_; // Direction axis value labels written.
    bool left_ticks_on_; // Draw ticks left from vertical Y-axis line.
    svg_color major_grid_color_; // Color of major grid lines.
    bool major_grid_on_; // Draw X grid at major ticks.
    double major_grid_width_; // Width of major grid lines.
    double major_interval_; // Stride or interval between major x ticks (Cartesian units).
    svg_color major_tick_color_; // Color (stroke) of tick lines.
    double major_tick_length_; // Length of major tick lines.
    double major_tick_width_; // Width of major tick lines.
    int major_value_labels_side_; // Which side of axis for label values for major ticks.
    double max_; // Maximum x value (Cartesian units).
    double min_; // Minimum x value (Cartesian units).
    svg_color minor_grid_color_; // color of minor grid lines.
    bool minor_grid_on_; // Draw X grid at minor ticks.
    double minor_grid_width_; // Wdith of minor grid lines.
    double minor_interval_; // Interval (Cartesian units) between minor ticks.
    svg_color minor_tick_color_; // Color (stroke) of tick lines.
    double minor_tick_length_; // Length of minor tick lines.
    double minor_tick_width_; // Width of minor tick lines.
    unsigned int num_minor_ticks_; // number of minor ticks, eg 4 gives major 0, minor 1,2,3,4, major 5 (All units □
    in svg units, default pixels).
    bool right_ticks_on_; // Draw ticks right from vertical Y-axis line.
    bool strip_e0s_; // If redundant zero, + and e are to be stripped, for example "+1.000e3" to "1e3".
    int ticks_on_window_or_on_axis_; // Value labels & ticks on a plot window border (rather than on X or Y-axis). □
    For Y-axis -1 = left, 0 = false = on X-axis, +1 = right. Default -1 to left of plot window. For X-axis -1 = bottom, □
    0 = false = on Y-axis, +1 = top. Default -1 below bottom of plot window. 0 = false puts the ticks and their labels □
    on the X or Y axis line which may be in the middle of the plot. For 1D the default overrides the constructor default □
    of -1 below, to tick and value label the X-axis. For 2D the default is left at -1, to use bottom and left of plot win□
    dow to tick and value label X and Y-axis.
    bool up_ticks_on_; // Draw ticks up from horizontal X-axis line.
    std::ios_base::fmtflags value_ioflags_; // IO formatting flags for the axis default std::ios::dec.
    text_style value_label_style_; // text style (font, size...) for value labels.
    int value_precision_; // Precision for tick value labels, usually 3 will suffice.
    svg_color values_color_; // Color of tick values labels.
};


```

Description

But NOT the X and Y axes lines. These can be either on the axis lines or on the plot window edge(s), (because different styles for x and y are possible).

ticks_labels_style public construct/copy/destruct

1. `ticks_labels_style(dim d = X, const text_style & style = no_style,
double max = 10., double min = -10.,
double major_interval = 2.,
unsigned int num_minor_ticks = 4);`

Constructor setting several parameters, but providing default values for all member data.

ticks_labels_style public member functions

1. `double label_length(double value);`

Find the length of label (like "1.23E-5") for a value.

2. `double longest_label();`

Update label_max_length_ with the longest value label as pixels, return the count of digits etc.

3. `int major_value_labels_side() const;`

Returns: side for tick value labels: left (<0), none (==0) or right (>0).

4. `ticks_labels_style & major_value_labels_side(int is);`

Set side for tick value labels: left (<0), none (==0) or right (>0).

Returns: `ticks_labels_style&` to make chainable.

5. `bool use_down_ticks() const;`

Returns: true if to draw ticks down from horizontal X-axis line.

6. `ticks_labels_style & use_down_ticks(bool side);`

Set true if to draw ticks down from horizontal X-axis line.

Returns: `ticks_labels_style&` to make chainable.

7. `bool use_up_ticks() const;`

Returns: true if to draw ticks up from horizontal X-axis line.

8. `ticks_labels_style & use_up_ticks(bool is);`

Set true to draw ticks up from horizontal X-axis line.

Returns: `ticks_labels_style&` to make chainable.

Class value_style

boost::svg::value_style — Data series point value label information, text, color, orientation, (uncertainty & df), name ID string, order in sequence, time and date.

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

class value_style {
public:
    // construct/copy/destruct
    value_style();
    value_style(rotate_style, int, std::ios_base::fmtflags, bool, text_style,
                const svg_color &, const svg_color &, bool, const svg_color &,
                bool, const svg_color &, bool, const svg_color &, bool,
                const svg_color &, bool, const svg_color &, bool,
                const svg_color &, std::string, std::string, std::string);

    // public data members
    svg_color addlimits_color_; // Color for confidence interval.
    bool addlimits_on_; // If an confidence interval is to be added, for example <4.5, 4.8>.
    svg_color datetime_color_; // Color for time and date string".
    bool datetime_on_; // If an time and/or date string to be appended. default == false,.
    svg_color df_color_; // Color for degrees for freedom, for example: 99 in "1.23 +-0.02 (99)".
    bool df_on_; // If a degrees of freedom estimate is to be appended.
    svg_color fill_color_; // Fill color for value.
    svg_color id_color_; // Color for id or name string".
    bool id_on_; // If an id or name string to be appended. default == false,.
    svg_color order_color_; // Color for sequence number #".
    bool order_on_; // If an order in sequence number # to be appended. default == false,.
    svg_color plusminus_color_; // Color for uncertainty, for example: 0.02 in "1.23 +-0.02".
    bool plusminus_on_;
    std::string prefix_; // Prefix to data point value, default none, but typically "[".
    std::string separator_; // Separator between x and y values, if both on same line (none if only X or only Y, or □ Y below X).
    bool strip_e0s_; // If true, then unnecessary zeros and + sign will be stripped to reduce length.
    svg_color stroke_color_; // Stroke color for value.
    std::string suffix_; // Suffix to data point value, default none, but typically "]".
    std::ios_base::fmtflags value_ioflags_; // Control of scientific, fixed, hex etc.
    rotate_style value_label_rotation_; // Direction point value labels written.
    int value_precision_; // Decimal digits of precision of value.
    text_style values_text_style_; // Font etc used for data point value marking.
};


```

Description

For example, to output: 5.123 +- 0.01 (19). Uncertainty and degrees of freedom estimate. Prefix, separator and suffix allow X and Y values to be together on one line, for example

[1.23+- 0.01 (3), 4.56 +-0.2 (10)]

Used in draw_plot_point_values (note plural - not used in singular draw_plot_point_value) where X **value_style** is used to provide the prefix and separator, and Y **value_style** to provide the suffix. Prefix, separator and suffix are ignored when X or Y are shown separately using draw_plot_point_value. "4.5+- 0.01 (3) Second #2, 2012-Mar-13 13:01:00"

value_style public construct/copy/destruct

1. `value_style();`

Default style for a data point value label.

< Constructor Data point value label style (provides default color and font).

Constructor Data point value label style (provides default color and font).

Default constructor initialises all private data.

2.

```
value_style(rotate_style r, int p, std::ios_base::fmtflags f, bool s,
    text_style ts, const svg_color & scol, const svg_color & fcol,
    bool pm, const svg_color & plusminus_color, bool lim,
    const svg_color & addlimits_color, bool df,
    const svg_color & df_color, bool id, const svg_color & id_color,
    bool dt, const svg_color & dt_color, bool ordno,
    const svg_color & ordno_color, std::string pre, std::string sep,
    std::string suf);
```

Constructor setting parameters with some defaults.

value_style public public data members

1.

```
bool plusminus_on_;
```

If an uncertainty estimate is to be appended (as + or - value).

See http://en.wikipedia.org/wiki/Plus-minus_sign

Global no_style

boost::svg::no_style — Text style that uses all constructor defaults.

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>
text_style no_style;
```

Global wh

boost::svg::wh — font text width/height ratio.

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>
static const double wh;
```

Function default_font

boost::svg::default_font

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

const char * default_font("Lucida Sans Unicode");
```

Description

Default font chosen is a Unicode font like ['Lucida Sans Unicode] that has the best chance of ['symbols] being rendered correctly. Used for title, legend, axes ... unless overridden by an explicit font specification.

Function operator!=

boost::svg::operator!=

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

bool operator!=(const text_style & lhs, const text_style & rhs);
```

Description

Compare two `text_style` for equality. Note `operator==` and `operator<<` both needed to use Boost.Test. (But can be avoided with a macro define).

Function operator<<

boost::svg::operator<<

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

std::ostream & operator<<(std::ostream & os, const svg_style & s);
```

Description

Output a string description of a `svg_style`. Usage: `svg_style my_svg_style; cout << my_svg_style << endl;` Outputs: `svg_style(RGB(0,0,0), RGB(0,0,0), 0, no fill, no stroke, no width)`

Function operator<<

boost::svg::operator<<

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

std::ostream & operator<<(std::ostream & os, const text_style & ts);
```

Description

Output a text style as a text string (mainly useful for diagnostic use).

Usage: `text_style` `ts(12, "Arial", "italic", "bold", "", "")`; `cout << t << endl`; Outputs: `text_style(18, "Arial", "italic", "bold", "", "")`

Function operator<<

`boost::svg::operator<<`

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

std::ostream & operator<<(std::ostream & os, plot_point_style p);
```

Description

Output description of data value plot point marker(s).

Example: `plot_point_style` `p`; `cout << p << endl`; Outputs: `point_style(1, RGB(0,0,0), RGB(0,0,0), 10, X)`

Function operator<<

`boost::svg::operator<<`

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

std::ostream & operator<<(std::ostream & os, plot_line_style p);
```

Description

Output description of `plot_line_style`. (mainly useful for diagnosis).

Example Usage: `plot_line_style` `p`; `cout << p << endl`; Outputs: `point_line_style(RGB(0,0,0), blank, line, no bezier)`

Function operator==

`boost::svg::operator==`

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

bool operator==(const text_style &lhs, const text_style &rhs);
```

Description

Compare two `text_style` for equality Note operator== and operator << both needed to use Boost.text. (But can be avoided with a macro define).

Function string_svg_length

`boost::svg::string_svg_length`

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

double string_svg_length(const std::string &s, const text_style &style);
```

Description

<http://www.w3.org/TR/SVG/text.html#FontSizeProperty> Font size is the height of the text's font, so width = wh * font_size.

Even after reading <http://www.w3.org/TR/SVG/fonts.html>,
unclear how to determine the exact width of digits, so an arbitrary average width height ratio wh = 0.7 is used as a good approximation.

Function strip_e0s

`boost::svg::strip_e0s`

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>

const std::string strip_e0s(std::string s);
```

Description

Returns: length of trimmed string (perhaps unchanged).
 length of trimmed string (perhaps unchanged).

Header <[boost/svg_plot/uncertain.hpp](http://www.boost.org/doc/libs/1_65_0/include/boost/svg_plot/uncertain.hpp)>

Class for storing Uncertainties and simple propagation according to a pure Gaussian model.

This simplified version assuming uncorrelated uncertainties (the common case) is based on code by Evan Manning (manning@alumni.caltech.edu) Evan Marshal Manning, C/C++ Users Journal, March 1996 page 29 to 38. original downloaded from <ftp://beowulf.jpl.nasa.gov/pub/manning> This is a simple model of uncertainties, designed to accompany an article published in C/C++

Users Journal March 1996. A fuller collection of even fancier classes also given in UReal.h. And also based on a extended version including uncertainty as standard deviation & its uncertainty as degrees of freedom, and other information about the value added Paul A Bristow from 31 Mar 98.

See Also:

http://en.wikipedia.org/wiki/Plus-minus_sign

Paul A. Bristow

Mar 2009

```
namespace boost {
namespace svg {
template<bool correlated = false> class unc;

static const double plusminus; // Nominal factor of 2 (strictly 1.96) corresponds to 95% confidence limit.
template<bool correlated>
std::ostream & operator<<(std::ostream &, const unc< correlated > &);
template<bool correlated>
std::ostream &
operator<<(std::ostream &,
const std::pair< unc< correlated >, unc< correlated > > &);

template<typename T> float unc_of(T);
template<bool correlated> float unc_of(unc< correlated >);
template<typename T> std::pair< float, float > uncs_of(T);
template<typename T> std::pair< float, float > uncs_of(std::pair< T, T >);

template<typename T>
std::pair< const float, float > uncs_of(std::pair< const T, T >);
template<typename T> double value_of(T);
template<> double value_of(unc< true >);
template<> double value_of(unc< false >);
template<typename T> std::pair< double, double > values_of(T);
template<typename T>
std::pair< double, double > values_of(std::pair< const T, T >);

template<bool correlated>
std::pair< double, double >
values_of(std::pair< unc< correlated >, unc< correlated > >);

}
```

Class template unc

boost::svg::unc — Uncertain class for storing an observed or measured value together with information about its uncertainty (previously called 'error' or 'plusminus', but now deprecated) represented nominally one standard deviation (but displayed as a multiple, usually two standard deviations).

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<bool correlated = false>
class unc : public std::char_traits< char > {
public:
    // construct/copy/destruct
    unc(double = 0., float = -1.f,
        short unsigned = (std::numeric_limits< unsigned short int >::max)(),
        short unsigned = 0U);
    unc & operator=(const unc &);

    // public member functions
    short unsigned deg_free() const;
    void deg_free(short);
    bool operator<(const unc &) const;
    bool operator<=(unc &) const;
    bool operator==(const unc &) const;
    short unsigned types() const;
    void types(short);
    float uncertainty() const;
    void uncertainty(float);
    double value() const;
    void value(double);

    // friend functions
    friend std::ostream & operator<<(std::ostream &, const unc< correlated > &);
    friend std::ostream &
    operator<<(std::ostream &, const std::pair< unc, unc > &);
};

}
```

Description

This version assumes uncorrelated uncertainties (by far the most common case).

See Also:

<http://www.measurementuncertainty.org/>

International Vocabulary of Basic and General Terms in Metrology; ISO/TAG 4 1994
 ISO, Guide to the expression of uncertainty in measurement, ISO, Geneva, 1993.
 Eurochem, Quantifying uncertainty in analytical measurements.

unc public construct/copy/destruct

1. `unc(double v = 0., float u = -1.f,
 short unsigned df = (std::numeric_limits< unsigned short int >::max)(),
 short unsigned ty = 0U);`

Constructor allowing an unc to be constructed from just value providing defaults for all other parameters. Note the defaults so that unspecified variables have 'undefined' status.

2. `unc & operator=(const unc & rhs);`

Assignment simply copies all values, including those with 'undefined' status.

to make chainable.

unc public member functions

1. `short unsigned deg_free() const;`

Returns: Degrees of freedom, usually the number of observations -1.

2. `void deg_free(short unsigned);`

Set degrees of freedom, usually = number of observations -1;

Set degrees of freedom, usually = number of observations -1;

3. `bool operator<(const unc & rhs) const;`

Less operator only compares the value, ignoring any uncertainty information.

4. `bool operator<(unc & rhs) const;`

Less operator only compares the value, ignoring any uncertainty information.

5. `bool operator==(const unc & rhs) const;`

Equality operator only compares the value, ignoring any uncertainty information.

6. `short unsigned types() const;`

Returns: degrees of freedom, usually = number of observations -1;

Returns: Other information about the uncertain value.

7. `void types(short unsigned);`

Set other information about the value.

Set other information about the uncertain value.

8. `float uncertainty() const;`

Returns: estimate of uncertainty, typically one standard deviation.

Returns: Estimate of uncertainty, typically standard deviation.

9. `void uncertainty(float);`

Set estimate of uncertainty, typically standard deviation.

Set estimate of uncertainty, typically standard deviation.

10. `double value() const;`

Returns: most likely value, typically the mean.

Returns: Most likely value, typically the mean.

11. `void value(double);`

Set most likely value, typically the mean.

Set most likely value, typically the mean.

Returns: other information about the value.

unc friend functions

1.

```
friend std::ostream &
operator<<(std::ostream & os, const unc< correlated > & u);
```

Output an value with (if defined) uncertainty and degrees of freedom (and type). For example: "1.23 +/- 0.01 (13)".

Note that the uncertainty is input and stored as one standard deviation, but output multiplied for a user configurable 'confidence factor' plusminus, default is two standard deviation for about 95% confidence (but could also be one for 67% or 3 for 99% confidence).

Output a single value with (if defined) uncertainty and degrees of freedom (and type). For example: "1.23 +/- 0.01 (13)". /details Note that the uncertainty is input and stored as one standard deviation, but output multiplied for a user configurable 'confidence factor' plusminus, default two for about 95% confidence (but could also be one for 67% or 3 for 99% confidence).

Note that the plus or minus can be output using several methods.

256 character 8-bit codepage plusminus symbol octal 361, or os << char(241) decimal 241 or os << char(0xF1) hexadecimal F1, or os << " ±" Unicode space plusminus glyph, or os << "+or-" << u.uncertainty_; Plain ANSI 7 bit code chars.

2.

```
friend std::ostream &
operator<<(std::ostream & os, const std::pair< unc, unc > & u);
```

Output a pair of (X and Y) values with (if defined) uncertainty and degrees of freedom.

For example: "1.23 +/- 0.01 (13), 3.45 +/- 0.06 (78)".

Global plusminus

`boost::svg::plusminus` — Nominal factor of 2 (strictly 1.96) corresponds to 95% confidence limit.

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

static const double plusminus;
```

Description

Number of standard deviations used for plusminus text display.

Function template operator<<

`boost::svg::operator<<`

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<bool correlated>
std::ostream & operator<<(std::ostream & os, const unc< correlated > & u);
```

Description

Note that the uncertainty is input and stored as one standard deviation, but output multiplied for a user configurable 'confidence factor' plusminus, default is two standard deviation for about 95% confidence (but could also be one for 67% or 3 for 99% confidence).

Output a single value with (if defined) uncertainty and degrees of freedom (and type). For example: "1.23 +/- 0.01 (13)".
 /details Note that the uncertainty is input and stored as one standard deviation, but output multiplied for a user configurable 'confidence factor' plusminus, default two for about 95% confidence (but could also be one for 67% or 3 for 99% confidence).

Note that the plus or minus can be output using several methods.

256 character 8-bit codepage plusminus symbol octal 361, or os << char(241) decimal 241 or os << char(0xF1) hexadecimal F1, or
 os << " ±" Unicode space plusminus glyph, or
 os << "+or-" << u.uncertainty_; Plain ANSI 7 bit code chars.

Function template operator<<

boost::svg::operator<<

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<bool correlated>
std::ostream &
operator<<(std::ostream & os,
const std::pair< unc< correlated >, unc< correlated > > & u);
```

Description

Output a pair (X and Y) value with (if defined) uncertainty and degrees of freedom.

For example: "1.23 +/- 0.01 (13), 3.45 +/- 0.06 (78)".

Function template unc_of

boost::svg::unc_of

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<typename T> float unc_of(T);
```

Description

Allow uncertainty (standard deviation) part of variables of class unc to be assigned to, and compared with float.

Template Parameters:

T Built-in floating-point type, float, double or long double, or uncertain type unc.

Returns:

zero always (because no uncertainty information is available for built-in double, float, or long double).

zero always (because no uncertainty information is available for built-in double, float, or long double).

Function template unc_of

boost::svg::unc_of

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<bool correlated> float unc_of(unc< correlated > v);
```

Description

Returns: unc.uncertainty() as a float. (Can be cast or converted to double without loss of accuracy).

Function template uncs_of

boost::svg::uncs_of — Get uncertainties (standard deviation) of a pair of values.

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<typename T> std::pair< float, float > uncs_of(T);
```

Description

Template Parameters: T Built-infloating-point type or unc.

Function template uncs_of

boost::svg::uncs_of — Get uncertainties (standard deviation) of a pair of values.

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<typename T> std::pair< float, float > uncs_of(std::pair< T, T > vp);
```

Description

Template Parameters: T Built-in floating-point type or unc.

Function template uncs_of

`boost::svg::uncs_of` — Get uncertainties (standard deviation) of a pair of values.

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<typename T>
std::pair< const float, float > uncs_of(std::pair< const T, T > vp);
```

Description

Template Parameters: T Built-in floating point type or unc.
Returns: uncertainty parts (if any) as a pair of floats.

Function template value_of

`boost::svg::value_of` — <

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<typename T> double value_of(T v);
```

Description

Allow value part of variables of class unc to be assigned to, and compared with double.

Template Parameters: T Built-in floating-point type, float, double or long double, or uncertain type unc.
Returns: value type convertible to double.
value as a double.
value as a double.

Function template value_of

`boost::svg::value_of`

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<> double value_of(unc< true > v);
```

Description

Returns: unc.value() as a double.

Function template value_of

boost::svg::value_of

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<> double value_of(unc< false > v);
```

Description

Returns: unc.value() as a double.

Function template values_of

boost::svg::values_of — Get double values of a pair of values.

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<typename T> std::pair< double, double > values_of(T);
```

Description

Template Parameters: T Built-in floating-point type, float, double, long double or unc.

Function template values_of

boost::svg::values_of

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<typename T>
std::pair< double, double > values_of(std::pair< const T, T > vp);
```

Description

<
Template Parameters: T Built-infloating-point type, float, double, long double or unc.
Returns: values of a pair of double values.

Function template values_of

boost::svg::values_of

Synopsis

```
// In header: <boost/svg_plot/uncertain.hpp>

template<bool correlated>
std::pair< double, double >
values_of(std::pair< unc< correlated >, unc< correlated > > up);
```

Description

Returns: value (part) as a pair of doubles.

Using Inkscape

These two programs can be regarded as *reference* programs for rendering SVG images. The results are always superior to any general purpose browsers.

Using Inkscape

You will need to use the Windows Explorer, Tools, Folder Options, and add extension .svg if necessary, and then associate .svg files with inkscape.exe.

You may also find it convenient to add "c:\program Files\inkscape" to your Path using Control Panel, System, Advanced system settings, Environment variables, Path, and edit to append the folder. You can then use commands like "inkscape my_picture.svg" from a command window.

Inkview allows one to open many SVG files as a slideshow and move forward and backward (and back to start) with the cursor keys.

This works as expected using a command window, for example after changing to the directory containing your SVG files, and running "inkscape *.svg" to display all the images in the directory as a slideshow.

Sadly, by default, if you select many files (but not more than 15!) in Windows Explorer, multiple windows are opened instead of a slideshow. If you know how to change this 'by design' feature, please tell me.

You can use this conveniently from your desktop (for example), by creating a shortcut to inkview.exe (right click in the inkscape install directory and choose Create Shortcut). Drag the shortcut to the desktop. Append "*.svg" to the Target, usually \c "C:\Program Files\Inkscape\inkscape.exe", or perhaps the name of a single file. If necessary, change Start in to the folder containing the svg files, for example, \c C:\Users\Paul\Desktop\My_Plots. You can add comment describing what it will do, and change the name of the shortcut file to something more helpful, for example, "View My Plots".

You may instead find it useful to create a batch file, for example called view_demo_plots.bat containing, for example:

```
START "Inkscape" /MIN inkscape.exe C:\Users\Paul\Desktop\My_Plots\*.svg
```

that will display all the svg images in folder \My_Plots in turn as the cursor keys are pressed. Of course, you can also omit the folder specification, and just have *.svg when the location of the batch file will determine which svg files are being viewed.

Using Inkscape to edit your plot, for example to add annotation

You can add text, and other graphics like lines and shape, using Inkscape.

It is probably best to save as "Plain SVG" to avoid additional Inkscape specific metadata that will increase file size a little.

Of course, if you recreate the SVG file from your C++ program, all these additions will be lost, but for presentations, using Inkscape may be most convenient.

You can also add text and graphics to the C++ program but then you cannot see what it will look like immediately as you can with Inkscape.

Using Inkscape to convert your plot, for example to Portable Network Graphic png

Inkscape allows you to save in several formats:

- Plain SVG - with a minimum of XML metadata.
- Inkscape SVG - more XML metadata.

and compressed (zipped) versions of these (producing tiny file but sadly not automatically unzipped by browsers, only by Inkscape).

- PostScript (.ps) and Encapsulated PostScript (.eps)

- PDF via Cairo (.pdf)
- Enhanced metafile (.emf)
- Open document drawing (.odg)
- LaTex with PSTricks macros (.text)

And you can Export (all or part) as a bitmap (.bmp).

And you can Import as many types including:

- BMP Bitmap
- JPEG Independent JPEG Group(.jpeg)
- TIFF Tagged Image File Format(.tif)
- Adobe Portable document Format (.pdf)
- PNG Portable Network Graphics (.png)

Of these, PNG is often most useful for creating Boost documentation as html, probably using Quickbook etc. For html, PNG is preferred because one common browser does not support SVG without an add-in.

(For generating PDF, at least using RenderX, the quality of graphics using PNG is poor and using SVG is very much preferred).

Using Inkscape to convert Scalable Vector graphic SVG files to Portable Network Graphic PNG

Unlike the GUI version, you don't get any feedback messages.

so you will be surprised/disappointed/confused that

```
>inkscape.exe -v
```

produces no output!

(But errorlevel is set non-zero if it fails).

How to Use Inkscape's Command Line Options on Win32

[inkscapec.exe - A Command Line Wrapper for Win32](#) In a nutshell it's a tiny console application, which spawns Inkscape (with the specified arguments) and captures its stdout (default output) and stderr (default error output) streams and redirects it to this console. It also does the same with stdin for good measure. Its sole purpose is direct command line interaction (and determining which actions trigger GTK warnings). If you want to spawn Inkscape from other programs/scripts use inkscape.exe instead. For example:

```
C:\Users\Paul>inkscapec.exe -V
Inkscape 0.46 (Apr 1 2008)
```

and

```
C:\Users\Paul>inkscapec.exe -?
```

```
Usage: inkscape.exe [OPTIONS...] [FILE...]
```

Available options:

-V, --version	Print the Inkscape version number (NOTE CAPITAL V).
-z, --without-gui	Do not use X server (only process files from console).
-g, --with-gui	Try to use X server (even if \$DISPLAY is not set).
-f, --file=FILENAME	Open specified document(s) (option string may be excluded).
-p, --print=FILENAME	Print document(s) to specified output file (use ' program' for pipe).
-e, --export-png=FILENAME	Export document to a PNG file.
-d, --export-dpi=DPI	The resolution used for exporting SVG into bitmap (default 90).
-a, --export-area=x0:y0:x1:y1	Exported area in SVG user units (default is the canvas; 0,0 is lower-left corner).
-D, --export-area-drawing	Exported area is the entire drawing (not canvas).
-C, --export-area-canvas	Exported area is the entire canvas.
--export-area-snap	Snap the bitmap export area outwards to the nearest integer values (in SVG □ user units).
-w, --export-width=WIDTH	The width of exported bitmap in pixels (overrides export-dpi).
-h, --export-height=HEIGHT	The height of exported bitmap in pixels (overrides export-dpi).
-i, --export-id=ID	The ID of the object to export.
-j, --export-id-only	Export just the object with export-id, hide all others (only with export-id).
-t, --export-use-hints	Use stored filename and DPI hints when exporting (only with export-id).
-b, --export-background=COLOR	Background color of exported bitmap (any SVG-supported color string).
-y, --export-background-opacity=VALUE	Background opacity of exported bitmap (either 0.0 to 1.0, or 1 to 255).
-l, --export-plain-svg=FILENAME	Export document to plain SVG file (no sodipodi or inkscape namespaces).
-P, --export-ps=FILENAME	Export document to a PS file.
-E, --export-eps=FILENAME	Export document to an EPS file.
-A, --export-pdf=FILENAME	Export document to a PDF file.
-M, --export-emf=FILENAME	Export document to an Enhanced Metafile (EMF) File.
-T, --export-text-to-path	Convert text object to paths on export (EPS).
-F, --export-embed-fonts	Embed fonts on export (Type 1 only) (EPS).
-B, --export-bbox-page	Export files with the bounding box set to the page size (EPS).
-X, --query-x	Query the X coordinate of the drawing or, if specified, of the object with --query-id
query-id	Query the Y coordinate of the drawing or, if specified, of the object with --query-y
id	Query the width of the drawing or, if specified, of the object with --query-id
-W, --query-width	Query the height of the drawing or, if specified, of the object with --query-id
-H, --query-height	id
-S, --query-all	List id,x,y,w,h for all objects.
-I, --query-id=ID	The ID of the object whose dimensions are queried.
-x, --extension-directory	Print out the extension directory and exit.
--vacuum-defs	Remove unused definitions from the defs section(s) of the document.
--verb-list	List the IDs of all the verbs in Inkscape.
--verb=VERB-ID	Verb to call when Inkscape opens.
--select=OBJECT-ID	Object ID to select when Inkscape opens.
Help options:	
-?, --help	Show this help message
--usage	Display brief usage message

For an interactive batch conversion using Windows at the command prompt in the svg source folder:

```
X:\svg\source\folder>
FOR %? IN (*.svg) DO "c:\program files\inkscape\inkscape.exe" -f X:\svg\source\folder\%? -e E:\png\tar\get\folder\%?.png

FOR \r %i IN (*.svg) DO echo %i
```

will list all .svg recursively in sub-folders.

A batch file (for example, called svg2png1.bat) that contains either the cryptic or verbose versions:

```
inkscape my_plot.svg -d 120 -e my_plot.png  
inkscape my_plot.svg --export-png=my_plot.png --export-dpi=120
```

converts my_plot.svg to my_plot.svg.png

A batch file (for example, called svg2png.bat) containing

```
FOR %%? IN (*.svg) DO inkscape.exe -f J:\cpp\svg%%? -e J:\cpp\svg%%?.png
```

converts all the .svg file in folder J:\cpp\svgto .svg.png

(The file type *.svg.png is useful in that it shows the fact that the original source is a SVG file).

```
FOR /r %%? IN (*.svg) DO inkscape.exe -f %%? -e %%?.png
```

does the same but recursing down sub-folders, leaving the converted .png in the same folder as the original .svg.

```
J:\Cpp\SVG>inkscape.exe -f J:\Cpp\SVG\sf_graphs\powm1.svg -e J:\Cpp\SVG\sf_graphs\powm1.svg.png
```

Derived from [Examples of using Inkscape command line](#)

Examples of using Inkscape Command line options

While obviously Inkscape is primarily intended as a GUI application, it can be used for doing SVG processing on the command line as well.

Several of these will be useful to control the dimensions and resolution (and thus size) of exported PNG images. In general, PNG files are rather larger, perhaps a few times, than the original SVG file, but might be smaller if low enough resolution, for example for an icon.

Open an SVG file in the GUI:

```
inkscape filename.svg
```

Print an SVG file from the command line:

```
inkscape filename.svg -p '| lpr'
```

Convert an SVG to PNG

```
inkscape -f file.svg -e file.png
```

Export an SVG file into PNG with the default resolution of 90dpi (one SVG user unit translates to one bitmap pixel):

```
inkscape filename.svg --export-png=filename.png
```

Same, but force the PNG file to be 600x400 pixels:

```
inkscape filename.svg --export-png=filename.png -w600 -h400
```

Same, but export the drawing (bounding box of all objects), not the page:

```
inkscape filename.svg --export-png=filename.png --export-area-drawing
```

Export to PNG the object with id="text1555", using the output filename and the resolution that were used for that object last time when it was exported from the GUI:

```
inkscape filename.svg --export-id=text1555 --export-use-hints
```

Same, but use the default 90 dpi resolution, specify the filename, and snap the exported area outwards to the nearest whole SVG user unit values (to preserve pixel-alignment of objects and thus minimize aliasing):

```
inkscape filename.svg --export-id=text1555 --export-png=text.png --export-snap-area
```

Convert an Inkscape SVG document to plain SVG:

```
inkscape filename1.svg --export-plain-svg=filename2.svg
```

Convert an SVG document to EPS, converting all texts to paths:

```
inkscape filename.svg --export-eps=filename.eps --export-text-to-path
```

Query the width of the object with id="text1555":

```
inkscape filename.svg --query-width --query-id text1555
```

Duplicate the object with id="path1555", rotate the duplicate 90 degrees, save SVG, and quit:

```
inkscape filename.svg --select=path1555 --verb>EditDuplicate --verb=ObjectRotate90 --verb=FileSave --verb=FileClose
```

Using Python to run Inkscape

See [How to Use Inkscape's Command Line Options with Python](#)

InkCL.py (place it in the same directory as inkscape.exe)

```
#!/opt/oss/bin/python"

import os, subprocess, sys

def spawn():
    cmd=sys.argv
    cmd.pop(0)
    cmd.insert(0,'inkscape')
    run=subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    out,err=[e.splitlines() for e in run.communicate()]
    return run.returncode, out, err

if __name__=='__main__':
    r=spawn()
    if not r[0]==0:
        print 'return code:',r[0]
    for l in r[1]:
        print l
    for l in r[2]:
        print l
InkCL.py usage:
python InkCL.py <inkscape options>
```

PNG export example:

```
C:\Inkscape-0.45.1-1\inkscape>python InkCL.py -e image.png ext.svg
Background RRGGBBAA: ffffff00
Area 0:0:64:64 exported to 64 x 64 pixels (90 dpi)
Bitmap saved as: image.png
```

If you don't have Python installed. Fear not! Inkscape is actually shipped with a Python interpreter, which is used by many extensions.

We can use this interpreter and also shorten the command line by using a batch file.

InkCL.bat (place it in the same directory as inkscape.exe)

```
@"./python/python.exe" InkCL.py %*
InkCL.bat usage:
InkCL.bat <inkscape options>
The following also works:
inkcl <inkscape options>
```

PNG export example:

```
C:\Inkscape-0.45.1-1\inkscape>inkcl -e image.png ext.svg
Background RRGGBBAA: ffffff00
Area 0:0:64:64 exported to 64 x 64 pixels (90 dpi)
Bitmap saved as: image.png
```

Download: [inkscape_win32_command_line.zip](#) (1kb - contains: InkCL.py, InkCL.bat, and a readme)

Implementation, History & Rationale

History

1. 2007 Initial versions without uncertainty handling by Jake Voytko.
2. 2009 Simple uncertainty handling added by Paul A. Bristow.
3. 2010 Release as a [sourceforge project](#).
4. 2012 Full uncertainty handling added by Paul A. Bristow.
5. 2013 Update documentation to use C++11 and GIT.

Compilers and Examples

Compiler and platforms versions

The code is header-only and does not require any pre-built libraries, but to avoid requiring the Boost C99 math library, you should define the macro BOOST_ALL_NO_LIB. For example, in the jamfile this is done with:

```
<define>BOOST_ALL_NO_LIB # Avoid using 'libboost_math_c99-vc100-mt-gd-1_53.lib'.
```

In a preprocessor definitions, write -DBOOST_ALL_NO_LIB

You will require a Boost library to be in your include path and the SVG plot modules

```
<boost/svg_plot/*.hpp>
```

The uncertainty handling code has been stored at

```
<boost/quan/*.hpp>
```

The code of version 2 is written to comply with the C++11 standard (and use some features).



Note

If you want to use older compilers, you should probably instead use the 1st release of this software which did not provide a full implementation of uncertainty, but this feature requires the C++11 auto feature.

The examples have been built with

1. Microsoft Visual Studio 2010 and 2012 (MSVC).
2. GCC 4.7.2 needs option -std=c++11
3. Clang 3.1 needs option -std=c++11

but might work on some earlier releases.

To avoid warnings:

with GCC and Clang these options may be useful

```
-Wno-missing-braces, -Wno-reorder, -Wno-unused-variable
```

and with MSVC

```
/wd4800 # Forcing value to bool 'true' or 'false'
/wd4996 # Deprecated.
/wd4512 # Assignment operator could not be generated.
/wd4127 # Expression is constant.
```

Other compilers like Apple Darwin and Intel have not been used, but are likely to work OK.

IDE

The examples have been built using two IDEs:

- Microsoft Visual Studio (using the Microsoft compilers only).
- Netbeans 7.2 (using Clang and GCC, and Microsoft compiler using VCC4N C++ Compiler tool).

Building the examples

A Boost b2 (aka bjam) jamfile.v2 is provided in the libs/examples folder to build all the examples using the chosen compiler. You may like to copy and modify this to build your own programs using b2. It contains the necessary options and warnings disablers for the above compilers.

Implementation and other notes

This section provide more information about this implementation and some of the rationale for design decisions.

Switches for axis labels and autoscaling

If a axis label string is provided, or autoscaling is requested, it is assumed that it should be displayed/acted on. So there functions have the effect of switching these options on, as if, for example:

```
x_autoscale(true); y_label_on(true);
```

This avoids users providing an axis label string and then wondering why it appears to have no effect.

It also avoids wasting plot space for empty labels.

It is still possible to switch these options off, for example with:

```
x_autoscale(false); y_label_on(false);
```

Obviously these must come after the function call that switches the option on.

Number of Minor ticks

`x_num_minor_ticks()` and `x_num_minor_ticks()`

are provided control the number of minor ticks between the major ticks.

There are a total of `x_num_minor_ticks + 1` ticks for each major tick, for example, a major on 0, minor on 1,2,3,4, major on 5 ...

For integer usually binary, octal and hexadecimal, `num_minor_ticks = 2[super]n - 1` are useful, for example 1 if major ticks are even will give minor ticks on odd values), 3 if major on 0, minor on 1, 2, 3 and major on 4... 7 if major on 0, minor on 1,2,3,4,5,6,7 and major on 8 ... 15 if major on 0, minor on 1,2,3,4,5,6,7, 8,9,A,B,C,D,E,F, major on 0x10...

For decimal based values, num_minor_ticks 1, 4, 9 are useful, for example:
 1 if major are even, major 0, minor 1, major 2 ...
 4 if major on 0, minor on 1,2,3,4, major on 5 ...
 9 if major on 0, minor on 1,2,3,4,5,6,7,8,9, major on 10

[/h4 Minor ticks]

SVG Specification

SVG version 1.1 was used in the design of this version but [SVG 1.2 draft specification](#) is also available and appears to be final and designed to be backward compatible. No changes that affect the code produced have been detected from a quick perusal of this document. It is probable but untested that the SVG files produced will also comply with the [Tiny SVG for mobiles](#) specification.

Design

SVG would be a flat format if it weren't for the <g> elements: these make it parse into a tree.

If each element has its own style (fill, stroke etc) then the .svg file size would be increased. (This style is used by other packages that output SVG and often leads to larger file sizes).

So the [group element](#) is used with each type given an id to allow reference back to it.

svg_style_details.hpp contains a list of these groups, for example: PLOT_X_AXIS, PLOT_TITLE... indexing an array of string document id "yAxis", "title"...

and these can be seen, with style information in the output, for example:

```
<g id="yAxis" stroke="rgb(0,0,0)"></g>
<g id="title">
<text x="250" y="20" text-anchor="middle" font-size="20">Demo 1D plot</text></g>
```

In the general case, the most that occurs grouped together is the style information: axis lines all share the same style information, points in a series all share the same color information, and this is also a logical grouping. One can add a series, and then come back later and change the <g> element above the points, which is a single change, and have the change reflected on every point in the series. It seems this is the most logical way to represent the data in memory.

Economising on SVG File Size

Some of the factors affecting the file size are:

1. Excessive precision of data points.

If plots are to be viewed only at a modest size, then the precision of x and y coordinates does not need to be higher than about 1 in 1000, so a precision of about 3 decimal digits will suffice.

A default of 3 decimal digits has been chosen, but the precision can be controlled to permit a higher resolution, for example on a map that is printed at 2000 dpi. or even only 2 decimal digits for a display on a mobile, where file transfer speed may make reducing file size important.

1. Excessive data points. When plotting functions, it is sensible to avoid using more data points than are justified by the resolution. Again 1 in 1000 is a typical number, and by using bezier curve fitting, as few as 100 points should be sufficient to produce a visibly perfectly smooth curve for 'well-behaved' functions.

Plotting extremely large datasets from files may cause memory overflow, as with any STL container held in memory. It may be necessary to perform some averaging or smoothing, or just using a 'stride' to select, for example, every 100th value to be plotted.

1. Redundant style specifications.

The use of the g or group element has been used to try to reduce repeated (and thus redundant) style specifications. More efficient use of groups may be possible.

1. Not using default style and attributes where possible.

Some effort has been made to use defaults, but more may be possible.

Adding document information to the svg Image files

Several ways of adding useful information to the svg document are provided. If the strings are null (the default) nothing is output.

Member function `svg& description(const std::string)` allows output like:

```
<desc>My Document description</desc>
svg& document_title(const std::string)
<title>My Document title</title>
```

(Note that this is **not** the same as the title of the plot.)

Adding Copyright and License conditions for the SVG Image files

In general, setting values for items like `copyright_holder` will ensure that they appear in the SVG document both as XML and as XML comment. If none are set, nothing will be output, to minimize file size.

www.w3.org/TR/2004/WD-SVG12-20041027 discusses primary documents, of type .svg in section 17.3.

Adding Copyright information to an SVG document: SVG encourages the use of a common metadata format for inclusion of copyright information. Metadata relevant to the data copyright of the entire document should be added to metadata element of the topmost `svg` element. This allows the author to unambiguously state the licensing terms for the entire document. The scheme may also be used elsewhere in the document, for pieces that have different licensing. For example, an SVG font may have specific licensing details expressed in its own metadata element.

Note that inclusion of this metadata does not provide the author with a method in which to protect or enforce their copyright, it simply bundles the copyright information with the content in a defined manner. Providing methods, technical or non-technical, for data protection is currently beyond the scope of the SVG specification.

This does not exclude the use of other metadata schemes.

A simple way of adding metadata for copyright is provided by functions to set

```
copyright_holder(std::string);
copyright_date(std::string);
```

(as well as `description` and `document_title`, and .svg filename if any).

For example, setting these will incorporate an XML comment

```
<!-- SVG Plot Copyright Paul A. Bristow 2013 -->
```

and also as meta data:

```
<meta name="copyright" content="Paul A. Bristow" />
<meta name="date" content="2014" />
```

The Creative Commons Metadata Set is also provided as an option in this implementation, following [this example](#).

The [Creative Commons License](#) is one method of providing license terms in a machine-readable format.

Typical data added to the file would be XML like this:

```
<metadata>
<rdf:RDF xmlns:cc="http://web.resource.org/cc/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<cc:work rdf:about="filename.svg">
    <!-- insert filename or title here -->
    <dc:title>Plot title</dc:title>
    <dc:creator>Boost.Plot</dc:creator>
    <dc:author>Paul A. Bristow</dc:author>
    <dc:format>application/xhtml+xml+svg</dc:format>
    <cc:license rdf:about="http://creativecommons.org/licenses/by-sa/.0/">
        <cc:requires rdf:resource="http://web.resource.org/cc/Notice"/>
        <cc:permits rdf:resource="http://web.resource.org/cc/Reproduction"/>
        <cc:permits rdf:resource="http://web.resource.org/cc/Distribution"/>
        <cc:requires rdf:resource="http://web.resource.org/cc/Attribution"/>
        <cc:permits rdf:resource="http://web.resource.org/cc/CommercialUse"/>
    </cc:license>
</cc:work>
</rdf:RDF>
</metadata>
```

cc:permits can also be cc:requires or cc:prohibits, as required.

A license that permits reproduction, distribution and commercial use, but requires both notice & attribution is probably most suitable for Boost documents as is therefore chosen as the default.

This license can be included by calling `svg` member function `is_license(true)`. If this license will be included can be discovered by calling `svg` member function `is_license()`.

Similarly functions `const std::string license_reproduction();` `const std::string license_distribution();` `const std::string license_attribution();` `const std::string license_commercialuse()` allow you to find the current license requirements.

[RDF](#) is the metadata format chosen by [Creative Commons](#).

and references the resource (cc) at <http://web.resource.org/cc/> using the definition of its XML namespace:

```
rdf:RDF xmlns:cc="http://web.resource.org/cc/"
```

Options are [summarized](#).

A way of setting the author is also provided, in case the copyright has been assigned to someone else, for example a publisher. `svg` set and get member functions:

```
author(const std::string);
const std::string author();
```

Using Unicode Symbols (usually Math Symbols and Greek letters)

Unicode symbols that work on most browsers in html are listed in some Quickbook files:

....doc/html4_symbols.gbkdoc/latin1_symbols.qbkdoc/math_symbols.qbk

[reference/html40/entities/symbols](#), and [demos](#).

However support for Unicode in SVG is much less fully implemented and displayed results are variable.

The Unicode value in decimal 9830 or hex x2666 must be prefixed with & and terminated with ; for example, &x2666; for xml and then enveloped with "" to convert to a const std::string, for example: "&x2666;" for diamond.

Thus diamond can be used a point marker.

Similarly for greek in title, legend and axes:

```
.legend_title("My Legend &#956;") // generates <em>&#956;</em> greek mu
```

Subscript and superscript in title, legend and labels

It is very common need to show superscript, in units for example, area (m[super 2]), and subscript for a[sub 0], a[sub 2]...

Sadly, although these needs are both OK with html, showing sub and superscripts in svg doesn't work well as yet because browsers don't handle <sub> or <sup>, nor baseline-shift.

The only reasonably widely supported feature is Unicode symbols for superscript 1, 2 and 3 (only). For example, Latin1_symbols: sup1 &185; sup2 &178; sup3 &179; work on Firefox & IE6/7.

Butࠗ ࠛ ࠪ - show just square boxes.

The SVG specification covers <sub>, <super>, and the general baseline-shift, but these are just not implemented yet.

2.8 Superscripts and Subscripts symbols

At Nov 2007, the following commands don't have very useful coverage. <tspan baseline-shift = "50%" font-stretch = "wider" font-variant = "small-caps" > 3 </tspan> baseline-shift = "50%" & super and subscript down't work on Firefox 2. but DO on IE6 (and probably 7) font-stretch = "wider" No effect Firefox or IE6. font-variant = "small-caps" No effect Firefox or IE6. <tspan baseline-shift = "50%" >3/4 </tspan>

This has been reported as a bug to Mozilla, and is regarded a duplicate of other reports on sub and super https://bugzilla.mozilla.org/show_bug.cgi?id=401102

but baseline-shift has no effect on Firefox 2. See <http://www.imagemagick.org/discourse-server/viewtopic.php?f=3&t=7410&view=previous>

The use of <tspan> to shift characters **is** feasible, as shown [in this example](#), which displays correctly in all browsers,

and it is also possible to use

```
<svg:g transform="translate(5.062500, -5.367188)">
```

to shift the next letter up, but these parameters are font-size related,* and the svg most verbose.

None of these method provides a **convenient** method of creating the right string for titles, legends, or axis labels, but it may be possible to devise code that does.

Coding style

In general [Boost coding style guide](#) and [Boost coding guidelines](#) has been used.

Some data members end with _ to avoid name clashes, another widely used convention.

To Do List

This project is still in development: here is a 'known wishes' list. Suggestions to pbrisstow at http://u-net.com.

- Allow **function pointers** to specify functions as input. A typical use case is to just see what a function like $\sin(x)$ looks like.
- **Radian coordinate system.**
- Implementation of **SVG's DOM**. This will make the SVG class useful for other things than solely making graphs! The existing SVG features implemented are now documented better, so one could consider using it for other purposes.
- **Automatic scaling** of the axis has been implemented. Feedback on how 'nicely' it works in practice would be welcome. Some of the uglier features of Microsoft Excel autoscaling have been avoided, but opinions on aesthetics vary widely.
- Allow **2D plots of 1D containers** - a very common need, for example for Time Series. Jake had trouble with having functors that update state interact with `make_transform_iterator`, and omitted the feature.
- Make time and/or data labelled axis possible, with datetime a first class citizen type.
- Allow **other image formats**. There are many inherent difficulties with this part. The solution that was explored is allowing the user to pass a functor that traverses the document tree. Generalizing images to an `image` class is fraught with difficulties, as SVG is a tree-based format, which it does not share with many other formats.
- (Note that [Inkscape](#) permits conversion to bitmap format, .png etc, and other tools can accept and convert bitmaps. This is used in the Boost.Math documentation. A Python file called generate.sh in the math toolkit shows how to automate this conversion process for many .svg files in a folder using Inkscape in command line mode. `$inkscape -d $dpi -e $(cygpath -a -w $pngfile) $(cygpath -a -w $svgfile)`)
- **Avoid redrawing the entire plot each time.** This is the easiest way to write the program initially, but it would be more efficient if the program could keep track of what has been changed and what hasn't, so that it may be more efficient if lots of images are being produced. This hasn't been found to be a problem in practice so far.
- Allow the user to provide a function object for generating **custom axis labels**, for example, label axes with names of months (Jan, Feb, Mar...) instead of integer representations of months (1, 2, 3...).
- **Logarithmic Axes.** This does not look simple. Meanwhile you can take the log of the data.
- Allow an **External stylesheet** to be loaded to style the graph. External stylesheets will allow a standard and easy way to style the document so that users don't have to come up with their own home-grown solutions. This was sketched out, but seemed much more complicated now that very many more options to control features have been added. (The 'first try' code that Jake produced now longer compiles, so has been commented out).

Acknowledgements

Jake Voytko would like to thank the following people:

- **Google**: For offering Summer of Code, and giving me the opportunity to do something I never would have done otherwise.
- **Joaquín M^a López Muñoz**: My GSoC mentor. His proofreading and advice helped shape the project into what it is today, and prevented small problems from becoming major problems.
- **Paul A. Bristow**: For showing an active interest, offering literally dozens of minor and major features he'd like to see, and for helping with Boost.Build.
- **Matias Capeletto**: For showing an active interest, and offering feature suggestions.
- **Sarah Braun**: For helping me pick colors for examples.
- **Boost Community**: For all of the encouragement, suggestions, disagreements, and patience.

Paul A. Bristow would like to thank

- Jake Voytko for setting up the plot package during his Google Summer of Code period in 2007. I hope he will approve of its current state.
- John Maddock for much assistance with problems, and especially with documentation and development of indexing and pdf production.
- Joel de Guzman and Eric Niebler for producing the Quickbook and Doxygen indexing system.
- Daniel James for enhancements to Doxygen with Docbook.
- And of course all the bits of Boost that proved invaluable.

Class Index

Symbols

A

axis_line_style
 Class axis_line_style, 323
a_path
 Struct a_path, 110

B

bar_style
 Class bar_style, 325
box_style
 Class box_style, 327

C

circle_element
 Class circle_element, 112
clip_path_element
 Class clip_path_element, 114
close_to
 Class template close_to, 105
c_path

Struct c_path, 111

E

ellipse_element

 Class ellipse_element, 116

G

g_element

 Class g_element, 119

H

histogram_style

 Class histogram_style, 329

h_path

 Struct h_path, 124

L

line_element

 Class line_element, 126

l_path

 Struct l_path, 125

M

m_path

 Struct m_path, 128

P

path_element

 Class path_element, 129

path_point

 Struct a_path, 110

 Struct c_path, 111

 Struct h_path, 124

 Struct l_path, 125

 Struct m_path, 128

 Struct path_point, 134

 Struct q_path, 142

 Struct s_path, 149

 Struct t_path, 152

 Struct v_path, 164

 Struct z_path, 165

plot_line_style

 Class plot_line_style, 330

plot_point_style

 Class plot_point_style, 332

polygon_element

 Struct polygon_element, 136

polyline_element

 Class polyline_element, 139

poly_path_point

 Struct poly_path_point, 135

Q

curve_element

 Class curve_element, 143

q_path

Struct q_path, 142

R

rect_element

 Class rect_element, 146

S

smallest

 Class template smallest, 106

svg

 Class circle_element, 112

 Class clip_path_element, 114

 Class ellipse_element, 116

 Class g_element, 119

 Class line_element, 126

 Class path_element, 129

 Class polyline_element, 139

 Class curve_element, 143

 Class rect_element, 146

 Class svg, 172

 Class text_element, 153

 Class text_element_text, 157

 Class tspan_element, 159

 Struct a_path, 110

 Struct c_path, 111

 Struct h_path, 124

 Struct l_path, 125

 Struct m_path, 128

 Struct polygon_element, 136

 Struct q_path, 142

 Struct s_path, 149

 Struct t_path, 152

 Struct v_path, 164

 Struct z_path, 165

svg_1d_plot

 Class svg_1d_plot, 179

svg_1d_plot_series

 Class svg_1d_plot_series, 212

svg_2d_plot

 Class svg_2d_plot, 216

svg_2d_plot_series

 Class svg_2d_plot_series, 263

svg_boxplot

 Class svg_boxplot, 268

svg_boxplot_series

 Class svg_boxplot_series, 309

svg_element

 Class circle_element, 112

 Class clip_path_element, 114

 Class ellipse_element, 116

 Class g_element, 119

 Class line_element, 126

 Class path_element, 129

 Class polyline_element, 139

 Class curve_element, 143

 Class rect_element, 146

 Class svg_element, 150

Class text_element, 153
Class tspan_element, 159
Struct polygon_element, 136
svg_style
 Class svg_style, 334
s_path
 Struct s_path, 149

T

text_element
 Class text_element, 153
text_element_text
 Class text_element_text, 157
text_parent
 Class text_element_text, 157
 Class text_parent, 158
 Class tspan_element, 159
text_style
 Class text_style, 337
ticks_labels_style
 Class ticks_labels_style, 340
tspan_element
 Class tspan_element, 159
t_path
 Struct t_path, 152

U

unc
 Class template unc, 348

V

value_style
 Class value_style, 342
v_path
 Struct v_path, 164

Z

z_path
 Struct z_path, 165

Function Index

Symbols

A

add_g_element
 Class g_element, 119, 120
 Class svg, 172, 173
alignment
 Class text_element, 153, 154
area_fill
 Class bar_style, 325, 326
 Class plot_line_style, 330
 Class svg_2d_plot_series, 263, 264
attribution

Class `svg`, 172, 173
author
 Class `svg`, 172, 173
 Implementation and other notes, 363
autoscale
 1-D Autoscaling Various Containers Examples, 40
 Class `svg_1d_plot`, 179, 184
 Class `svg_2d_plot`, 216, 223, 224
 Class `svg_boxplot`, 268, 274
 Demonstration of using 2D data that includes information about its uncertainty, 76
autoscale_check_limits
 1-D Auto scaling Examples, 31
 Class `svg_1d_plot`, 179, 184, 185
 Class `svg_2d_plot`, 216, 224
 Class `svg_boxplot`, 268, 274
 Demonstration of using 2D data that includes information about its uncertainty, 76
autoscale_plusminus
 Class `svg_1d_plot`, 179, 185
 Class `svg_2d_plot`, 216, 224
 Class `svg_boxplot`, 268, 275
 Demonstration of using 2D data that includes information about its uncertainty, 76
axes_on
 Class `svg_1d_plot`, 179, 185
 Class `svg_2d_plot`, 216, 224
 Class `svg_boxplot`, 268, 275
axis_color
 Class `svg_boxplot`, 268, 275
 Class `svg_boxplot_series`, 309, 310
axis_width
 Class `svg_boxplot`, 268, 275
 Class `svg_boxplot_series`, 309, 310, 311

B

background_border_color
 Class `svg_1d_plot`, 179, 185
 Class `svg_2d_plot`, 216, 224
 Class `svg_boxplot`, 268, 275
background_border_width
 Class `svg_1d_plot`, 179, 185
 Class `svg_2d_plot`, 216, 224
 Class `svg_boxplot`, 268, 275, 276
background_color
 Class `svg_1d_plot`, 179, 185
 Class `svg_2d_plot`, 216, 224
 Class `svg_boxplot`, 268, 276
bar_area_fill
 Class `svg_2d_plot_series`, 263, 264
bar_color
 Class `svg_2d_plot_series`, 263, 264
bar_opt
 Class `bar_style`, 325, 326
 Class `svg_2d_plot_series`, 263, 264, 265
bar_width
 Class `svg_2d_plot_series`, 263, 265
bezier_on
 Class `plot_line_style`, 330, 331
 Class `svg_1d_plot_series`, 212, 213

Class `svg_2d_plot_series`, 263, 265
`boost_license_on`
 Class `svg`, 172, 173, 174
 Class `svg_1d_plot`, 179, 185
 Class `svg_2d_plot`, 216, 225
 Class `svg_boxplot`, 268, 276
`border_on`
 Class `box_style`, 327
`box_border`
 Class `svg_boxplot`, 268, 276
 Class `svg_boxplot_series`, 309, 311
`box_fill`
 Class `svg_boxplot`, 268, 276
 Class `svg_boxplot_series`, 309, 311
`box_style`
 Class `box_style`, 327
 Class `svg_boxplot_series`, 309, 311
`box_width`
 Class `svg_boxplot`, 268, 276
 Class `svg_boxplot_series`, 309, 311

C

`c`
 Class `path_element`, 129, 130
`C`
 Class `path_element`, 129, 130
`calculate_quantiles`
 Class `svg_boxplot_series`, 309, 311
`circle`
 Class `g_element`, 119, 120
 Class `svg`, 172, 174
`class_id`
 Class `circle_element`, 112, 113
 Class `clip_path_element`, 114, 115
 Class `ellipse_element`, 116, 117
 Class `g_element`, 119, 120
 Class `line_element`, 126, 127
 Class `path_element`, 129, 130
 Class `polyline_element`, 139, 140
 Class `curve_element`, 143, 144
 Class `rect_element`, 146, 147
 Class `svg_element`, 150, 151
 Class `text_element`, 153, 154
 Class `tspan_element`, 159, 160
 Struct `polygon_element`, 136, 137
`clear`
 Class `g_element`, 119, 120
`clip_id`
 Class `circle_element`, 112, 113
 Class `clip_path_element`, 114, 115
 Class `ellipse_element`, 116, 117
 Class `g_element`, 119, 120, 121
 Class `line_element`, 126, 127
 Class `path_element`, 129, 130
 Class `polyline_element`, 139, 140
 Class `curve_element`, 143, 144
 Class `rect_element`, 146, 147

Class `svg_element`, 150, 151
Class `text_element`, 153, 155
Class `tspan_element`, 159, 160
Struct `polygon_element`, 136, 137
`clip_path`
 Class `svg`, 172, 174
`color`
 Class `axis_line_style`, 323, 324
 Class `bar_style`, 325, 326
 Class `plot_line_style`, 330, 331
 Class `svg_2d_plot_series`, 263, 264, 265, 266
 Class `svg_color`, 316, 317
 Class `svg_style`, 334, 335, 336
`commercialuse`
 Class `svg`, 172, 174
`confidence`
 Class `svg_1d_plot`, 179, 186
 Class `svg_2d_plot`, 216, 225
 Class `svg_boxplot`, 268, 276, 277
`constant_to_rgb`
 Function `constant_to_rgb`, 319
 Header <`boost/svg_plot/svg_color.hpp`>, 315
`coord_precision`
 Class `svg`, 172, 174
 Class `svg_1d_plot`, 179, 186, 197
 Class `svg_2d_plot`, 216, 225
 Class `svg_boxplot`, 268, 277
`copyright_date`
 Class `svg`, 172, 174
 Class `svg_1d_plot`, 179, 186
 Class `svg_2d_plot`, 216, 225
 Class `svg_boxplot`, 268, 277
`copyright_holder`
 Class `svg`, 172, 174
 Class `svg_1d_plot`, 179, 186
 Class `svg_2d_plot`, 216, 225
 Class `svg_boxplot`, 268, 277

D

`data_lines_width`
 Class `svg_1d_plot`, 179, 186
 Class `svg_2d_plot`, 216, 225
 Class `svg_boxplot`, 268, 277
`default_font`
 Function `default_font`, 344
 Header <`boost/svg_plot/svg_style.hpp`>, 321
`default_plot_point_style`
 Header <`boost/svg_plot/svg_style.hpp`>, 321
`deg_free`
 Class template `unc`, 348, 349
`derived`
 Class `svg_1d_plot`, 179, 186, 187
 Class `svg_2d_plot`, 216, 226
 Class `svg_boxplot`, 268, 277
`description`
 Class `svg`, 172, 175
 Class `svg_1d_plot`, 179, 187

Class `svg_2d_plot`, 216, 226
Class `svg_boxplot`, 268, 278
Implementation and other notes, 363
distribution
 Class `svg`, 172, 175
document_size
 Class `svg`, 172, 175
document_title
 Class `svg`, 172, 175
Class `svg_1d_plot`, 179, 187
Class `svg_2d_plot`, 216, 226
Class `svg_boxplot`, 268, 278
Implementation and other notes, 363
double
 Class `svg_1d_plot`, 193
draw_line
 Class `svg_1d_plot`, 179, 187
 Class `svg_2d_plot`, 216, 226
 Class `svg_boxplot`, 268, 278
draw_note
 Class `svg_1d_plot`, 179, 187
 Class `svg_2d_plot`, 216, 226
 Class `svg_boxplot`, 268, 278
draw_plot_curve
 Class `svg_1d_plot`, 179, 187
 Class `svg_2d_plot`, 216, 226
 Class `svg_boxplot`, 268, 278
draw_plot_line
 Class `svg_1d_plot`, 179, 187
 Class `svg_2d_plot`, 216, 227
 Class `svg_boxplot`, 268, 278
draw_plot_point_values
 Class `value_style`, 342
dx
 Class `tspan_element`, 159, 160
dy
 Class `tspan_element`, 159, 160

E

ellipse
 Class `g_element`, 119, 121
 Class `svg`, 172, 175
epsilon
 Header < boost/svg_plot/detail/fp_compare.hpp >, 104
extreme_outlier_color
 Class `svg_boxplot`, 268, 279
 Class `svg_boxplot_series`, 309, 312
extreme_outlier_fill
 Class `svg_boxplot`, 268, 279
 Class `svg_boxplot_series`, 309, 312
extreme_outlier_shape
 Class `svg_boxplot`, 268, 279
 Class `svg_boxplot_series`, 309, 312
extreme_outlier_size
 Class `svg_boxplot`, 268, 279
 Class `svg_boxplot_series`, 309, 312
extreme_outlier_values_on

Class `svg_boxplot`, 268, 279

F

f

Simple Example, 84

Tutorial: 1D More Layout Examples, 20

Tutorial: Fuller Layout Example, 60

file

Class `svg_2d_plot`, 236

Using Inkscape, 356

fill

Class `box_style`, 327, 328

Class `svg_boxplot`, 276, 279, 280, 281, 283, 284, 285, 288, 290

Class `svg_boxplot_series`, 311, 312, 314

fill_color

Class `plot_point_style`, 332, 333

Class `svg_1d_plot_series`, 212, 213

Class `svg_2d_plot_series`, 263, 265

Class `svg_style`, 334, 335

Class `tspan_element`, 159, 161

fill_on

Class `box_style`, 327, 328

Class `path_element`, 129, 130, 131

Class `svg_style`, 334, 335

Class `tspan_element`, 159, 161

font

Class `svg_1d_plot`, 189, 190, 196, 197, 201, 208, 209, 210

Class `svg_2d_plot`, 228, 229, 234, 235, 239, 240, 246, 247, 248, 253, 254, 260, 262

Class `svg_boxplot`, 281, 282, 289, 290, 294, 295, 302, 303, 304, 307

Fonts, 8

font_decoration

Class `text_style`, 337, 338

font_family

Class `text_style`, 337, 338

Class `tspan_element`, 159, 161

font_size

Class `text_style`, 337, 338

Class `tspan_element`, 159, 161

font_stretch

Class `text_style`, 337, 338

font_style

Class `text_style`, 337, 338

Class `tspan_element`, 159, 161

font_weight

Class `text_style`, 337, 339

Class `tspan_element`, 159, 162

fpt_abs

Function template `fpt_abs`, 107

Header <`boost/svg_plot/detail/fp_compare.hpp`>, 104

G

g

Class `g_element`, 119, 121

Class `svg`, 172, 175

Simple Example, 84

Tutorial: 1D More Layout Examples, 20

Tutorial: Fuller Layout Example, 60

generate_text
 Class text_element, 153, 154

H

h
 Class path_element, 129, 131
Tutorial: 1D More Layout Examples, 20
Tutorial: Fuller Layout Example, 60
H
 Class path_element, 129, 131
hexagon
 Class g_element, 119, 121
 Class svg, 172, 175
histogram
 Class histogram_style, 328, 329
 Class svg_2d_plot_series, 263, 265

I

id
 Class circle_element, 112, 113
 Class clip_path_element, 114, 115
 Class ellipse_element, 116, 117
 Class g_element, 119, 121
 Class line_element, 126, 127
 Class path_element, 129, 131
 Class polyline_element, 139, 140
 Class quurve_element, 143, 144
 Class rect_element, 146, 147
 Class svg_element, 150, 151
 Class text_element, 153, 155
 Class tspan_element, 159, 162
 Struct polygon_element, 136, 137, 138
image_border_margin
 Class svg_1d_plot, 179, 188
 Class svg_2d_plot, 216, 227
 Class svg_boxplot, 268, 279, 280
image_border_width
 1-D Auto scaling Examples, 31
 Class svg_1d_plot, 179, 188
 Class svg_2d_plot, 216, 227
 Class svg_boxplot, 268, 280
image_filename
 Class svg, 172, 175
image_x_size
 Class svg_1d_plot, 179, 188
 Class svg_2d_plot, 216, 227
 Class svg_boxplot, 268, 280
image_y_size
 Class svg_1d_plot, 179, 188
 Class svg_2d_plot, 216, 227
 Class svg_boxplot, 268, 280
info
 Class polyline_element, 141
 Function operator<<, 167
is
 Class svg, 177
isfinite

Function template `isfinite`, 95
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 94
`isnan`
 Showing data values at Numerical Limits, 92
`is_blank`
 Class `svg_color`, 316, 317
 Function `is_blank`, 319
 Header <`boost/svg_plot/svg_color.hpp`>, 315
`is_license`
 Implementation and other notes, 363

L

`l`
 Class `path_element`, 129, 131

`L`
 Class `path_element`, 129, 131

`label_length`
 Class `ticks_labels_style`, 340, 341

`label_on`
 Class `axis_line_style`, 323, 324

`label_units_on`
 Class `axis_line_style`, 323, 324

`legend_background_color`
 Class `svg_1d_plot`, 179, 188
 Class `svg_2d_plot`, 216, 227
 Class `svg_boxplot`, 268, 280

`legend_border_color`
 Class `svg_1d_plot`, 179, 188
 Class `svg_2d_plot`, 216, 228
 Class `svg_boxplot`, 268, 280

`legend_box_fill_on`
 Class `svg_1d_plot`, 179, 189
 Class `svg_2d_plot`, 216, 228
 Class `svg_boxplot`, 268, 281

`legend_color`
 Class `svg_1d_plot`, 179, 189
 Class `svg_2d_plot`, 216, 228
 Class `svg_boxplot`, 268, 281

`legend_font_family`
 Class `svg_1d_plot`, 179, 189
 Class `svg_2d_plot`, 216, 228
 Class `svg_boxplot`, 268, 281

`legend_font_weight`
 Class `svg_1d_plot`, 179, 189
 Class `svg_2d_plot`, 216, 228
 Class `svg_boxplot`, 268, 281

`legend_header_font_size`
 Class `svg_1d_plot`, 179, 189
 Class `svg_2d_plot`, 216, 228
 Class `svg_boxplot`, 268, 281

`legend_lines`
 Class `svg_1d_plot`, 179, 189
 Class `svg_2d_plot`, 216, 228, 229
 Class `svg_boxplot`, 268, 281

`legend_on`
 Class `svg_1d_plot`, 179, 190
 Class `svg_2d_plot`, 216, 229

Class `svg_boxplot`, 268, 281, 282
legend_outside
 Class `svg_1d_plot`, 179, 190
 Class `svg_2d_plot`, 216, 229
 Class `svg_boxplot`, 268, 282
legend_place
 Class `svg_1d_plot`, 179, 190
 Class `svg_2d_plot`, 216, 229
 Class `svg_boxplot`, 268, 282
legend_title
 Class `svg_1d_plot`, 179, 190
 Class `svg_2d_plot`, 216, 229
 Class `svg_boxplot`, 268, 282
legend_title_font_size
 Class `svg_1d_plot`, 179, 190
 Class `svg_2d_plot`, 216, 229
 Class `svg_boxplot`, 268, 282
legend_top_left
 Class `svg_1d_plot`, 179, 190
 Class `svg_2d_plot`, 216, 229, 230
 Class `svg_boxplot`, 268, 282
legend_width
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 282, 283
license
 Class `svg`, 172, 176
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 283
license_attribution
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 283
 Implementation and other notes, 363
license_commercialuse
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 283
 Implementation and other notes, 363
license_distribution
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 283
 Implementation and other notes, 363
license_on
 Class `svg`, 172, 176
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 283
license_reproduction
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 283
 Implementation and other notes, 363
limits_count
 Class `svg_2d_plot_series`, 263, 265
limit_color

Class `svg_1d_plot`, 179, 191
Class `svg_2d_plot`, 216, 230, 231
Class `svg_boxplot`, 268, 283
`limit_fill_color`
 Class `svg_1d_plot`, 179, 191, 192
 Class `svg_2d_plot`, 216, 231
 Class `svg_boxplot`, 268, 283
`limit_point_color`
 Class `svg_1d_plot_series`, 212, 213
`line`
 Class `axis_line_style`, 323
 Class `bar_style`, 325
 Class `box_style`, 327
 Class `g_element`, 119, 121
 Class `plot_line_style`, 330, 331
 Class `svg`, 172, 176
 Class `svg_1d_plot`, 203
 Class `svg_2d_plot`, 256
 Class `svg_2d_plot_series`, 264, 265
 Class `svg_boxplot`, 292, 293, 297, 306
 Class `value_style`, 342
 Demonstration of adding lines and curves, typically a least squares fit, 80
`line_color`
 Class `svg_1d_plot_series`, 212, 213
 Class `svg_2d_plot_series`, 263, 265
`line_on`
 Class `plot_line_style`, 330, 331
 Class `svg_1d_plot_series`, 212, 214
 Class `svg_2d_plot_series`, 263, 266
`line_style`
 Class `svg_2d_plot_series`, 263, 266
`line_width`
 Class `svg_1d_plot_series`, 212, 214
 Class `svg_2d_plot_series`, 263, 266
`longest_label`
 Class `ticks_labels_style`, 340, 341
`l_or_r`
 Header <boost/svg_plot/show_1d_settings.hpp>, 169

M

`m`
 Class `path_element`, 129, 131
`M`
 Class `path_element`, 129, 132
`main`
 Fonts, 8
 Real-life Heat flow data, 48
 Simple Code Example, 58
 Simple Example, 84
 Tutorial: 1D Gridlines & Axes - more examples, 22
 Tutorial: 1D More Layout Examples, 20
 Tutorial: Fuller Layout Example, 60
`major_value_labels_side`
 Class `ticks_labels_style`, 340, 341
`margin`
 Class `box_style`, 327, 328
`max_value`

Header < boost/svg_plot/detail/fp_compare.hpp >, 104
max_whisker_color
 Class `svg_boxplot_series`, 309, 312, 313
max_whisker_width
 Class `svg_boxplot_series`, 309, 313
median
 Definitions of the Quartiles, 87
 Function `median`, 168
 Header < boost/svg_plot/quantile.hpp >, 168
median_color
 Class `svg_boxplot`, 268, 284
 Class `svg_boxplot_series`, 309, 313
median_style
 Class `svg_boxplot_series`, 309, 313
median_values_on
 Class `svg_boxplot`, 268, 284
median_width
 Class `svg_boxplot`, 268, 284
 Class `svg_boxplot_series`, 309, 313
min_value
 Header < boost/svg_plot/detail/fp_compare.hpp >, 104
min_whisker_color
 Class `svg_boxplot_series`, 309, 313
min_whisker_width
 Class `svg_boxplot_series`, 309, 314
my_blank
 Class `svg_color`, 317
my_color
 Function `operator<<`, 320

O

one_sd_color
 Class `svg_1d_plot`, 179, 192
 Class `svg_2d_plot`, 216, 231
 Class `svg_boxplot`, 268, 284
outFmtFlags
 Function `outFmtFlags`, 170
 Header < boost/svg_plot/show_1d_settings.hpp >, 169
outlier_color
 Class `svg_boxplot`, 268, 284
 Class `svg_boxplot_series`, 309, 314
outlier_fill
 Class `svg_boxplot`, 268, 284, 285
 Class `svg_boxplot_series`, 309, 314
outlier_shape
 Class `svg_boxplot`, 268, 285
 Class `svg_boxplot_series`, 309, 314
outlier_size
 Class `svg_boxplot`, 268, 285
 Class `svg_boxplot_series`, 309, 314
outlier_style
 Class `svg_boxplot`, 268, 285
 Class `svg_boxplot_series`, 309, 315
outlier_values_on
 Class `svg_boxplot`, 268, 285

P

P

- Class polyline_element, 139, 141
- Struct polygon_element, 136, 138

p

- Function operator<<, 167
- Struct polygon_element, 138

p0

- Function operator<<, 167

path

- Class g_element, 119, 122
- Class svg, 172, 176

pentagon

- Class g_element, 119, 122
- Class svg, 172, 176

plot

- Class svg_1d_plot, 179, 192, 193, 195
- Class svg_2d_plot, 216, 231, 232, 234
- Class svg_boxplot, 268, 285, 286
- Colors, 6
- Function template scale_axis, 100
- Showing data values at Numerical Limits, 92
- Simple Code Example, 58
- Tutorial: 2D Special Features, 62

plot_background_color

- Class svg_1d_plot, 179, 193, 194
- Class svg_2d_plot, 216, 232
- Class svg_boxplot, 268, 286

plot_border_color

- Class svg_1d_plot, 179, 194
- Class svg_2d_plot, 216, 232
- Class svg_boxplot, 268, 286

plot_border_width

- Class svg_1d_plot, 179, 194
- Class svg_2d_plot, 216, 232
- Class svg_boxplot, 268, 287

plot_window_on

- Class svg_1d_plot, 179, 194
- Class svg_2d_plot, 216, 232, 233
- Class svg_boxplot, 268, 287

plot_window_x

- Class svg_1d_plot, 179, 194
- Class svg_2d_plot, 216, 233
- Class svg_boxplot, 268, 287

plot_window_x_left

- Class svg_1d_plot, 179, 194
- Class svg_2d_plot, 216, 233
- Class svg_boxplot, 268, 287

plot_window_x_right

- Class svg_1d_plot, 179, 194
- Class svg_2d_plot, 216, 233
- Class svg_boxplot, 268, 287

plot_window_y

- Class svg_1d_plot, 179, 194, 195
- Class svg_2d_plot, 216, 233
- Class svg_boxplot, 268, 287

plot_window_y_bottom

Class `svg_1d_plot`, 179, 195
 Class `svg_2d_plot`, 216, 233
 Class `svg_boxplot`, 268, 287

`plot_window_y_top`
 Class `svg_1d_plot`, 179, 195
 Class `svg_2d_plot`, 216, 233
 Class `svg_boxplot`, 268, 287

`polygon`
 Class `g_element`, 119, 122
 Class `svg`, 172, 176
 Struct `polygon_element`, 136

`polyline`
 Class `g_element`, 119, 122
 Class `svg`, 172, 176, 177

`position`
 Class `axis_line_style`, 323, 324
 Class `svg_1d_plot`, 199
 Class `svg_2d_plot`, 237

`prefix`
 Class `svg_2d_plot`, 258

`push_back`
 Class `g_element`, 119, 122

Q

`q`
 Class `path_element`, 129, 132

`Q`
 Class `path_element`, 129, 132

`quantile`
 Definitions of the Quartiles, 86
 Function `quantile`, 169
 Header <`boost/svg_plot/quantile.hpp`>, 168

`quartile_definition`
 Class `svg_boxplot`, 268, 288
 Class `svg_boxplot_series`, 309, 315

R

`r`
 Function operator<<, 166

`rect`
 Class `g_element`, 119, 122
 Class `svg`, 172, 177

`reproduction`
 Class `svg`, 172, 177

`rhombus`
 Class `g_element`, 119, 123
 Class `svg`, 172, 177

`rotation`
 Class `svg_1d_plot`, 210, 211
 Class `svg_2d_plot`, 249
 Class `svg_boxplot`, 305
 Class `text_element`, 153, 155
 Class `tspan_element`, 159, 162

`rounddown10`
 Function `rounddown10`, 96
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 94

`rounddown2`

Function rounddown2, 96
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
rounddown5
Function rounddown5, 97
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
roundup10
Function roundup10, 97
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
roundup2
Function roundup2, 97
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
roundup5
Function roundup5, 98
Header < boost/svg_plot/detail/auto_axes.hpp >, 94

S

s
2-D Autoscaling Examples, 73
Class path_element, 129, 132
S
Class path_element, 129, 132
safe_fpt_division
Function template safe_fpt_division, 107
Header < boost/svg_plot/detail/fp_compare.hpp >, 104
scale_axis
Function scale_axis, 98, 99
Function template scale_axis, 99, 100, 101
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
series_count
Class svg_1d_plot_series, 212, 214
series_limits_count
Class svg_1d_plot_series, 212, 214
set_ids
Header < boost/svg_plot/detail/svg_boxplot_detail.hpp >, 108
shape
Class plot_point_style, 332, 333
Class svg, 176
Class svg_1d_plot_series, 212, 214
Class svg_2d_plot_series, 263, 266
show
Function template show, 102
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
show_1d_plot_settings
1-D Auto scaling Examples, 31
1-D Data Values Examples, 43
Function show_1d_plot_settings, 170
Header < boost/svg_plot/show_1d_settings.hpp >, 169
Tutorial: 2D Special Features, 62
show_2d_plot_settings
2-D Data Values Examples, 67
Function show_2d_plot_settings, 171
Header < boost/svg_plot/show_2d_settings.hpp >, 170
show_all
Function template show_all, 102
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
sin
Simple Example, 84

To Do List, 368

size
 Class `g_element`, 119, 123
 Class `plot_point_style`, 332, 333
 Class `svg`, 172, 177
 Class `svg_1d_plot`, 179, 189, 195, 201, 207, 208, 209, 212
 Class `svg_1d_plot_series`, 212, 214
 Class `svg_2d_plot`, 216, 228, 233, 239, 240, 245, 247, 259
 Class `svg_2d_plot_series`, 263, 266
 Class `svg_boxplot`, 268, 281, 288, 295, 301, 303, 308
 Class template `close_to`, 105
 Class template `smallest`, 106
 Class `text_style`, 337, 338
 Using Inkscape, 356

sqrt
 Tutorial: 1D More Layout Examples, 20
 Tutorial: Fuller Layout Example, 60

strength
 Class template `close_to`, 105

string_svg_length
 Function `string_svg_length`, 346
 Header <`boost/svg_plot/svg_style.hpp`>, 321

strip_e0s
 Function `strip_e0s`, 346
 Header <`boost/svg_plot/svg_style.hpp`>, 321

stroke
 Class `box_style`, 327, 328

stroke_color
 Class `plot_point_style`, 332, 333
 Class `svg_1d_plot_series`, 212, 214, 215
 Class `svg_2d_plot_series`, 263, 266
 Class `svg_style`, 334, 335

stroke_on
 Class `svg_style`, 334, 335

stroke_width
 Class `svg_style`, 334, 335, 336

style
 Class `circle_element`, 112, 113
 Class `clip_path_element`, 114, 115
 Class `ellipse_element`, 116, 117, 118
 Class `g_element`, 119, 123
 Class `line_element`, 126, 127
 Class `path_element`, 129, 132
 Class `plot_point_style`, 332, 333
 Class `polyline_element`, 139, 141
 Class `curve_element`, 143, 144
 Class `rect_element`, 146, 148
 Class `svg_element`, 150, 152
 Class `text_element`, 153, 155
 Class `ticks_labels_style`, 340
 Class `tspan_element`, 159, 162, 163
 Class `value_style`, 342
 Function operator`<<`, 166
 Implementation and other notes, 363
 Struct `polygon_element`, 136, 138

svg_color
 Class `svg_color`, 316
 Function `constant_to_rgb`, 319

symbols

 Class plot_point_style, 332, 333, 334
 Class svg_1d_plot_series, 212, 215

T

t

 Class path_element, 129, 132
 Function operator<<, 166

T

 Class path_element, 129, 133

text

 Class g_element, 119, 123
 Class svg, 172, 177
 Class text_element, 153, 155, 156
 Class tspan_element, 159, 163
 Header < boost/svg_plot/svg_style.hpp >, 321

textstyle

 Class text_element, 153, 156
 Class tspan_element, 159, 163

text_length

 Class tspan_element, 159, 163

three_sd_color

 Class svg_1d_plot, 179, 195
 Class svg_2d_plot, 216, 233, 234
 Class svg_boxplot, 268, 288

title

 Class svg_1d_plot, 179, 187, 189, 190, 192, 193, 195, 196, 197
 Class svg_2d_plot, 216, 223, 226, 228, 229, 231, 232, 234, 235
 Class svg_boxplot, 268, 278, 281, 282, 285, 286, 288, 289, 290, 295
 Class svg_boxplot_series, 309, 310, 315

title_color

 Class svg_1d_plot, 179, 195
 Class svg_2d_plot, 216, 234
 Class svg_boxplot, 268, 288, 289

title_font_alignment

 Class svg_1d_plot, 179, 196
 Class svg_2d_plot, 216, 234
 Class svg_boxplot, 268, 289

title_font_decoration

 Class svg_1d_plot, 179, 196
 Class svg_2d_plot, 216, 234
 Class svg_boxplot, 268, 289

title_font_family

 Class svg_1d_plot, 179, 196
 Class svg_2d_plot, 216, 234
 Class svg_boxplot, 268, 289

title_font_rotation

 Class svg_1d_plot, 179, 196
 Class svg_2d_plot, 216, 234, 235
 Class svg_boxplot, 268, 289

title_font_size

 Class svg_1d_plot, 179, 196
 Class svg_2d_plot, 216, 235
 Class svg_boxplot, 268, 289

title_font_stretch

 Class svg_1d_plot, 179, 196
 Class svg_2d_plot, 216, 235

Class `svg_boxplot`, 268, 289
`title_font_style`
 Class `svg_1d_plot`, 179, 196, 197
 Class `svg_2d_plot`, 216, 235
 Class `svg_boxplot`, 268, 290
`title_font_weight`
 Class `svg_1d_plot`, 179, 197
 Class `svg_2d_plot`, 216, 235
 Class `svg_boxplot`, 268, 290
`title_on`
 Class `svg_1d_plot`, 179, 197
 Class `svg_2d_plot`, 216, 235
 Class `svg_boxplot`, 268, 290
`title_size`
 Class `svg_boxplot`, 268, 290
`triangle`
 Class `g_element`, 119, 123
 Class `svg`, 172, 177
`ts`
 Function operator`<<`, 345
`tspan`
 Class `text_element`, 153, 156
`two_sd_color`
 Class `svg_1d_plot`, 179, 197
 Class `svg_2d_plot`, 216, 235, 236
 Class `svg_boxplot`, 268, 290
`types`
 Class `template unc`, 348, 349
`t_or_b`
 Header <`boost/svg_plot/show_1d_settings.hpp`>, 169

U

`u1`
 Demonstration of using 2D data that includes information about its uncertainty, 76
`u2`
 Demonstration of using 2D data that includes information about its uncertainty, 76
`uncertainty`
 Class `template unc`, 347, 348, 349, 350
 Function template `unc_of`, 351, 352
`unc_of`
 Function template `unc_of`, 351, 352
 Header <`boost/svg_plot/uncertain.hpp`>, 346
`use_down_ticks`
 Class `ticks_labels_style`, 340, 341
`use_style`
 Class `tspan_element`, 159, 163
`use_up_ticks`
 Class `ticks_labels_style`, 340, 341

V

`v`
 Class `path_element`, 129, 133
`V`
 Class `path_element`, 129, 133
`value`
 2-D Data Values Examples, 67
 Class `axis_line_style`, 323

Class `svg_boxplot`, 279, 284, 285
Class `template unc`, 348, 349
Class `ticks_labels_style`, 340
Tutorial: 2D Special Features, 62
`values_count`
 Class `svg_2d_plot_series`, 263, 266
`value_of`
 Function template `value_of`, 353, 354
 Header <`boost/svg_plot/uncertain.hpp`>, 346

W

`whisker_length`
 Class `svg_boxplot`, 268, 290
 Class `svg_boxplot_series`, 309, 315
`width`
 Class `axis_line_style`, 323, 324
 Class `bar_style`, 325, 326
 Class `box_style`, 327, 328
 Class `plot_line_style`, 330, 331
 Class `rect_element`, 146, 148
 Class `svg_1d_plot`, 188, 202, 207
 Class `svg_2d_plot`, 227, 240, 245
 Class `svg_boxplot`, 280, 288, 296
 Class `svg_style`, 335
 Demonstration of using 1D data that includes information about its Uncertainty, 54
`width_on`
 Class `svg_style`, 334, 336
`write`
 Class `circle_element`, 112, 113
 Class `clip_path_element`, 114, 115
 Class `ellipse_element`, 116, 118
 Class `g_element`, 119, 123
 Class `line_element`, 126, 127
 Class `path_element`, 129, 133
 Class `polyline_element`, 139, 141
 Class `curve_element`, 143, 145
 Class `rect_element`, 146, 148
 Class `svg`, 172, 177
 Class `svg_1d_plot`, 179, 197
 Class `svg_2d_plot`, 216, 236
 Class `svg_boxplot`, 268, 291
 Class `svg_color`, 316, 317
 Class `svg_element`, 150, 152
 Class `svg_style`, 334, 336
 Class `text_element`, 153, 156
 Class `text_element_text`, 157
 Class `text_parent`, 158
 Class `tspan_element`, 159, 163
 Struct `a_path`, 110
 Struct `c_path`, 111
 Struct `h_path`, 124
 Struct `l_path`, 125
 Struct `m_path`, 128
 Struct `path_point`, 134
 Struct `polygon_element`, 136, 138
 Struct `poly_path_point`, 135
 Struct `q_path`, 142

Struct s_path, 149
 Struct t_path, 152
 Struct v_path, 164, 165
 Struct z_path, 165
 write_attributes
 Class circle_element, 112, 114
 Class clip_path_element, 114, 116
 Class ellipse_element, 116, 118
 Class g_element, 119, 123
 Class line_element, 126, 128
 Class path_element, 129, 133
 Class polyline_element, 139, 141
 Class quurve_element, 143, 145
 Class rect_element, 146, 148
 Class svg_element, 150, 151
 Class text_element, 153, 157
 Class tspan_element, 159, 164
 Struct polygon_element, 136, 138

X

x
 Class rect_element, 146, 148
 Class text_element, 153, 156
 Class tspan_element, 159, 163, 164

xy_autoscale
 Class svg_2d_plot, 216, 249

xy_values_on
 Class svg_2d_plot, 216, 249

x_addlimits_color
 Class svg_1d_plot, 179, 198
 Class svg_2d_plot, 216, 236
 Class svg_boxplot, 268, 291

x_addlimits_on
 Class svg_1d_plot, 179, 198
 Class svg_2d_plot, 216, 236
 Class svg_boxplot, 268, 291

x_autoscale
 1-D Auto scaling Examples, 31
 Class svg_1d_plot, 179, 198, 199, 204, 207
 Class svg_2d_plot, 216, 236, 237, 242, 245
 Class svg_boxplot, 268, 291, 292, 298, 301
 Tutorial: 1D Autoscale with Multiple Containers, 18

x_auto_max_value
 Class svg_1d_plot, 179, 198
 Class svg_2d_plot, 216, 236
 Class svg_boxplot, 268, 291

x_auto_min_value
 Class svg_1d_plot, 179, 198
 Class svg_2d_plot, 216, 236
 Class svg_boxplot, 268, 291

x_auto_ticks
 Class svg_1d_plot, 179, 198
 Class svg_2d_plot, 216, 236
 Class svg_boxplot, 268, 291

x_auto_tick_interval
 Class svg_1d_plot, 179, 198
 Class svg_2d_plot, 216, 236

Class `svg_boxplot`, 268, 291
`x_axis`
 Class `svg_2d_plot`, 216, 237
`x_axis_color`
 1-D Auto scaling Examples, 31
 Class `svg_1d_plot`, 179, 199
 Class `svg_2d_plot`, 216, 237
 Class `svg_boxplot`, 268, 292
`x_axis_label_color`
 Class `svg_1d_plot`, 179, 199
 Class `svg_2d_plot`, 216, 237
 Class `svg_boxplot`, 268, 292
`x_axis_on`
 Class `svg_1d_plot`, 179, 199
 Class `svg_2d_plot`, 216, 237
 Class `svg_boxplot`, 268, 292
`x_axis_position`
 Class `svg_1d_plot`, 179, 199
 Class `svg_2d_plot`, 216, 237
 Class `svg_boxplot`, 268, 292
`x_axis_vertical`
 Class `svg_1d_plot`, 179, 199
 Class `svg_2d_plot`, 216, 238
 Class `svg_boxplot`, 268, 293
`x_axis_width`
 Class `svg_1d_plot`, 179, 200
 Class `svg_2d_plot`, 216, 238
 Class `svg_boxplot`, 268, 293
`x_datetime_color`
 Class `svg_1d_plot`, 179, 200
 Class `svg_2d_plot`, 216, 238
 Class `svg_boxplot`, 268, 293
`x_datetime_on`
 Class `svg_1d_plot`, 179, 200
 Class `svg_2d_plot`, 216, 238
 Class `svg_boxplot`, 268, 293
`x_decor`
 Class `svg_1d_plot`, 179, 200
 Class `svg_2d_plot`, 216, 238
 Class `svg_boxplot`, 268, 293
`x_df_color`
 Class `svg_1d_plot`, 179, 200
 Class `svg_2d_plot`, 216, 238
 Class `svg_boxplot`, 268, 293
`x_df_on`
 Class `svg_1d_plot`, 179, 200
 Class `svg_2d_plot`, 216, 238, 239
 Class `svg_boxplot`, 268, 293, 294
`x_id_color`
 Class `svg_1d_plot`, 179, 200, 201
 Class `svg_2d_plot`, 216, 239
 Class `svg_boxplot`, 268, 294
`x_id_on`
 Class `svg_1d_plot`, 179, 201
 Class `svg_2d_plot`, 216, 239
 Class `svg_boxplot`, 268, 294
`x_label`
 Class `svg_1d_plot`, 179, 201

Class `svg_2d_plot`, 216, 239
Class `svg_boxplot`, 268, 294
`x_labels_strip_e0s`
 Class `svg_1d_plot`, 179, 202
 Class `svg_2d_plot`, 216, 240
 Class `svg_boxplot`, 268, 296
`x_label_color`
 Class `svg_1d_plot`, 179, 201
 Class `svg_2d_plot`, 216, 239
 Class `svg_boxplot`, 268, 294
`x_label_font_family`
 Class `svg_1d_plot`, 179, 201
 Class `svg_2d_plot`, 216, 239
 Class `svg_boxplot`, 268, 294, 295
`x_label_font_size`
 Class `svg_1d_plot`, 179, 201
 Class `svg_2d_plot`, 216, 239, 240
 Class `svg_boxplot`, 268, 295
`x_label_on`
 Class `svg_1d_plot`, 179, 201, 202
 Class `svg_2d_plot`, 216, 239, 240
 Class `svg_boxplot`, 268, 295
`x_label_size`
 Class `svg_boxplot`, 268, 295
`x_label_text`
 Class `svg_boxplot`, 268, 295
`x_label_units`
 Class `svg_1d_plot`, 179, 202
 Class `svg_2d_plot`, 216, 240
 Class `svg_boxplot`, 268, 295
`x_label_units_on`
 Class `svg_1d_plot`, 179, 202
 Class `svg_2d_plot`, 216, 240
 Class `svg_boxplot`, 268, 295, 296
`x_label_width`
 Class `svg_1d_plot`, 179, 202
 Class `svg_2d_plot`, 216, 240
 Class `svg_boxplot`, 268, 296
`x_major_grid_color`
 Class `svg_1d_plot`, 179, 202
 Class `svg_2d_plot`, 216, 241
 Class `svg_boxplot`, 268, 296
`x_major_grid_on`
 Class `svg_1d_plot`, 179, 202, 203
 Class `svg_2d_plot`, 216, 241
 Class `svg_boxplot`, 268, 296
`x_major_grid_width`
 Class `svg_1d_plot`, 179, 203
 Class `svg_2d_plot`, 216, 241
 Class `svg_boxplot`, 268, 296
`x_major_interval`
 Class `svg_1d_plot`, 179, 203
 Class `svg_2d_plot`, 216, 241
 Class `svg_boxplot`, 268, 296
`x_major_labels`
 Class `svg_boxplot`, 268, 297
`x_major_labels_on`
 Class `svg_boxplot`, 268, 297

x_major_labels_side
Class svg_1d_plot, 179, 203
Class svg_2d_plot, 216, 241, 242
Class svg_boxplot, 268, 297

x_major_label_rotation
Class svg_1d_plot, 179, 203
Class svg_2d_plot, 216, 241
Class svg_boxplot, 268, 297

x_major_tick
Class svg_1d_plot, 179, 203
Class svg_2d_plot, 216, 242
Class svg_boxplot, 268, 297

x_major_tick_color
Class svg_1d_plot, 179, 204
Class svg_2d_plot, 216, 242
Class svg_boxplot, 268, 297

x_major_tick_length
Class svg_1d_plot, 179, 204
Class svg_2d_plot, 216, 242
Class svg_boxplot, 268, 298

x_major_tick_width
Class svg_1d_plot, 179, 204
Class svg_2d_plot, 216, 242
Class svg_boxplot, 268, 298

x_max
Class svg_1d_plot, 179, 204
Class svg_2d_plot, 216, 242
Class svg_boxplot, 268, 298

x_min
Class svg_1d_plot, 179, 204
Class svg_2d_plot, 216, 242, 243
Class svg_boxplot, 268, 298

x_minor_grid_color
Class svg_1d_plot, 179, 205
Class svg_2d_plot, 216, 243
Class svg_boxplot, 268, 298

x_minor_grid_on
Class svg_1d_plot, 179, 205
Class svg_2d_plot, 216, 243
Class svg_boxplot, 268, 299

x_minor_grid_width
Class svg_1d_plot, 179, 205
Class svg_2d_plot, 216, 243
Class svg_boxplot, 268, 299

x_minor_interval
Class svg_1d_plot, 179, 205
Class svg_2d_plot, 216, 243
Class svg_boxplot, 268, 299

x_minor_tick_color
Class svg_1d_plot, 179, 205
Class svg_2d_plot, 216, 243
Class svg_boxplot, 268, 299

x_minor_tick_length
Class svg_1d_plot, 179, 205
Class svg_2d_plot, 216, 244
Class svg_boxplot, 268, 299

x_minor_tick_width
Class svg_1d_plot, 179, 205, 206

Class `svg_2d_plot`, 216, 244
Class `svg_boxplot`, 268, 299
`x_min_ticks`
 Class `svg_1d_plot`, 179, 204
 Class `svg_2d_plot`, 216, 243
 Class `svg_boxplot`, 268, 298
`x_num_minor_ticks`
 Class `svg_1d_plot`, 179, 206
 Class `svg_2d_plot`, 216, 244
 Class `svg_boxplot`, 268, 300
 Implementation and other notes, 363
`x_order_color`
 Class `svg_1d_plot`, 179, 206
 Class `svg_2d_plot`, 216, 244
 Class `svg_boxplot`, 268, 300
`x_order_on`
 Class `svg_1d_plot`, 179, 206
 Class `svg_2d_plot`, 216, 244
 Class `svg_boxplot`, 268, 300
`x_plusminus_color`
 Class `svg_1d_plot`, 179, 206
 Class `svg_2d_plot`, 216, 244
 Class `svg_boxplot`, 268, 300
`x_plusminus_on`
 Class `svg_1d_plot`, 179, 206
 Class `svg_2d_plot`, 216, 245
 Class `svg_boxplot`, 268, 300
`x_prefix`
 Class `svg_1d_plot`, 179, 206
 Class `svg_2d_plot`, 216, 245
 Class `svg_boxplot`, 268, 300
`x_range`
 1-D Auto scaling Examples, 31
 1-D Autoscaling Various Containers Examples, 40
 1-D Axis Scaling, 26
 Class `svg_1d_plot`, 179, 207
 Class `svg_2d_plot`, 216, 245
 Class `svg_boxplot`, 268, 300
 Tutorial: 1D Autoscale with Multiple Containers, 18
`x_separator`
 Class `svg_1d_plot`, 179, 207
 Class `svg_2d_plot`, 216, 245
 Class `svg_boxplot`, 268, 301
`x_size`
 1-D Data Values Examples, 43
 Class `svg`, 172, 177, 178
 Class `svg_1d_plot`, 179, 188, 207
 Class `svg_2d_plot`, 216, 227, 245
 Class `svg_boxplot`, 268, 280, 301
`x_steps`
 Class `svg_1d_plot`, 179, 207
 Class `svg_2d_plot`, 216, 245
 Class `svg_boxplot`, 268, 301
`x_suffix`
 Class `svg_1d_plot`, 179, 207
 Class `svg_2d_plot`, 216, 246
 Class `svg_boxplot`, 268, 301
`x_ticks`

Class `svg_2d_plot`, 216, 246
x_ticks_down_on
 Class `svg_1d_plot`, 179, 207
 Class `svg_2d_plot`, 216, 246
 Class `svg_boxplot`, 268, 302
x_ticks_on_window_or_axis
 Class `svg_1d_plot`, 179, 208
 Class `svg_2d_plot`, 216, 246
 Class `svg_boxplot`, 268, 302
x_ticks_up_on
 Class `svg_1d_plot`, 179, 208
 Class `svg_2d_plot`, 216, 246
 Class `svg_boxplot`, 268, 302
x_ticks_values_color
 Class `svg_1d_plot`, 179, 208
 Class `svg_2d_plot`, 216, 246
 Class `svg_boxplot`, 268, 302
x_ticks_values_font_family
 Class `svg_1d_plot`, 179, 208
 Class `svg_2d_plot`, 216, 246
 Class `svg_boxplot`, 268, 302, 303
x_ticks_values_font_size
 Class `svg_1d_plot`, 179, 208
 Class `svg_2d_plot`, 216, 247
 Class `svg_boxplot`, 268, 303
x_ticks_values_ioflags
 Class `svg_1d_plot`, 179, 208, 209
 Class `svg_2d_plot`, 216, 247
 Class `svg_boxplot`, 268, 303
x_ticks_values_precision
 Class `svg_1d_plot`, 179, 209
 Class `svg_2d_plot`, 216, 247
 Class `svg_boxplot`, 268, 303
x_tick_color
 Class `svg_boxplot`, 268, 301
x_tick_length
 Class `svg_boxplot`, 268, 301
x_tick_width
 Class `svg_boxplot`, 268, 302
x_tight
 Class `svg_1d_plot`, 179, 209
 Class `svg_2d_plot`, 216, 247
 Class `svg_boxplot`, 268, 303
x_values_color
 Class `svg_1d_plot`, 179, 209, 210
 Class `svg_2d_plot`, 216, 248
 Class `svg_boxplot`, 268, 304
x_values_font_family
 Class `svg_1d_plot`, 179, 210
 Class `svg_2d_plot`, 216, 248
 Class `svg_boxplot`, 268, 304
x_values_font_size
 Class `svg_1d_plot`, 179, 210
 Class `svg_2d_plot`, 216, 248
 Class `svg_boxplot`, 268, 304
x_values_ioflags
 Class `svg_1d_plot`, 179, 210
 Class `svg_2d_plot`, 216, 248

Class `svg_boxplot`, 268, 304
`x_values_on`
 Class `svg_1d_plot`, 179, 210
 Class `svg_2d_plot`, 216, 248
 Class `svg_boxplot`, 268, 304
`x_values_precision`
 Class `svg_1d_plot`, 179, 210
 Class `svg_2d_plot`, 216, 249
 Class `svg_boxplot`, 268, 305
`x_values_rotation`
 Class `svg_1d_plot`, 179, 210, 211
 Class `svg_2d_plot`, 216, 249
 Class `svg_boxplot`, 268, 305
`x_value_font_size`
 Class `svg_1d_plot`, 179, 209
 Class `svg_2d_plot`, 216, 247
 Class `svg_boxplot`, 268, 303
`x_value_ioflags`
 Class `svg_1d_plot`, 179, 209
 Class `svg_2d_plot`, 216, 247, 248
 Class `svg_boxplot`, 268, 303, 304
`x_value_precision`
 Class `svg_1d_plot`, 179, 209
 Class `svg_2d_plot`, 216, 248
 Class `svg_boxplot`, 268, 304
`x_with_zero`
 Class `svg_1d_plot`, 179, 211
 Class `svg_2d_plot`, 216, 249
 Class `svg_boxplot`, 268, 305

Y

`y`
 Class `rect_element`, 146, 148
 Class `text_element`, 153, 156
 Class `tspan_element`, 159, 164
`y_addlimits_color`
 Class `svg_2d_plot`, 216, 249, 250
`y_addlimits_on`
 Class `svg_2d_plot`, 216, 250
`y_autoscale`
 Class `svg_2d_plot`, 216, 250
 Class `svg_boxplot`, 268, 305, 306
`y_axis`
 Class `svg_2d_plot`, 216, 250
`y_axis_color`
 Class `svg_1d_plot`, 179, 211
 Class `svg_2d_plot`, 216, 251
 Class `svg_boxplot`, 268, 306
`y_axis_label_color`
 Class `svg_2d_plot`, 216, 251
`y_axis_on`
 Class `svg_1d_plot`, 179, 211
 Class `svg_2d_plot`, 216, 251
 Class `svg_boxplot`, 268, 306
`y_axis_position`
 Class `svg_2d_plot`, 216, 251
 Class `svg_boxplot`, 268, 306

y_axis_value_color
 Class `svg_2d_plot`, 216, 251
y_axis_width
 Class `svg_2d_plot`, 216, 251, 252
y_decor
 Class `svg_2d_plot`, 216, 252
y_df_color
 Class `svg_2d_plot`, 216, 252
y_df_on
 Class `svg_2d_plot`, 216, 252
y_label
 Class `svg_1d_plot`, 179, 211
 Class `svg_2d_plot`, 216, 252
 Class `svg_boxplot`, 268, 306
y_labels_strip_e0s
 Class `svg_1d_plot`, 179, 212
 Class `svg_2d_plot`, 216, 254
 Class `svg_boxplot`, 268, 307
y_label_axis
 Class `svg_2d_plot`, 216, 252, 253
y_label_color
 Class `svg_1d_plot`, 179, 211
 Class `svg_2d_plot`, 216, 253
 Class `svg_boxplot`, 268, 306, 307
y_label_font_family
 Class `svg_2d_plot`, 216, 253
y_label_font_size
 Class `svg_2d_plot`, 216, 253
 Class `svg_boxplot`, 268, 307
y_label_on
 Class `svg_1d_plot`, 211
 Class `svg_2d_plot`, 216, 252, 253
 Class `svg_boxplot`, 268, 306, 307
y_label_text
 Class `svg_boxplot`, 268, 307
y_label_units
 Class `svg_1d_plot`, 179, 211
 Class `svg_2d_plot`, 216, 253, 254
 Class `svg_boxplot`, 268, 307
y_label_units_on
 Class `svg_2d_plot`, 216, 254
y_label_weight
 Class `svg_2d_plot`, 216, 254
y_label_width
 Class `svg_2d_plot`, 216, 254
y_major_grid_color
 Class `svg_2d_plot`, 216, 254, 255
y_major_grid_on
 Class `svg_2d_plot`, 216, 255
y_major_grid_width
 Class `svg_2d_plot`, 216, 255
y_major_interval
 Class `svg_2d_plot`, 216, 255
 Class `svg_boxplot`, 268, 307
y_major_labels_on
 Class `svg_boxplot`, 268, 307
y_major_labels_side
 Class `svg_2d_plot`, 216, 256

y_major_label_rotation
 Class `svg_2d_plot`, 216, 255
y_major_tick_color
 Class `svg_2d_plot`, 216, 256
 Class `svg_boxplot`, 268, 307
y_major_tick_length
 Class `svg_2d_plot`, 216, 256
 Class `svg_boxplot`, 268, 307
y_major_tick_width
 Class `svg_2d_plot`, 216, 256
 Class `svg_boxplot`, 268, 308
y_max
 Class `svg_2d_plot`, 216, 256
y_min
 Class `svg_2d_plot`, 216, 256
y_minor_grid_color
 Class `svg_2d_plot`, 216, 257
y_minor_grid_on
 Class `svg_2d_plot`, 216, 257
y_minor_grid_width
 Class `svg_2d_plot`, 216, 257
y_minor_interval
 Class `svg_1d_plot`, 179, 212
 Class `svg_2d_plot`, 216, 257
 Class `svg_boxplot`, 268, 308
y_minor_tick_color
 Class `svg_2d_plot`, 216, 257
 Class `svg_boxplot`, 268, 308
y_minor_tick_length
 Class `svg_2d_plot`, 216, 257
 Class `svg_boxplot`, 268, 308
y_minor_tick_width
 Class `svg_2d_plot`, 216, 258
 Class `svg_boxplot`, 268, 308
y_num_minor_ticks
 Class `svg_2d_plot`, 216, 258
 Class `svg_boxplot`, 268, 308
y_plusminus_color
 Class `svg_2d_plot`, 216, 258
y_plusminus_on
 Class `svg_2d_plot`, 216, 258
y_prefix
 Class `svg_2d_plot`, 216, 258
y_range
 Class `svg_2d_plot`, 216, 258, 259
 Class `svg_boxplot`, 268, 308
y_separator
 Class `svg_2d_plot`, 216, 259
y_size
 1-D Data Values Examples, 43
 Class `svg`, 172, 178
 Class `svg_1d_plot`, 179, 188, 212
 Class `svg_2d_plot`, 216, 227, 259
 Class `svg_boxplot`, 268, 280, 308, 309
y_suffix
 Class `svg_2d_plot`, 216, 259
y_ticks
 Class `svg_2d_plot`, 216, 259

y_ticks_left_on
 Class svg_2d_plot, 216, 259
 y_ticks_on_window_or_axis
 Class svg_2d_plot, 216, 259
 y_ticks_right_on
 Class svg_2d_plot, 216, 260
 y_ticks_values_color
 Class svg_2d_plot, 216, 260
 y_ticks_values_font_family
 Class svg_2d_plot, 216, 260
 y_ticks_values_font_size
 Class svg_2d_plot, 216, 260
 y_ticks_values_ioflags
 Class svg_2d_plot, 216, 260, 261
 y_ticks_values_precision
 Class svg_2d_plot, 216, 261
 y_values_color
 Class svg_2d_plot, 216, 261
 y_values_font_family
 Class svg_2d_plot, 216, 262
 y_values_font_size
 Class svg_2d_plot, 216, 262
 y_values_ioflags
 Class svg_2d_plot, 216, 262
 y_values_on
 Class svg_2d_plot, 216, 262
 y_values_precision
 Class svg_2d_plot, 216, 262
 y_values_rotation
 Class svg_2d_plot, 216, 263
 y_value_ioflags
 Class svg_2d_plot, 216, 261
 y_value_precision
 Class svg_2d_plot, 216, 261

Z

z
 Class path_element, 129, 133
Z
 Class path_element, 129, 133

Index

Symbols

1-D Auto scaling Examples
 autoscale_check_limits, 31
 background, 31
 color, 31
 container, 31
 data, 31
 example, 31
 fill, 31
 image_border_width, 31
 legend, 31
 maximum, 31, 36
 minimum, 31, 36

origin, 31
scaling, 31
series, 31
show_1d_plot_settings, 31
stroke, 31
SVG, 31
title, 31
vector_iterator, 31
x_autoscale, 31
x_axis_color, 31
x_range, 31

1-D Autoscaling Various Containers Examples

autoscale, 40
container, 40
data, 40
example, 40
fill, 40
label, 40
maximum, 40
minimum, 40
series, 40
SVG, 40
title, 40
x_range, 40

1-D Axis Scaling

container, 26
data, 26
example, 26
fill, 26
maximum, 26
minimum, 26
origin, 26
scaling, 26
series, 26
SVG, 26
vector_iterator, 26
x_range, 26

1-D Data Values Examples

color, 43
container, 43
data, 43
example, 43
fill, 43
font, 43
ioflags, 43
label, 43
marker, 43
precision, 43
series, 43
show_1d_plot_settings, 43
stroke, 43
SVG, 43
title, 43
Unicode, 43
x_size, 43
y_size, 43

1-D STL Containers Examples

container, 15

data, 15
deque, 15
example, 15
series, 15
SVG, 15, 17, 18
title, 15
1-D Vector Example
background, 14
color, 14
container, 14
data, 14
example, 14
legend, 14
marker, 14
scaling, 14
series, 14
SVG, 14
title, 14
1D Tutorials
data, 14
example, 14
2-D Autoscaling Examples
container, 73
data, 73
example, 73
iter, 73
maximum, 73
minimum, 73
s, 73
series, 73
SVG, 73
title, 73
2-D Data Values Examples
color, 67
container, 67
data, 67, 69
example, 67
fill, 67
font, 67
ioflags, 67
label, 67
marker, 67
maximum, 67
precision, 67
series, 67
show_2d_plot_settings, 67
stroke, 67
SVG, 67
title, 67
uncertainty, 67
value, 67

A

Acknowledgements
color, 369
example, 369
minimum, 369

add_g_element
 Class g_element, 119, 120
 Class svg, 172, 173

alignment
 Class text_element, 153, 154

area_fill
 Class bar_style, 325, 326
 Class plot_line_style, 330
 Class svg_2d_plot_series, 263, 264

attribution
 Class svg, 172, 173

author
 Class svg, 172, 173
 Implementation and other notes, 363

autoscale
 1-D Autoscaling Various Containers Examples, 40
 Class svg_1d_plot, 179, 184
 Class svg_2d_plot, 216, 223, 224
 Class svg_boxplot, 268, 274
 Demonstration of using 2D data that includes information about its uncertainty, 76

autoscale_check_limits
 1-D Auto scaling Examples, 31
 Class svg_1d_plot, 179, 184, 185
 Class svg_2d_plot, 216, 224
 Class svg_boxplot, 268, 274
 Demonstration of using 2D data that includes information about its uncertainty, 76

autoscale_plusminus
 Class svg_1d_plot, 179, 185
 Class svg_2d_plot, 216, 224
 Class svg_boxplot, 268, 275
 Demonstration of using 2D data that includes information about its uncertainty, 76

axes_on
 Class svg_1d_plot, 179, 185
 Class svg_2d_plot, 216, 224
 Class svg_boxplot, 268, 275

axis_color
 Class svg_boxplot, 268, 275
 Class svg_boxplot_series, 309, 310

axis_line_style
 Class axis_line_style, 323

axis_width
 Class svg_boxplot, 268, 275
 Class svg_boxplot_series, 309, 310, 311

a_path
 Struct a_path, 110

B

background
 1-D Auto scaling Examples, 31
 1-D Vector Example, 14
 Class g_element, 120, 123
 Class svg_1d_plot, 179, 185, 188, 189, 193, 194
 Class svg_2d_plot, 216, 223, 224, 227, 228, 232
 Class svg_boxplot, 268, 275, 276, 280, 281, 286
 Colors, 6
 Demonstration of using 1D data that includes information about its Uncertainty, 54
 Simple Code Example, 58, 59

Simple Example, 84
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 1D More Layout Examples, 20
Tutorial: Fuller Layout Example, 60
Using Inkscape, 356

background_border_color
 Class `svg_1d_plot`, 179, 185
 Class `svg_2d_plot`, 216, 224
 Class `svg_boxplot`, 268, 275

background_border_width
 Class `svg_1d_plot`, 179, 185
 Class `svg_2d_plot`, 216, 224
 Class `svg_boxplot`, 268, 275, 276

background_color
 Class `svg_1d_plot`, 179, 185
 Class `svg_2d_plot`, 216, 224
 Class `svg_boxplot`, 268, 276

bar_area_fill
 Class `svg_2d_plot_series`, 263, 264

bar_color
 Class `svg_2d_plot_series`, 263, 264

bar_opt
 Class `bar_style`, 325, 326
 Class `svg_2d_plot_series`, 263, 264, 265

bar_style
 Class `bar_style`, 325

bar_width
 Class `svg_2d_plot_series`, 263, 265

bezier_on
 Class `plot_line_style`, 330, 331
 Class `svg_1d_plot_series`, 212, 213
 Class `svg_2d_plot_series`, 263, 265

Boost.SVG plot C++ Reference
 SVG, 94

boost_license_on
 Class `svg`, 172, 173, 174
 Class `svg_1d_plot`, 179, 185
 Class `svg_2d_plot`, 216, 225
 Class `svg_boxplot`, 268, 276

border
 Class `box_style`, 327
 Class `g_element`, 120
 Class `svg_1d_plot`, 185, 188, 189, 194
 Class `svg_2d_plot`, 223, 224, 227, 228, 232
 Class `svg_boxplot`, 275, 276, 279, 280, 284, 286, 287, 292, 293, 306
 Class `svg_boxplot_series`, 311
 Class `svg_style`, 334
 Class `ticks_labels_style`, 340
 Colors, 6
 Demonstration of using 1D data that includes information about its Uncertainty, 54
 Demonstration of using 2D data that includes information about its uncertainty, 76
 Simple Code Example, 59
 Simple Example, 84

border_on
 Class `box_style`, 327

boxplot
 Class `box_style`, 326
 Class `svg_boxplot`, 267, 274, 285, 286, 288, 290, 291

Class `svg_boxplot_series`, 310, 311
Definitions of the Quartiles, 86
Function `quantile`, 169
Global `document_ids_`, 108
Header <`boost/svg_plot/detail/axis_plot_frame.hpp`>, 102
Header <`boost/svg_plot/detail/svg_boxplot_detail.hpp`>, 108
Header <`boost/svg_plot/svg.hpp`>, 171
Header <`boost/svg_plot/svg_boxplot.hpp`>, 267
Preface, 3
Real-life Heat flow data, 48
Simple Example, 84
Tutorial: Boxplot, 84
`box_border`
 Class `svg_boxplot`, 268, 276
 Class `svg_boxplot_series`, 309, 311
`box_fill`
 Class `svg_boxplot`, 268, 276
 Class `svg_boxplot_series`, 309, 311
`box_style`
 Class `box_style`, 327
 Class `svg_boxplot_series`, 309, 311
`box_width`
 Class `svg_boxplot`, 268, 276
 Class `svg_boxplot_series`, 309, 311

C

`c`
 Class `path_element`, 129, 130
`C`
 Class `path_element`, 129, 130
`calculate_quantiles`
 Class `svg_boxplot_series`, 309, 311
`circle`
 Class `g_element`, 119, 120
 Class `svg`, 172, 174
`circle_element`
 Class `circle_element`, 112
`Class axis_line_style`
 `axis_line_style`, 323
 `color`, 323, 324
 `data`, 323, 325
 `example`, 323, 324
 `label`, 323, 324
 `label_on`, 323, 324
 `label_units_on`, 323, 324
 `line`, 323
 `maximum`, 323
 `minimum`, 323
 `position`, 323, 324
 `scaling`, 323
 `stroke`, 323
 `SVG`, 322, 323
 `value`, 323
 `width`, 323, 324
`Class bar_style`
 `area_fill`, 325, 326
 `bar_opt`, 325, 326

bar_style, 325
color, 325, 326
data, 325
fill, 325, 326
histogram, 325
line, 325
stroke, 325
SVG, 325
width, 325, 326

Class box_style
border, 327
border_on, 327
boxplot, 326
box_style, 327
color, 327, 328
data, 327
fill, 327, 328
fill_on, 327, 328
line, 327
margin, 327, 328
stroke, 327, 328
SVG, 326
width, 327, 328

Class circle_element
circle_element, 112
class_id, 112, 113
clip_id, 112, 113
color, 113, 114
data, 112
example, 112, 113
fill, 114
id, 112, 113
stroke, 114
style, 112, 113
SVG, 111, 112, 113
svg, 112
svg_element, 112
write, 112, 113
write_attributes, 112, 114

Class clip_path_element
class_id, 114, 115
clip_id, 114, 115
clip_path_element, 114
color, 115, 116
data, 114
example, 114, 115
fill, 116
id, 114, 115
stroke, 116
style, 114, 115
SVG, 114, 115
svg, 114
svg_element, 114
write, 114, 115
write_attributes, 114, 116

Class ellipse_element
class_id, 116, 117
clip_id, 116, 117

color, 118
data, 116, 117
ellipse_element, 116
example, 117, 118
fill, 118
id, 116, 117
stroke, 118
style, 116, 117, 118
SVG, 116, 117, 118
svg, 116
svg_element, 116
write, 116, 118
write_attributes, 116, 118

Class g_element
add_g_element, 119, 120
background, 120, 123
border, 120
circle, 119, 120
class_id, 119, 120
clear, 119, 120
clip_id, 119, 120, 121
color, 123
container, 118, 120, 124
data, 119, 124
ellipse, 119, 121
example, 120, 121, 123
fill, 120, 123
g, 119, 121
g_element, 119
hexagon, 119, 121
id, 119, 121
legend, 123
line, 119, 121
path, 119, 122
pentagon, 119, 122
polygon, 119, 122
polyline, 119, 122
push_back, 119, 122
rect, 119, 122
rhombus, 119, 123
size, 119, 123
stroke, 123
style, 119, 123
SVG, 118, 119, 120, 121, 123
svg, 119
svg_element, 119
text, 119, 123
triangle, 119, 123
write, 119, 123
write_attributes, 119, 123

Class histogram_style
data, 329
fill, 329
histogram, 328, 329
histogram_style, 329
SVG, 328

Class line_element
class_id, 126, 127

clip_id, 126, 127
color, 127, 128
data, 126
example, 126, 127
fill, 128
id, 126, 127
line_element, 126
stroke, 128
style, 126, 127
SVG, 125, 126, 127
svg, 126
svg_element, 126
write, 126, 127
write_attributes, 126, 128

Class path_element
c, 129, 130
C, 129, 130
class_id, 129, 130
clip_id, 129, 130
color, 132, 133
container, 129
data, 129
example, 130, 131, 133
fill, 130, 131, 133
fill_on, 129, 130, 131
h, 129, 131
H, 129, 131
id, 129, 131
l, 129, 131
L, 129, 131
m, 129, 131
M, 129, 132
path_element, 129
q, 129, 132
Q, 129, 132
s, 129, 132
S, 129, 132
scaling, 129
stroke, 133
style, 129, 132
SVG, 128, 129, 130, 131, 133
svg, 129
svg_element, 129
t, 129, 132
T, 129, 133
v, 129, 133
V, 129, 133
write, 129, 133
write_attributes, 129, 133
z, 129, 133
Z, 129, 133

Class plot_line_style
area_fill, 330
bezier_on, 330, 331
color, 330, 331
data, 329, 330, 331
fill, 330
line, 330, 331

line_on, 330, 331
plot_line_style, 330
series, 329, 330
stroke, 330
SVG, 329
width, 330, 331

Class plot_point_style
color, 331, 332, 333
data, 331, 332, 333
example, 332
fill, 332, 333
fill_color, 332, 333
font, 332, 333
marker, 331
plot_point_style, 332
shape, 332, 333
size, 332, 333
stroke, 332, 333
stroke_color, 332, 333
style, 332, 333
SVG, 331
symbols, 332, 333, 334
uncertainty, 332
Unicode, 332

Class polyline_element
class_id, 139, 140
clip_id, 139, 140
color, 141
container, 139
data, 139
example, 140, 141
fill, 141
id, 139, 140
info, 141
P, 139, 141
polyline_element, 139
stroke, 141
style, 139, 141
SVG, 139, 140, 141
svg, 139
svg_element, 139
write, 139, 141
write_attributes, 139, 141

Class quurve_element
class_id, 143, 144
clip_id, 143, 144
color, 144, 145
data, 143
example, 144, 145
fill, 145
id, 143, 144
quurve_element, 143
stroke, 145
style, 143, 144
SVG, 142, 143, 144, 145
svg, 143
svg_element, 143
write, 143, 145

write_attributes, 143, 145

Class rect_element

class_id, 146, 147

clip_id, 146, 147

color, 148

data, 146

example, 147, 148

fill, 148

id, 146, 147

rect_element, 146

stroke, 148

style, 146, 148

SVG, 145, 146, 147, 148

svg, 146

svg_element, 146

width, 146, 148

write, 146, 148

write_attributes, 146, 148

x, 146, 148

y, 146, 148

Class svg

add_g_element, 172, 173

attribution, 172, 173

author, 172, 173

boost_license_on, 172, 173, 174

circle, 172, 174

clip_path, 172, 174

commercialuse, 172, 174

coord_precision, 172, 174

copyright_date, 172, 174

copyright_holder, 172, 174

data, 174, 177

description, 172, 175

distribution, 172, 175

document_size, 172, 175

document_title, 172, 175

ellipse, 172, 175

example, 174

g, 172, 175

hexagon, 172, 175

image_filename, 172, 175

is, 177

license, 172, 176

license_on, 172, 176

line, 172, 176

path, 172, 176

pentagon, 172, 176

polygon, 172, 176

polyline, 172, 176, 177

precision, 174

rect, 172, 177

reproduction, 172, 177

rhombus, 172, 177

scaling, 177

shape, 176

size, 172, 177

SVG, 172, 173, 174, 175, 176, 177, 178

svg, 172

text, 172, 177
title, 174, 175
triangle, 172, 177
write, 172, 177
x_size, 172, 177, 178
y_size, 172, 178
Class `svg_1d_plot`
 autoscale, 179, 184
 autoscale_check_limits, 179, 184, 185
 autoscale_plusminus, 179, 185
 axes_on, 179, 185
 background, 179, 185, 188, 189, 193, 194
 background_border_color, 179, 185
 background_border_width, 179, 185
 background_color, 179, 185
 boost_license_on, 179, 185
 border, 185, 188, 189, 194
 color, 179, 185, 187, 188, 189, 191, 192, 193, 194, 195, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 208, 209, 210, 211
 confidence, 179, 186
 container, 184, 187, 192, 193, 199
 coord_precision, 179, 186, 197
 copyright_date, 179, 186
 copyright_holder, 179, 186
 data, 179, 184, 186, 187, 189, 192, 193, 194, 197, 198, 200, 201, 204, 206, 207, 208, 209, 210, 211
 data_lines_width, 179, 186
 derived, 179, 186, 187
 description, 179, 187
 document_title, 179, 187
 double, 193
 draw_line, 179, 187
 draw_note, 179, 187
 draw_plot_curve, 179, 187
 draw_plot_line, 179, 187
 example, 187, 192, 193, 195, 196, 197, 198, 199, 200, 201, 202, 206, 209, 212
 fill, 188, 189, 191, 192, 193, 194, 195, 197
 font, 189, 190, 196, 197, 201, 208, 209, 210
 greek, 187, 195
 grid, 202, 203, 205
 image_border_margin, 179, 188
 image_border_width, 179, 188
 image_x_size, 179, 188
 image_y_size, 179, 188
 label, 199, 201, 202, 203, 208, 209, 211, 212
 legend, 179, 188, 189, 190, 191
 legend_background_color, 179, 188
 legend_border_color, 179, 188
 legend_box_fill_on, 179, 189
 legend_color, 179, 189
 legend_font_family, 179, 189
 legend_font_weight, 179, 189
 legend_header_font_size, 179, 189
 legend_lines, 179, 189
 legend_on, 179, 190
 legend_outside, 179, 190
 legend_place, 179, 190
 legend_title, 179, 190
 legend_title_font_size, 179, 190
 legend_top_left, 179, 190

legend_width, 179, 191
license, 179, 191
license_attribution, 179, 191
license_commercialuse, 179, 191
license_distribution, 179, 191
license_on, 179, 191
license_reproduction, 179, 191
limit_color, 179, 191
limit_fill_color, 179, 191, 192
line, 203
marker, 198, 200, 201, 206, 208, 209, 210, 211
maximum, 194, 198, 204, 207
minimum, 194, 198, 204, 205, 206, 207, 212
one_sd_color, 179, 192
plot, 179, 192, 193, 195
plot_background_color, 179, 193, 194
plot_border_color, 179, 194
plot_border_width, 179, 194
plot_window_on, 179, 194
plot_window_x, 179, 194
plot_window_x_left, 179, 194
plot_window_x_right, 179, 194
plot_window_y, 179, 194, 195
plot_window_y_bottom, 179, 195
plot_window_y_top, 179, 195
position, 199
precision, 186, 197, 209, 210
rotation, 210, 211
scaling, 194, 197, 201, 208
series, 189, 192, 193
size, 179, 189, 195, 201, 207, 208, 209, 212
stroke, 188, 189, 191
SVG, 179, 185, 186, 187, 188, 189, 190, 191, 194, 195, 196, 197, 201, 204, 205, 206, 207, 208, 209, 212
svg_1d_plot, 179
three_sd_color, 179, 195
title, 179, 187, 189, 190, 192, 193, 195, 196, 197
title_color, 179, 195
title_font_alignment, 179, 196
title_font_decoration, 179, 196
title_font_family, 179, 196
title_font_rotation, 179, 196
title_font_size, 179, 196
title_font_stretch, 179, 196
title_font_style, 179, 196, 197
title_font_weight, 179, 197
title_on, 179, 197
two_sd_color, 179, 197
Unicode, 187, 195, 196, 200, 201, 207, 208
width, 188, 202, 207
write, 179, 197
x_addlimits_color, 179, 198
x_addlimits_on, 179, 198
x_autoscale, 179, 198, 199, 204, 207
x_auto_max_value, 179, 198
x_auto_min_value, 179, 198
x_auto_ticks, 179, 198
x_auto_tick_interval, 179, 198
x_axis_color, 179, 199

x_axis_label_color, 179, 199
x_axis_on, 179, 199
x_axis_position, 179, 199
x_axis_vertical, 179, 199
x_axis_width, 179, 200
x_datetime_color, 179, 200
x_datetime_on, 179, 200
x_decor, 179, 200
x_df_color, 179, 200
x_df_on, 179, 200
x_id_color, 179, 200, 201
x_id_on, 179, 201
x_label, 179, 201
x_labels_strip_e0s, 179, 202
x_label_color, 179, 201
x_label_font_family, 179, 201
x_label_font_size, 179, 201
x_label_on, 179, 201, 202
x_label_units, 179, 202
x_label_units_on, 179, 202
x_label_width, 179, 202
x_major_grid_color, 179, 202
x_major_grid_on, 179, 202, 203
x_major_grid_width, 179, 203
x_major_interval, 179, 203
x_major_labels_side, 179, 203
x_major_label_rotation, 179, 203
x_major_tick, 179, 203
x_major_tick_color, 179, 204
x_major_tick_length, 179, 204
x_major_tick_width, 179, 204
x_max, 179, 204
x_min, 179, 204
x_minor_grid_color, 179, 205
x_minor_grid_on, 179, 205
x_minor_grid_width, 179, 205
x_minor_interval, 179, 205
x_minor_tick_color, 179, 205
x_minor_tick_length, 179, 205
x_minor_tick_width, 179, 205, 206
x_min_ticks, 179, 204
x_num_minor_ticks, 179, 206
x_order_color, 179, 206
x_order_on, 179, 206
x_plusminus_color, 179, 206
x_plusminus_on, 179, 206
x_prefix, 179, 206
x_range, 179, 207
x_separator, 179, 207
x_size, 179, 188, 207
x_steps, 179, 207
x_suffix, 179, 207
x_ticks_down_on, 179, 207
x_ticks_on_window_or_axis, 179, 208
x_ticks_up_on, 179, 208
x_ticks_values_color, 179, 208
x_ticks_values_font_family, 179, 208
x_ticks_values_font_size, 179, 208

x_ticks_values_ioflags, 179, 208, 209
x_ticks_values_precision, 179, 209
x_tight, 179, 209
x_values_color, 179, 209, 210
x_values_font_family, 179, 210
x_values_font_size, 179, 210
x_values_ioflags, 179, 210
x_values_on, 179, 210
x_values_precision, 179, 210
x_values_rotation, 179, 210, 211
x_value_font_size, 179, 209
x_value_ioflags, 179, 209
x_value_precision, 179, 209
x_with_zero, 179, 211
y_axis_color, 179, 211
y_axis_on, 179, 211
y_label, 179, 211
y_labels_strip_e0s, 179, 212
y_label_color, 179, 211
y_label_on, 211
y_label_units, 179, 211
y_minor_interval, 179, 212
y_size, 179, 188, 212

Class svg_1d_plot_series
bezier_on, 212, 213
color, 212, 213, 214, 215
container, 213
data, 212, 213, 214
fill, 213
fill_color, 212, 213
legend, 213
limit_point_color, 212, 213
line_color, 212, 213
line_on, 212, 214
line_width, 212, 214
marker, 213, 214, 215
series, 212, 213, 214
series_count, 212, 214
series_limits_count, 212, 214
shape, 212, 214
size, 212, 214
stroke, 213, 214, 215
stroke_color, 212, 214, 215
SVG, 212
svg_1d_plot_series, 212
symbols, 212, 215
title, 213
uncertainty, 213

Class svg_2d_plot
autoscale, 216, 223, 224
autoscale_check_limits, 216, 224
autoscale_plusminus, 216, 224
axes_on, 216, 224
background, 216, 223, 224, 227, 228, 232
background_border_color, 216, 224
background_border_width, 216, 224
background_color, 216, 224
boost_license_on, 216, 225

border, 223, 224, 227, 228, 232
color, 216, 223, 224, 226, 227, 228, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 241, 242, 243, 244, 246, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 260, 261
confidence, 216, 225
container, 223, 226, 231, 232, 237, 249, 250
coord_precision, 216, 225
copyright_date, 216, 225
copyright_holder, 216, 225
data, 215, 223, 225, 226, 228, 229, 231, 232, 233, 236, 238, 239, 242, 244, 245, 247, 248, 249, 250, 252, 258, 262, 263
data_lines_width, 216, 225
derived, 216, 226
description, 216, 226
document_title, 216, 226
draw_line, 216, 226
draw_note, 216, 226
draw_plot_curve, 216, 226
draw_plot_line, 216, 227
example, 223, 226, 227, 231, 232, 234, 236, 237, 238, 239, 240, 244, 247, 249, 252, 253, 254, 261, 262
file, 236
fill, 223, 227, 228, 231, 232, 233, 234, 235, 236
font, 228, 229, 234, 235, 239, 240, 246, 247, 248, 253, 254, 260, 262
greek, 226, 234
grid, 241, 243, 254, 255, 257
image_border_margin, 216, 227
image_border_width, 216, 227
image_x_size, 216, 227
image_y_size, 216, 227
ioflags, 261
label, 237, 239, 240, 241, 242, 246, 247, 248, 251, 252, 253, 254, 255, 256, 261, 263
layout, 215
legend, 216, 223, 227, 228, 229, 230
legend_background_color, 216, 227
legend_border_color, 216, 228
legend_box_fill_on, 216, 228
legend_color, 216, 228
legend_font_family, 216, 228
legend_font_weight, 216, 228
legend_header_font_size, 216, 228
legend_lines, 216, 228, 229
legend_on, 216, 229
legend_outside, 216, 229
legend_place, 216, 229
legend_title, 216, 229
legend_title_font_size, 216, 229
legend_top_left, 216, 229, 230
legend_width, 216, 230
license, 216, 230
license_attribution, 216, 230
license_commercialuse, 216, 230
license_distribution, 216, 230
license_on, 216, 230
license_reproduction, 216, 230
limit_color, 216, 230, 231
limit_fill_color, 216, 231
line, 256
marker, 215, 236, 238, 239, 244, 245, 247, 248, 249
maximum, 233, 236, 242, 245, 250, 256, 258, 259
minimum, 233, 236, 242, 243, 244, 245, 250, 256, 257, 258, 259

one_sd_color, 216, 231
plot, 216, 231, 232, 234
plot_background_color, 216, 232
plot_border_color, 216, 232
plot_border_width, 216, 232
plot_window_on, 216, 232, 233
plot_window_x, 216, 233
plot_window_x_left, 216, 233
plot_window_x_right, 216, 233
plot_window_y, 216, 233
plot_window_y_bottom, 216, 233
plot_window_y_top, 216, 233
position, 237
precision, 225, 247, 248, 249, 261, 262, 263
prefix, 258
rotation, 249
scaling, 233, 239, 246, 249, 250, 253, 260, 262
series, 228, 231, 232, 249, 250
size, 216, 228, 233, 239, 240, 245, 247, 259
stroke, 228, 230, 231, 251
SVG, 215, 223, 225, 226, 227, 228, 229, 230, 232, 233, 235, 236, 239, 240, 242, 244, 245, 246, 247, 253, 259, 260, 262
svg_2d_plot, 216
three_sd_color, 216, 233, 234
title, 216, 223, 226, 228, 229, 231, 232, 234, 235
title_color, 216, 234
title_font_alignment, 216, 234
title_font_decoration, 216, 234
title_font_family, 216, 234
title_font_rotation, 216, 234, 235
title_font_size, 216, 235
title_font_stretch, 216, 235
title_font_style, 216, 235
title_font_weight, 216, 235
title_on, 216, 235
two_sd_color, 216, 235, 236
uncertainty, 258
Unicode, 226, 234, 238, 239, 245, 246, 252, 253, 259, 260, 262
width, 227, 240, 245
write, 216, 236
xy_autoscale, 216, 249
xy_values_on, 216, 249
x_addlimits_color, 216, 236
x_addlimits_on, 216, 236
x_autoscale, 216, 236, 237, 242, 245
x_auto_max_value, 216, 236
x_auto_min_value, 216, 236
x_auto_ticks, 216, 236
x_auto_tick_interval, 216, 236
x_axis, 216, 237
x_axis_color, 216, 237
x_axis_label_color, 216, 237
x_axis_on, 216, 237
x_axis_position, 216, 237
x_axis_vertical, 216, 238
x_axis_width, 216, 238
x_datetime_color, 216, 238
x_datetime_on, 216, 238
x_decor, 216, 238

x_df_color, 216, 238
x_df_on, 216, 238, 239
x_id_color, 216, 239
x_id_on, 216, 239
x_label, 216, 239
x_labels_strip_e0s, 216, 240
x_label_color, 216, 239
x_label_font_family, 216, 239
x_label_font_size, 216, 239, 240
x_label_on, 216, 239, 240
x_label_units, 216, 240
x_label_units_on, 216, 240
x_label_width, 216, 240
x_major_grid_color, 216, 241
x_major_grid_on, 216, 241
x_major_grid_width, 216, 241
x_major_interval, 216, 241
x_major_labels_side, 216, 241, 242
x_major_label_rotation, 216, 241
x_major_tick, 216, 242
x_major_tick_color, 216, 242
x_major_tick_length, 216, 242
x_major_tick_width, 216, 242
x_max, 216, 242
x_min, 216, 242, 243
x_minor_grid_color, 216, 243
x_minor_grid_on, 216, 243
x_minor_grid_width, 216, 243
x_minor_interval, 216, 243
x_minor_tick_color, 216, 243
x_minor_tick_length, 216, 244
x_minor_tick_width, 216, 244
x_min_ticks, 216, 243
x_num_minor_ticks, 216, 244
x_order_color, 216, 244
x_order_on, 216, 244
x_plusminus_color, 216, 244
x_plusminus_on, 216, 245
x_prefix, 216, 245
x_range, 216, 245
x_separator, 216, 245
x_size, 216, 227, 245
x_steps, 216, 245
x_suffix, 216, 246
x_ticks, 216, 246
x_ticks_down_on, 216, 246
x_ticks_on_window_or_axis, 216, 246
x_ticks_up_on, 216, 246
x_ticks_values_color, 216, 246
x_ticks_values_font_family, 216, 246
x_ticks_values_font_size, 216, 247
x_ticks_values_ioflags, 216, 247
x_ticks_values_precision, 216, 247
x_tight, 216, 247
x_values_color, 216, 248
x_values_font_family, 216, 248
x_values_font_size, 216, 248
x_values_ioflags, 216, 248

x_values_on, 216, 248
x_values_precision, 216, 249
x_values_rotation, 216, 249
x_value_font_size, 216, 247
x_value_ioflags, 216, 247, 248
x_value_precision, 216, 248
x_with_zero, 216, 249
y_addlimits_color, 216, 249, 250
y_addlimits_on, 216, 250
y_autoscale, 216, 250
y_axis, 216, 250
y_axis_color, 216, 251
y_axis_label_color, 216, 251
y_axis_on, 216, 251
y_axis_position, 216, 251
y_axis_value_color, 216, 251
y_axis_width, 216, 251, 252
y_decor, 216, 252
y_df_color, 216, 252
y_df_on, 216, 252
y_label, 216, 252
y_labels_strip_e0s, 216, 254
y_label_axis, 216, 252, 253
y_label_color, 216, 253
y_label_font_family, 216, 253
y_label_font_size, 216, 253
y_label_on, 216, 252, 253
y_label_units, 216, 253, 254
y_label_units_on, 216, 254
y_label_weight, 216, 254
y_label_width, 216, 254
y_major_grid_color, 216, 254, 255
y_major_grid_on, 216, 255
y_major_grid_width, 216, 255
y_major_interval, 216, 255
y_major_labels_side, 216, 256
y_major_label_rotation, 216, 255
y_major_tick_color, 216, 256
y_major_tick_length, 216, 256
y_major_tick_width, 216, 256
y_max, 216, 256
y_min, 216, 256
y_minor_grid_color, 216, 257
y_minor_grid_on, 216, 257
y_minor_grid_width, 216, 257
y_minor_interval, 216, 257
y_minor_tick_color, 216, 257
y_minor_tick_length, 216, 257
y_minor_tick_width, 216, 258
y_num_minor_ticks, 216, 258
y_plusminus_color, 216, 258
y_plusminus_on, 216, 258
y_prefix, 216, 258
y_range, 216, 258, 259
y_separator, 216, 259
y_size, 216, 227, 259
y_suffix, 216, 259
y_ticks, 216, 259

y_ticks_left_on, 216, 259
y_ticks_on_window_or_axis, 216, 259
y_ticks_right_on, 216, 260
y_ticks_values_color, 216, 260
y_ticks_values_font_family, 216, 260
y_ticks_values_font_size, 216, 260
y_ticks_values_ioflags, 216, 260, 261
y_ticks_values_precision, 216, 261
y_values_color, 216, 261
y_values_font_family, 216, 262
y_values_font_size, 216, 262
y_values_ioflags, 216, 262
y_values_on, 216, 262
y_values_precision, 216, 262
y_values_rotation, 216, 263
y_value_ioflags, 216, 261
y_value_precision, 216, 261

Class `svg_2d_plot_series`
area_fill, 263, 264
bar_area_fill, 263, 264
bar_color, 263, 264
bar_opt, 263, 264, 265
bar_width, 263, 265
bezier_on, 263, 265
color, 263, 264, 265, 266
container, 264
data, 263, 264, 265, 266
example, 264
fill, 264, 265
fill_color, 263, 265
histogram, 263, 265
legend, 264, 266
limits_count, 263, 265
line, 264, 265
line_color, 263, 265
line_on, 263, 266
line_style, 263, 266
line_width, 263, 266
marker, 264, 265, 266
series, 263, 264, 265, 266
shape, 263, 266
size, 263, 266
stroke, 265, 266
stroke_color, 263, 266
SVG, 263
`svg_2d_plot_series`, 263
title, 264
uncertainty, 264
values_count, 263, 266

Class `svg_boxplot`
autoscale, 268, 274
autoscale_check_limits, 268, 274
autoscale_plusminus, 268, 275
axes_on, 268, 275
axis_color, 268, 275
axis_width, 268, 275
background, 268, 275, 276, 280, 281, 286
background_border_color, 268, 275

background_border_width, 268, 275, 276
background_color, 268, 276
boost_license_on, 268, 276
border, 275, 276, 279, 280, 284, 286, 287, 292, 293, 306
boxplot, 267, 274, 285, 286, 288, 290, 291
box_border, 268, 276
box_fill, 268, 276
box_width, 268, 276
color, 268, 274, 275, 276, 278, 279, 280, 281, 283, 284, 285, 286, 288, 289, 290, 291, 292, 293, 294, 296, 297, 298, 299, 300, 301, 302, 304, 306, 307, 308
confidence, 268, 276, 277
container, 274, 278, 285, 286, 292, 305, 306
coord_precision, 268, 277
copyright_date, 268, 277
copyright_holder, 268, 277
data, 267, 277, 278, 281, 285, 286, 287, 291, 293, 294, 298, 300, 301, 303, 304, 305
data_lines_width, 268, 277
derived, 268, 277
description, 268, 278
document_title, 268, 278
draw_line, 268, 278
draw_note, 268, 278
draw_plot_curve, 268, 278
draw_plot_line, 268, 278
example, 278, 285, 288, 289, 291, 292, 293, 294, 295, 296, 300, 303, 307, 308
extreme_outlier_color, 268, 279
extreme_outlier_fill, 268, 279
extreme_outlier_shape, 268, 279
extreme_outlier_size, 268, 279
extreme_outlier_values_on, 268, 279
fill, 276, 279, 280, 281, 283, 284, 285, 288, 290
font, 281, 282, 289, 290, 294, 295, 302, 303, 304, 307
greek, 278, 288
grid, 296, 298, 299
image_border_margin, 268, 279, 280
image_border_width, 268, 280
image_x_size, 268, 280
image_y_size, 268, 280
label, 292, 294, 295, 296, 297, 302, 303, 304, 306, 307
legend, 268, 280, 281, 282, 283
legend_background_color, 268, 280
legend_border_color, 268, 280
legend_box_fill_on, 268, 281
legend_color, 268, 281
legend_font_family, 268, 281
legend_font_weight, 268, 281
legend_header_font_size, 268, 281
legend_lines, 268, 281
legend_on, 268, 281, 282
legend_outside, 268, 282
legend_place, 268, 282
legend_title, 268, 282
legend_title_font_size, 268, 282
legend_top_left, 268, 282
legend_width, 268, 282, 283
license, 268, 283
license_attribution, 268, 283
license_commercialuse, 268, 283

license_distribution, 268, 283
license_on, 268, 283
license_reproduction, 268, 283
limit_color, 268, 283
limit_fill_color, 268, 283
line, 292, 293, 297, 306
marker, 279, 285, 286, 291, 293, 294, 300, 303, 304, 305
maximum, 285, 286, 287, 291, 298, 300, 305, 306, 308
median_color, 268, 284
median_values_on, 268, 284
median_width, 268, 284
minimum, 285, 286, 287, 291, 298, 299, 300, 305, 306, 308
one_sd_color, 268, 284
outlier, 279, 284, 285, 286
outlier_color, 268, 284
outlier_fill, 268, 284, 285
outlier_shape, 268, 285
outlier_size, 268, 285
outlier_style, 268, 285
outlier_values_on, 268, 285
plot, 268, 285, 286
plot_background_color, 268, 286
plot_border_color, 268, 286
plot_border_width, 268, 287
plot_window_on, 268, 287
plot_window_x, 268, 287
plot_window_x_left, 268, 287
plot_window_x_right, 268, 287
plot_window_y, 268, 287
plot_window_y_bottom, 268, 287
plot_window_y_top, 268, 287
precision, 277, 303, 304, 305
quartile, 268, 286, 288
quartile_definition, 268, 288
rotation, 305
scaling, 287, 294, 302, 305, 306
series, 267, 281, 285, 286
size, 268, 281, 288, 295, 301, 303, 308
stroke, 280, 283
SVG, 267, 275, 276, 277, 278, 279, 280, 281, 282, 283, 287, 288, 289, 290, 291, 294, 295, 298, 299, 301, 302, 303, 308, 309
svg_boxplot, 268
three_sd_color, 268, 288
title, 268, 278, 281, 282, 285, 286, 288, 289, 290, 295
title_color, 268, 288, 289
title_font_alignment, 268, 289
title_font_decoration, 268, 289
title_font_family, 268, 289
title_font_rotation, 268, 289
title_font_size, 268, 289
title_font_stretch, 268, 289
title_font_style, 268, 290
title_font_weight, 268, 290
title_on, 268, 290
title_size, 268, 290
two_sd_color, 268, 290
Unicode, 278, 288, 289, 293, 294, 301, 302
value, 279, 284, 285
whisker_length, 268, 290

width, 280, 288, 296
write, 268, 291
x_addlimits_color, 268, 291
x_addlimits_on, 268, 291
x_autoscale, 268, 291, 292, 298, 301
x_auto_max_value, 268, 291
x_auto_min_value, 268, 291
x_auto_ticks, 268, 291
x_auto_tick_interval, 268, 291
x_axis_color, 268, 292
x_axis_label_color, 268, 292
x_axis_on, 268, 292
x_axis_position, 268, 292
x_axis_vertical, 268, 293
x_axis_width, 268, 293
x_datetime_color, 268, 293
x_datetime_on, 268, 293
x_decor, 268, 293
x_df_color, 268, 293
x_df_on, 268, 293, 294
x_id_color, 268, 294
x_id_on, 268, 294
x_label, 268, 294
x_labels_strip_e0s, 268, 296
x_label_color, 268, 294
x_label_font_family, 268, 294, 295
x_label_font_size, 268, 295
x_label_on, 268, 295
x_label_size, 268, 295
x_label_text, 268, 295
x_label_units, 268, 295
x_label_units_on, 268, 295, 296
x_label_width, 268, 296
x_major_grid_color, 268, 296
x_major_grid_on, 268, 296
x_major_grid_width, 268, 296
x_major_interval, 268, 296
x_major_labels, 268, 297
x_major_labels_on, 268, 297
x_major_labels_side, 268, 297
x_major_label_rotation, 268, 297
x_major_tick, 268, 297
x_major_tick_color, 268, 297
x_major_tick_length, 268, 298
x_major_tick_width, 268, 298
x_max, 268, 298
x_min, 268, 298
x_minor_grid_color, 268, 298
x_minor_grid_on, 268, 299
x_minor_grid_width, 268, 299
x_minor_interval, 268, 299
x_minor_tick_color, 268, 299
x_minor_tick_length, 268, 299
x_minor_tick_width, 268, 299
x_min_ticks, 268, 298
x_num_minor_ticks, 268, 299, 300
x_order_color, 268, 300
x_order_on, 268, 300

x_plusminus_color, 268, 300
x_plusminus_on, 268, 300
x_prefix, 268, 300
x_range, 268, 300
x_separator, 268, 301
x_size, 268, 280, 301
x_steps, 268, 301
x_suffix, 268, 301
x_ticks_down_on, 268, 302
x_ticks_on_window_or_axis, 268, 302
x_ticks_up_on, 268, 302
x_ticks_values_color, 268, 302
x_ticks_values_font_family, 268, 302, 303
x_ticks_values_font_size, 268, 303
x_ticks_values_ioflags, 268, 303
x_ticks_values_precision, 268, 303
x_tick_color, 268, 301
x_tick_length, 268, 301
x_tick_width, 268, 302
x_tight, 268, 303
x_values_color, 268, 304
x_values_font_family, 268, 304
x_values_font_size, 268, 304
x_values_ioflags, 268, 304
x_values_on, 268, 304
x_values_precision, 268, 305
x_values_rotation, 268, 305
x_value_font_size, 268, 303
x_value_ioflags, 268, 303, 304
x_value_precision, 268, 304
x_with_zero, 268, 305
y_autoscale, 268, 305, 306
y_axis_color, 268, 306
y_axis_on, 268, 306
y_axis_position, 268, 306
y_label, 268, 306
y_labels_strip_e0s, 268, 307
y_label_color, 268, 306, 307
y_label_font_size, 268, 307
y_label_on, 268, 306, 307
y_label_text, 268, 307
y_label_units, 268, 307
y_major_interval, 268, 307
y_major_labels_on, 268, 307
y_major_tick_color, 268, 307
y_major_tick_length, 268, 307
y_major_tick_width, 268, 308
y_minor_interval, 268, 308
y_minor_tick_color, 268, 308
y_minor_tick_length, 268, 308
y_minor_tick_width, 268, 308
y_num_minor_ticks, 268, 308
y_range, 268, 308
y_size, 268, 280, 308, 309

Class `svg_boxplot_series`

- axis_color, 309, 310
- axis_width, 309, 310, 311
- border, 311

boxplot, 310, 311
box_border, 309, 311
box_fill, 309, 311
box_style, 309, 311
box_width, 309, 311
calculate_quantiles, 309, 311
color, 309, 310, 311, 312, 313, 314
container, 310
data, 310, 311, 315
extreme_outlier_color, 309, 312
extreme_outlier_fill, 309, 312
extreme_outlier_shape, 309, 312
extreme_outlier_size, 309, 312
fill, 311, 312, 314
marker, 312, 314
maximum, 309, 310, 312, 313, 315
max_whisker_color, 309, 312, 313
max_whisker_width, 309, 313
median_color, 309, 313
median_style, 309, 313
median_width, 309, 313
minimum, 309, 310, 312, 313, 314, 315
min_whisker_color, 309, 313
min_whisker_width, 309, 314
outlier, 311, 312, 314, 315
outlier_color, 309, 314
outlier_fill, 309, 314
outlier_shape, 309, 314
outlier_size, 309, 314
outlier_style, 309, 315
quartile, 309, 310, 311, 315
quartile_definition, 309, 315
scaling, 309, 311
series, 310, 311, 315
SVG, 309
svg_boxplot_series, 309
title, 309, 310, 315
uncertainty, 310
whisker_length, 309, 315

Class svg_color
color, 316, 317
container, 316
data, 316
example, 317
fill, 317
is_blank, 316, 317
my_blank, 317
SVG, 316, 317
svg_color, 316
write, 316, 317

Class svg_element
class_id, 150, 151
clip_id, 150, 151
color, 151, 152
container, 150
example, 150, 151
fill, 151
id, 150, 151

stroke, 150, 151
style, 150, 152
SVG, 149, 150, 151, 152
svg_element, 150
write, 150, 152
write_attributes, 150, 151

Class `svg_style`
border, 334
color, 334, 335, 336
data, 334
example, 334, 336
fill, 334, 335, 336
fill_color, 334, 335
fill_on, 334, 335
font, 334
stroke, 334, 335, 336
stroke_color, 334, 335
stroke_on, 334, 335
stroke_width, 334, 335, 336
SVG, 334, 335, 336
svg_style, 334
title, 334
width, 335
width_on, 334, 336
write, 334, 336

Class template `close_to`
`close_to`, 105
size, 105
strength, 105

Class template `smallest`
minimum, 106
size, 106
smallest, 106

Class template `unc`
`deg_free`, 348, 349
example, 350
maximum, 348
minimum, 350
scaling, 347
SVG, 347
types, 348, 349
unc, 348
uncertainty, 347, 348, 349, 350
Unicode, 350
value, 348, 349

Class `text_element`
alignment, 153, 154
`class_id`, 153, 154
`clip_id`, 153, 155
color, 155, 157
container, 153
example, 153, 154, 155
fill, 157
font, 153, 156
`generate_text`, 153, 154
greek, 153
id, 153, 155
rotation, 153, 155

stroke, 157
style, 153, 155
SVG, 153, 154, 155, 156
svg, 153
svg_element, 153
text, 153, 155, 156
textstyle, 153, 156
text_element, 153
tspan, 153, 156
Unicode, 153
write, 153, 156
write_attributes, 153, 157
x, 153, 156
y, 153, 156

Class text_element_text
 SVG, 157
 svg, 157
 text_element_text, 157
 text_parent, 157
 write, 157

Class text_parent
 SVG, 157, 158
 text_parent, 158
 write, 158

Class text_style
 data, 337
 example, 337, 338, 339
 font, 336, 337, 338, 339
 font_decoration, 337, 338
 font_family, 337, 338
 font_size, 337, 338
 font_stretch, 337, 338
 font_style, 337, 338
 font_weight, 337, 339
 Inkscape, 339
 minimum, 338
 size, 337, 338
 SVG, 336, 337, 338, 339
 text_style, 337
 Unicode, 337, 338

Class ticks_labels_style
 border, 340
 color, 340
 data, 340, 341
 example, 340
 font, 340
 grid, 339, 340
 label, 339, 340, 341
 label_length, 340, 341
 longest_label, 340, 341
 major_value_labels_side, 340, 341
 maximum, 340, 341
 minimum, 340, 341
 precision, 340
 stroke, 340
 style, 340
 SVG, 339, 340
 ticks_labels_style, 340

use_down_ticks, 340, 341
use_up_ticks, 340, 341
value, 340

Class tspan_element
 class_id, 159, 160
 clip_id, 159, 160
 color, 159, 161, 162, 163, 164
 dx, 159, 160
 dy, 159, 160
 example, 160, 161, 162
 fill, 161, 164
 fill_color, 159, 161
 fill_on, 159, 161
 font, 159, 161, 162, 163
 font_family, 159, 161
 font_size, 159, 161
 font_style, 159, 161
 font_weight, 159, 162
 id, 159, 162
 Inkscape, 162
 rotation, 159, 162
 stroke, 164
 style, 159, 162, 163
 SVG, 158, 159, 160, 161, 162, 163, 164
 svg, 159
 svg_element, 159
 text, 159, 163
 textstyle, 159, 163
 text_length, 159, 163
 text_parent, 159
 tspan_element, 159
 use_style, 159, 163
 write, 159, 163
 write_attributes, 159, 164
 x, 159, 163, 164
 y, 159, 164

Class value_style
 color, 342, 343
 data, 342, 343
 draw_plot_point_values, 342
 example, 342
 fill, 342
 font, 342
 label, 342
 line, 342
 minimum, 343
 precision, 342
 series, 342
 stroke, 342
 style, 342
 SVG, 342
 uncertainty, 342, 343
 value_style, 342

class_id
 Class circle_element, 112, 113
 Class clip_path_element, 114, 115
 Class ellipse_element, 116, 117
 Class g_element, 119, 120

```

Class line_element, 126, 127
Class path_element, 129, 130
Class polyline_element, 139, 140
Class curve_element, 143, 144
Class rect_element, 146, 147
Class svg_element, 150, 151
Class text_element, 153, 154
Class tspan_element, 159, 160
Struct polygon_element, 136, 137
clear
    Class g_element, 119, 120
clip_id
    Class circle_element, 112, 113
    Class clip_path_element, 114, 115
    Class ellipse_element, 116, 117
    Class g_element, 119, 120, 121
    Class line_element, 126, 127
    Class path_element, 129, 130
    Class polyline_element, 139, 140
    Class curve_element, 143, 144
    Class rect_element, 146, 147
    Class svg_element, 150, 151
    Class text_element, 153, 155
    Class tspan_element, 159, 160
    Struct polygon_element, 136, 137
clip_path
    Class svg, 172, 174
clip_path_element
    Class clip_path_element, 114
close_to
    Class template close_to, 105
color
    1-D Auto scaling Examples, 31
    1-D Data Values Examples, 43
    1-D Vector Example, 14
    2-D Data Values Examples, 67
    Acknowledgements, 369
    Class axis_line_style, 323, 324
    Class bar_style, 325, 326
    Class box_style, 327, 328
    Class circle_element, 113, 114
    Class clip_path_element, 115, 116
    Class ellipse_element, 118
    Class g_element, 123
    Class line_element, 127, 128
    Class path_element, 132, 133
    Class plot_line_style, 330, 331
    Class plot_point_style, 331, 332, 333
    Class polyline_element, 141
    Class curve_element, 144, 145
    Class rect_element, 148
    Class svg_1d_plot, 179, 185, 187, 188, 189, 191, 192, 193, 194, 195, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 208, 209, 210, 211
    Class svg_1d_plot_series, 212, 213, 214, 215
    Class svg_2d_plot, 216, 223, 224, 226, 227, 228, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 241, 242, 243, 244, 246, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 260, 261
    Class svg_2d_plot_series, 263, 264, 265, 266

```

Class `svg_boxplot`, 268, 274, 275, 276, 278, 279, 280, 281, 283, 284, 285, 286, 288, 289, 290, 291, 292, 293, 294, 296, 297, 298, 299, 300, 301, 302, 304, 306, 307, 308
Class `svg_boxplot_series`, 309, 310, 311, 312, 313, 314
Class `svg_color`, 316, 317
Class `svg_element`, 151, 152
Class `svg_style`, 334, 335, 336
Class `text_element`, 155, 157
Class `ticks_labels_style`, 340
Class `tspan_element`, 159, 161, 162, 163, 164
Class `value_style`, 342, 343
Colors, 6
Demonstration of adding lines and curves, typically a least squares fit, 80
Demonstration of using 1D data that includes information about its Uncertainty, 54
Demonstration of using 2D data that includes information about its uncertainty, 76
Fonts, 8
Function `constant_to_rgb`, 319
Function `is_blank`, 319, 320
Function `operator!=`, 320
Function `operator<<`, 320
Function `operator==`, 321
Global `color_array`, 318, 319
Header < boost/svg_plot/svg_color.hpp >, 315
Header < boost/svg_plot/svg_style.hpp >, 321
Implementation and other notes, 363
Showing data values at Numerical Limits, 92
Simple Code Example, 58, 59
Simple Example, 84
Struct `polygon_element`, 136, 138
Tutorial: 1D Autoscale with Multiple Containers, 18
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 1D More Layout Examples, 20
Tutorial: 2D Special Features, 62
Tutorial: Fuller Layout Example, 60
Type `svg_color_constant`, 317, 318
Using Inkscape, 356
Colors
background, 6
border, 6
color, 6
container, 6
example, 6
maximum, 6
minimum, 6
plot, 6
SVG, 6
commercialuse
Class `svg`, 172, 174
Compilers and Examples
container, 362
example, 362
SVG, 362
uncertainty, 362
confidence
Class `svg_1d_plot`, 179, 186
Class `svg_2d_plot`, 216, 225
Class `svg_boxplot`, 268, 276, 277
constant_to_rgb
Function `constant_to_rgb`, 319

Header < boost/svg_plot/svg_color.hpp >, 315
container
 1-D Auto scaling Examples, 31
 1-D Autoscaling Various Containers Examples, 40
 1-D Axis Scaling, 26
 1-D Data Values Examples, 43
 1-D STL Containers Examples, 15
 1-D Vector Example, 14
 2-D Autoscaling Examples, 73
 2-D Data Values Examples, 67
 Class g_element, 118, 120, 124
 Class path_element, 129
 Class polyline_element, 139
 Class svg_1d_plot, 184, 187, 192, 193, 199
 Class svg_1d_plot_series, 213
 Class svg_2d_plot, 223, 226, 231, 232, 237, 249, 250
 Class svg_2d_plot_series, 264
 Class svg_boxplot, 274, 278, 285, 286, 292, 305, 306
 Class svg_boxplot_series, 310
 Class svg_color, 316
 Class svg_element, 150
 Class text_element, 153
Colors, 6
Compilers and Examples, 362
Definitions of the Quartiles, 86
Demonstration of using 1D data that includes information about its Uncertainty, 54
Demonstration of using 2D data that includes information about its uncertainty, 76
Fonts, 8
Function template mnmx, 95
Function template range_all, 95
Function template range_mx, 96
Function template scale_axis, 99, 100, 101
Function template show, 102
Function template show_all, 102
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
How To Use This Documentation, 5
Implementation and other notes, 363
Preface, 3
Simple Code Example, 58
Struct polygon_element, 136
To Do List, 368
Tutorial: 1D Autoscale with Multiple Containers, 18
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 1D More Layout Examples, 20
Tutorial: 2D Special Features, 62
Using Inkscape, 356
coord_precision
 Class svg, 172, 174
 Class svg_1d_plot, 179, 186, 197
 Class svg_2d_plot, 216, 225
 Class svg_boxplot, 268, 277
copyright_date
 Class svg, 172, 174
 Class svg_1d_plot, 179, 186
 Class svg_2d_plot, 216, 225
 Class svg_boxplot, 268, 277
copyright_holder
 Class svg, 172, 174

Class `svg_1d_plot`, 179, 186
 Class `svg_2d_plot`, 216, 225
 Class `svg_boxplot`, 268, 277
`c_path`
 Struct `c_path`, 111

D

`data`
 1-D Auto scaling Examples, 31
 1-D Autoscaling Various Containers Examples, 40
 1-D Axis Scaling, 26
 1-D Data Values Examples, 43
 1-D STL Containers Examples, 15
 1-D Vector Example, 14
 1D Tutorials, 14
 2-D Autoscaling Examples, 73
 2-D Data Values Examples, 67, 69
 Class `axis_line_style`, 323, 325
 Class `bar_style`, 325
 Class `box_style`, 327
 Class `circle_element`, 112
 Class `clip_path_element`, 114
 Class `ellipse_element`, 116, 117
 Class `g_element`, 119, 124
 Class `histogram_style`, 329
 Class `line_element`, 126
 Class `path_element`, 129
 Class `plot_line_style`, 329, 330, 331
 Class `plot_point_style`, 331, 332, 333
 Class `polyline_element`, 139
 Class `curve_element`, 143
 Class `rect_element`, 146
 Class `svg`, 174, 177
 Class `svg_1d_plot`, 179, 184, 186, 187, 189, 192, 193, 194, 197, 198, 200, 201, 204, 206, 207, 208, 209, 210, 211
 Class `svg_1d_plot_series`, 212, 213, 214
 Class `svg_2d_plot`, 215, 223, 225, 226, 228, 229, 231, 232, 233, 236, 238, 239, 242, 244, 245, 247, 248, 249, 250, 252, 258, 262, 263
 Class `svg_2d_plot_series`, 263, 264, 265, 266
 Class `svg_boxplot`, 267, 277, 278, 281, 285, 286, 287, 291, 293, 294, 298, 300, 301, 303, 304, 305
 Class `svg_boxplot_series`, 310, 311, 315
 Class `svg_color`, 316
 Class `svg_style`, 334
 Class `text_style`, 337
 Class `ticks_labels_style`, 340, 341
 Class `value_style`, 342, 343
 Definitions of the Quartiles, 86, 87
 Demonstration of adding lines and curves, typically a least squares fit, 80
 Demonstration of using 1D data that includes information about its Uncertainty, 54
 Demonstration of using 2D data that includes information about its uncertainty, 76
 Fonts, 8
 Function `median`, 168
 Function `operator<<`, 345
 Function `quantile`, 169
 Function template `range_mx`, 96
 Function template `scale_axis`, 99, 100, 101
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 94
 Header <`boost/svg_plot/detail/functors.hpp`>, 107

Header < boost/svg_plot/detail/numeric_limits_handling.hpp >, 107
Header < boost/svg_plot/quantile.hpp >, 168
Header < boost/svg_plot/svg_1d_plot.hpp >, 178
Header < boost/svg_plot/svg_2d_plot.hpp >, 215
Header < boost/svg_plot/svg_boxplot.hpp >, 267
Header < boost/svg_plot/svg_style.hpp >, 321
Histograms of 2D data, 82
Implementation and other notes, 363, 364
Preface, 3
Real-life Heat flow data, 48
Showing data values at Numerical Limits, 92
Simple Code Example, 58
Simple Example, 84
Struct a_path, 110
Struct c_path, 111
Struct h_path, 124
Struct l_path, 125
Struct m_path, 128
Struct path_point, 134
Struct polygon_element, 136
Struct poly_path_point, 135
Struct q_path, 142
Struct s_path, 149
Struct t_path, 152
Struct v_path, 164
Struct z_path, 165
To Do List, 368
Tutorial: 1D Autoscale with Multiple Containers, 18
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 1D More Layout Examples, 20
Tutorial: 2D Special Features, 62
Tutorial: Fuller Layout Example, 60
data_lines_width
 Class svg_1d_plot, 179, 186
 Class svg_2d_plot, 216, 225
 Class svg_boxplot, 268, 277
default_font
 Function default_font, 344
 Header < boost/svg_plot/svg_style.hpp >, 321
default_plot_point_style
 Header < boost/svg_plot/svg_style.hpp >, 321
Definitions of the Quartiles
 boxplot, 86
 container, 86
 data, 86, 87
 example, 86
 label, 86
 maximum, 86
 median, 87
 minimum, 86, 87
 quantile, 86
 quartile, 86, 87
 scaling, 86, 87
 series, 86
 SVG, 86
 title, 86
deg_free
 Class template unc, 348, 349

Demonstration of adding lines and curves, typically a least squares fit

- color, 80
- data, 80
- example, 80
- line, 80
- origin, 80
- scaling, 80
- SVG, 80

Demonstration of using 1D data that includes information about its Uncertainty

- background, 54
- border, 54
- color, 54
- container, 54
- data, 54
- fill, 54
- label, 54
- legend, 54
- marker, 54
- precision, 54
- series, 54
- SVG, 54
- title, 54
- uncertainty, 54
- uncun, 54
- width, 54

Demonstration of using 2D data that includes information about its uncertainty

- autoscale, 76
- autoscale_check_limits, 76
- autoscale_plusminus, 76
- border, 76
- color, 76
- container, 76
- data, 76
- example, 76
- label, 76
- series, 76
- SVG, 76
- u1, 76
- u2, 76
- uncertainty, 76
- uncun, 76
- Unicode, 76

deque

- 1-D STL Containers Examples, 15
- Tutorial: 1D More Layout Examples, 20

derived

- Class svg_1d_plot, 179, 186, 187
- Class svg_2d_plot, 216, 226
- Class svg_boxplot, 268, 277

description

- Class svg, 172, 175
- Class svg_1d_plot, 179, 187
- Class svg_2d_plot, 216, 226
- Class svg_boxplot, 268, 278

Implementation and other notes, 363

distribution

- Class svg, 172, 175

document_size

Class `svg`, 172, 175
document_title
 Class `svg`, 172, 175
 Class `svg_1d_plot`, 179, 187
 Class `svg_2d_plot`, 216, 226
 Class `svg_boxplot`, 268, 278
Implementation and other notes, 363
double
 Class `svg_1d_plot`, 193
draw_line
 Class `svg_1d_plot`, 179, 187
 Class `svg_2d_plot`, 216, 226
 Class `svg_boxplot`, 268, 278
draw_note
 Class `svg_1d_plot`, 179, 187
 Class `svg_2d_plot`, 216, 226
 Class `svg_boxplot`, 268, 278
draw_plot_curve
 Class `svg_1d_plot`, 179, 187
 Class `svg_2d_plot`, 216, 226
 Class `svg_boxplot`, 268, 278
draw_plot_line
 Class `svg_1d_plot`, 179, 187
 Class `svg_2d_plot`, 216, 227
 Class `svg_boxplot`, 268, 278
draw_plot_point_values
 Class `value_style`, 342
dx
 Class `tspan_element`, 159, 160
dy
 Class `tspan_element`, 159, 160

E

ellipse
 Class `g_element`, 119, 121
 Class `svg`, 172, 175
ellipse_element
 Class `ellipse_element`, 116
epsilon
 Header < boost/svg_plot/detail/fp_compare.hpp >, 104
example
 1-D Auto scaling Examples, 31
 1-D Autoscaling Various Containers Examples, 40
 1-D Axis Scaling, 26
 1-D Data Values Examples, 43
 1-D STL Containers Examples, 15
 1-D Vector Example, 14
 1D Tutorials, 14
 2-D Autoscaling Examples, 73
 2-D Data Values Examples, 67
 Acknowledgements, 369
 Class `axis_line_style`, 323, 324
 Class `circle_element`, 112, 113
 Class `clip_path_element`, 114, 115
 Class `ellipse_element`, 117, 118
 Class `g_element`, 120, 121, 123
 Class `line_element`, 126, 127

Class path_element, 130, 131, 133
Class plot_point_style, 332
Class polyline_element, 140, 141
Class curve_element, 144, 145
Class rect_element, 147, 148
Class svg, 174
Class svg_1d_plot, 187, 192, 193, 195, 196, 197, 198, 199, 200, 201, 202, 206, 209, 212
Class svg_2d_plot, 223, 226, 227, 231, 232, 234, 236, 237, 238, 239, 240, 244, 247, 249, 252, 253, 254, 261, 262
Class svg_2d_plot_series, 264
Class svg_boxplot, 278, 285, 288, 289, 291, 292, 293, 294, 295, 296, 300, 303, 307, 308
Class svg_color, 317
Class svg_element, 150, 151
Class svg_style, 334, 336
Class template_unc, 350
Class text_element, 153, 154, 155
Class text_style, 337, 338, 339
Class ticks_labels_style, 340
Class tspan_element, 160, 161, 162
Class value_style, 342
Colors, 6
Compilers and Examples, 362
Definitions of the Quartiles, 86
Demonstration of adding lines and curves, typically a least squares fit, 80
Demonstration of using 2D data that includes information about its uncertainty, 76
Fonts, 8
Function constant_to_rgb, 319
Function operator<<, 166, 167, 320, 345
Function outFmtFlags, 170
Function quantile, 169
Function template operator<<, 351
Global package_info, 178
Header < boost/svg_plot/detail/pair.hpp >, 108
How To Use This Documentation, 5
Implementation and other notes, 363
Preface, 3
Real-life Heat flow data, 48
Showing data values at Numerical Limits, 92
Simple Code Example, 58
Simple Example, 84
Struct m_path, 128
Struct polygon_element, 137, 138
Struct poly_path_point, 135
SVG tutorial, 91
To Do List, 368
Tutorial: 1D Autoscale with Multiple Containers, 18
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 1D More Layout Examples, 20
Tutorial: 2D Special Features, 62
Tutorial: Fuller Layout Example, 60
Using Inkscape, 356
extreme_outlier_color
 Class svg_boxplot, 268, 279
 Class svg_boxplot_series, 309, 312
extreme_outlier_fill
 Class svg_boxplot, 268, 279
 Class svg_boxplot_series, 309, 312
extreme_outlier_shape
 Class svg_boxplot, 268, 279

Class `svg_boxplot_series`, 309, 312

`extreme_outlier_size`

Class `svg_boxplot`, 268, 279

Class `svg_boxplot_series`, 309, 312

`extreme_outlier_values_on`

Class `svg_boxplot`, 268, 279

F

`f`

Simple Example, 84

Tutorial: 1D More Layout Examples, 20

Tutorial: Fuller Layout Example, 60

`file`

Class `svg_2d_plot`, 236

Using Inkscape, 356

`fill`

1-D Auto scaling Examples, 31

1-D Autoscaling Various Containers Examples, 40

1-D Axis Scaling, 26

1-D Data Values Examples, 43

2-D Data Values Examples, 67

Class `bar_style`, 325, 326

Class `box_style`, 327, 328

Class `circle_element`, 114

Class `clip_path_element`, 116

Class `ellipse_element`, 118

Class `g_element`, 120, 123

Class `histogram_style`, 329

Class `line_element`, 128

Class `path_element`, 130, 131, 133

Class `plot_line_style`, 330

Class `plot_point_style`, 332, 333

Class `polyline_element`, 141

Class `curve_element`, 145

Class `rect_element`, 148

Class `svg_1d_plot`, 188, 189, 191, 192, 193, 194, 195, 197

Class `svg_1d_plot_series`, 213

Class `svg_2d_plot`, 223, 227, 228, 231, 232, 233, 234, 235, 236

Class `svg_2d_plot_series`, 264, 265

Class `svg_boxplot`, 276, 279, 280, 281, 283, 284, 285, 288, 290

Class `svg_boxplot_series`, 311, 312, 314

Class `svg_color`, 317

Class `svg_element`, 151

Class `svg_style`, 334, 335, 336

Class `text_element`, 157

Class `tspan_element`, 161, 164

Class `value_style`, 342

Demonstration of using 1D data that includes information about its Uncertainty, 54

Function operator `<<`, 344

Header < boost/svg_plot/svg_style.hpp >, 321

Implementation and other notes, 363

Showing data values at Numerical Limits, 92

Simple Example, 84

Struct `polygon_element`, 136, 138

Tutorial: 1D More Layout Examples, 20

Tutorial: 2D Special Features, 62

`fill_color`

Class `plot_point_style`, 332, 333
Class `svg_1d_plot_series`, 212, 213
Class `svg_2d_plot_series`, 263, 265
Class `svg_style`, 334, 335
Class `tspan_element`, 159, 161
`fill_on`
 Class `box_style`, 327, 328
 Class `path_element`, 129, 130, 131
 Class `svg_style`, 334, 335
 Class `tspan_element`, 159, 161
`font`
 1-D Data Values Examples, 43
 2-D Data Values Examples, 67
 Class `plot_point_style`, 332, 333
 Class `svg_1d_plot`, 189, 190, 196, 197, 201, 208, 209, 210
 Class `svg_2d_plot`, 228, 229, 234, 235, 239, 240, 246, 247, 248, 253, 254, 260, 262
 Class `svg_boxplot`, 281, 282, 289, 290, 294, 295, 302, 303, 304, 307
 Class `svg_style`, 334
 Class `text_element`, 153, 156
 Class `text_style`, 336, 337, 338, 339
 Class `ticks_labels_style`, 340
 Class `tspan_element`, 159, 161, 162, 163
 Class `value_style`, 342
Fonts, 8
Function `default_font`, 344
Function `string_svg_length`, 346
Global reducer, 103
Global wh, 343
Header <`boost/svg_plot/detail/axis_plot_frame.hpp`>, 102
Header <`boost/svg_plot/detail/svg_tag.hpp`>, 108
Header <`boost/svg_plot/svg_style.hpp`>, 321
Implementation and other notes, 363
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 2D Special Features, 62
Using Inkscape, 356
Fonts
 color, 8
 container, 8
 data, 8
 example, 8
 font, 8
 greek, 8
 Inkscape, 8
 label, 8
 legend, 8
 main, 8
 minimum, 8
 precision, 8
 scaling, 8
 SVG, 8
 title, 8
 Unicode, 8
 `font_decoration`
 Class `text_style`, 337, 338
 `font_family`
 Class `text_style`, 337, 338
 Class `tspan_element`, 159, 161
 `font_size`

Class `text_style`, 337, 338
 Class `tspan_element`, 159, 161

`font_stretch`
 Class `text_style`, 337, 338
 Class `tspan_element`, 159, 161

`font_style`
 Class `text_style`, 337, 338
 Class `tspan_element`, 159, 161

`font_weight`
 Class `text_style`, 337, 339
 Class `tspan_element`, 159, 162

`fpt_abs`
 Function template `fpt_abs`, 107
 Header <`boost/svg_plot/detail/fp_compare.hpp`>, 104

Function `constant_to_rgb`
 `color`, 319
 `constant_to_rgb`, 319
 `example`, 319
 `SVG`, 319
 `svg_color`, 319

Function `default_font`
 `default_font`, 344
 `font`, 344
 `legend`, 344
 `SVG`, 343
 `title`, 344
 `Unicode`, 344

Function `is_blank`
 `color`, 319, 320
 `is_blank`, 319
 `SVG`, 319

Function `median`
 `data`, 168
 `maximum`, 168
 `median`, 168
 `minimum`, 168
 `SVG`, 168

Function `operator!=`
 `color`, 320
 `SVG`, 165, 166, 320, 344

Function `operator<<`
 `color`, 320
 `data`, 345
 `example`, 166, 167, 320, 345
 `fill`, 344
 `info`, 167
 `marker`, 345
 `my_color`, 320
 `p`, 167
 `p0`, 167
 `r`, 166
 `stroke`, 344
 `style`, 166
 `SVG`, 166, 167, 320, 344, 345
 `t`, 166
 `ts`, 345

Function `operator==`
 `color`, 321
 `SVG`, 167, 320, 345

Function `outFmtFlags`

`example`, 170
 `outFmtFlags`, 170
 `SVG`, 170

Function `quantile`

`boxplot`, 169
 `data`, 169
 `example`, 169
 `maximum`, 169
 `minimum`, 169
 `quantile`, 169
 `quartile`, 169
 `SVG`, 168

Function `rounddown10`

`rounddown10`, 96
 `SVG`, 96

Function `rounddown2`

`rounddown2`, 96
 `SVG`, 96

Function `rounddown5`

`rounddown5`, 97
 `SVG`, 97

Function `roundup10`

`roundup10`, 97
 `SVG`, 97

Function `roundup2`

`roundup2`, 97
 `SVG`, 97

Function `roundup5`

`roundup5`, 98
 `SVG`, 98

Function `scale_axis`

`marker`, 99
 `maximum`, 98, 99
 `minimum`, 98, 99
 `origin`, 98, 99
 `scale_axis`, 98, 99
 `SVG`, 98
 `uncertainty`, 99

Function `show_1d_plot_settings`

`show_1d_plot_settings`, 170
 `SVG`, 170

Function `show_2d_plot_settings`

`show_2d_plot_settings`, 171
 `SVG`, 171

Function `string_svg_length`

`font`, 346
 `string_svg_length`, 346
 `SVG`, 346

Function `strip_e0s`

`strip_e0s`, 346
 `SVG`, 346

Function template `fpt_abs`

`fpt_abs`, 107

Function template `isfinite`

`isfinite`, 95

Function template `mnmx`

`container`, 95

maximum, 95
minimum, 95
SVG, 95
Function template operator<<
example, 351
minimum, 351
SVG, 350, 351
uncertainty, 351
Unicode, 351
Function template range_all
container, 95
maximum, 95
minimum, 95
SVG, 95
Function template range_mx
container, 96
data, 96
maximum, 96
minimum, 96
series, 96
SVG, 96
Function template safe_fpt_division
safe_fpt_division, 107
Function template scale_axis
container, 99, 100, 101
data, 99, 100, 101
maximum, 99, 100, 101
minimum, 99, 100, 101
origin, 99, 100, 101
plot, 100
scale_axis, 99, 100, 101
scaling, 99, 100
series, 99, 100, 101
SVG, 99, 100
uncertainty, 99, 100, 101
Function template show
container, 102
show, 102
SVG, 101, 102
Function template show_all
container, 102
show_all, 102
SVG, 102
Function template uncs_of
SVG, 352, 353
uncertainty, 352, 353
Function template unc_of
SVG, 351, 352
uncertainty, 351, 352
unc_of, 351, 352
Function template values_of
SVG, 354, 355
uncertainty, 354, 355
Function template value_of
SVG, 353, 354
uncertainty, 353, 354
value_of, 353, 354

G**g**

- Class `g_element`, 119, 121
- Class `svg`, 172, 175
- Simple Example, 84
- Tutorial: 1D More Layout Examples, 20
- Tutorial: Fuller Layout Example, 60

generate_text

- Class `text_element`, 153, 154

Global color_array

- color, 318, 319
 - SVG, 318

Global document_ids_

- boxplot, 108
 - SVG, 108

Global fmtFlagWords

- SVG, 169

Global no_style

- SVG, 343

Global package_info

- example, 178

- SVG, 178

- title, 178

Global plusminus

- SVG, 350

- uncertainty, 350

Global reducer

- font, 103

- SVG, 103

- uncertainty, 103

Global sin45

- label, 103

- scaling, 103

- SVG, 103

Global text_plusminus

- SVG, 103

Global wh

- font, 343

- SVG, 343

greek

- Class `svg_1d_plot`, 187, 195

- Class `svg_2d_plot`, 226, 234

- Class `svg_boxplot`, 278, 288

- Class `text_element`, 153

- Fonts, 8

- Implementation and other notes, 363

grid

- Class `svg_1d_plot`, 202, 203, 205

- Class `svg_2d_plot`, 241, 243, 254, 255, 257

- Class `svg_boxplot`, 296, 298, 299

- Class `ticks_labels_style`, 339, 340

- Tutorial: 1D Gridlines & Axes - more examples, 22

- Tutorial: 2D Special Features, 62

g_element

- Class `g_element`, 119

H

h

Class path_element, 129, 131
Tutorial: 1D More Layout Examples, 20
Tutorial: Fuller Layout Example, 60

H

Class path_element, 129, 131
Header < boost/svg_plot/detail/auto_axes.hpp >
container, 94
data, 94
isfinite, 94
maximum, 94
minimum, 94
rounddown10, 94
rounddown2, 94
rounddown5, 94
roundup10, 94
roundup2, 94
roundup5, 94
scale_axis, 94
show, 94
show_all, 94
SVG, 94

Header < boost/svg_plot/detail/axis_plot_frame.hpp >
boxplot, 102
font, 102
label, 102
legend, 102
scaling, 102
SVG, 102
uncertainty, 102

Header < boost/svg_plot/detail/fp_compare.hpp >
epsilon, 104
fpt_abs, 104
maximum, 104
max_value, 104
minimum, 104
min_value, 104
neareq, 104
safe_fpt_division, 104
tiny, 104

Header < boost/svg_plot/detail/functors.hpp >
data, 107
precision, 107
SVG, 107
uncertainty, 107

Header < boost/svg_plot/detail/numeric_limits_handling.hpp >
data, 107
maximum, 107
minimum, 107

Header < boost/svg_plot/detail/pair.hpp >
example, 108

Header < boost/svg_plot/detail/svg_boxplot_detail.hpp >
boxplot, 108
set_ids, 108
SVG, 108

Header < boost/svg_plot/detail/svg_tag.hpp >

font, 108
SVG, 108
Header < boost/svg_plot/quantile.hpp >
 data, 168
 median, 168
 minimum, 168
 quantile, 168
 quartile, 168
 scaling, 168
 SVG, 168
Header < boost/svg_plot/show_1d_settings.hpp >
 l_or_r, 169
 outFmtFlags, 169
 show_1d_plot_settings, 169
 SVG, 169
 t_or_b, 169
Header < boost/svg_plot/show_2d_settings.hpp >
 show_2d_plot_settings, 170
 SVG, 170
Header < boost/svg_plot/svg.hpp >
 boxplot, 171
 SVG, 171
Header < boost/svg_plot/svg_1d_plot.hpp >
 data, 178
 layout, 178
 marker, 178
 SVG, 178
Header < boost/svg_plot/svg_2d_plot.hpp >
 data, 215
 layout, 215
 marker, 215
 SVG, 215
Header < boost/svg_plot/svg_boxplot.hpp >
 boxplot, 267
 data, 267
 layout, 267
 marker, 267
 quartile, 267
 SVG, 267
Header < boost/svg_plot/svg_color.hpp >
 color, 315
 constant_to_rgb, 315
 is_blank, 315
 SVG, 315
Header < boost/svg_plot/svg_style.hpp >
 color, 321
 data, 321
 default_font, 321
 default_plot_point_style, 321
 fill, 321
 font, 321
 label, 321
 string_svg_length, 321
 strip_e0s, 321
 stroke, 321
 SVG, 321
 text, 321
 Unicode, 321

Header < boost/svg_plot/uncertain.hpp >

minimum, 346

SVG, 346

uncertainty, 346

unc_of, 346

value_of, 346

hexagon

Class g_element, 119, 121

Class svg, 172, 175

histogram

Class bar_style, 325

Class histogram_style, 328, 329

Class svg_2d_plot_series, 263, 265

Histograms of 2D data

data, 82

histogram_style

Class histogram_style, 329

History

uncertainty, 362

How To Use This Documentation

container, 5

example, 5

h_path

Struct h_path, 124

|

id

Class circle_element, 112, 113

Class clip_path_element, 114, 115

Class ellipse_element, 116, 117

Class g_element, 119, 121

Class line_element, 126, 127

Class path_element, 129, 131

Class polyline_element, 139, 140

Class curve_element, 143, 144

Class rect_element, 146, 147

Class svg_element, 150, 151

Class text_element, 153, 155

Class tspan_element, 159, 162

Struct polygon_element, 136, 137, 138

image_border_margin

Class svg_1d_plot, 179, 188

Class svg_2d_plot, 216, 227

Class svg_boxplot, 268, 279, 280

image_border_width

1-D Auto scaling Examples, 31

Class svg_1d_plot, 179, 188

Class svg_2d_plot, 216, 227

Class svg_boxplot, 268, 280

image_filename

Class svg, 172, 175

image_x_size

Class svg_1d_plot, 179, 188

Class svg_2d_plot, 216, 227

Class svg_boxplot, 268, 280

image_y_size

Class svg_1d_plot, 179, 188

Class `svg_2d_plot`, 216, 227

Class `svg_boxplot`, 268, 280

Implementation and other notes

author, 363

color, 363

container, 363

data, 363, 364

description, 363

document_title, 363

example, 363

fill, 363

font, 363

greek, 363

is_license, 363

label, 363

legend, 363

license_attribution, 363

license_commercialuse, 363

license_distribution, 363

license_reproduction, 363

marker, 363

minimum, 363

precision, 363, 364

scaling, 363

series, 363

stroke, 363

style, 363

SVG, 363

title, 363

Unicode, 363

x_num_minor_ticks, 363

info

Class `polyline_element`, 141

Function `operator<<`, 167

Inkscape

Class `text_style`, 339

Class `tspan_element`, 162

Fonts, 8

Preface, 3

To Do List, 368

Using Inkscape, 356

ioflags

1-D Data Values Examples, 43

2-D Data Values Examples, 67

Class `svg_2d_plot`, 261

Tutorial: 2D Special Features, 62

is

Class `svg`, 177

isfinite

Function template `isfinite`, 95

Header <`boost/svg_plot/detail/auto_axes.hpp`>, 94

isnan

Showing data values at Numerical Limits, 92

is_blank

Class `svg_color`, 316, 317

Function `is_blank`, 319

Header <`boost/svg_plot/svg_color.hpp`>, 315

is_license

Implementation and other notes, 363

iter

2-D Autoscaling Examples, 73

L

1

Class path_element, 129, 131

L

Class path_element, 129, 131

label

1-D Autoscaling Various Containers Examples, 40

1-D Data Values Examples, 43

2-D Data Values Examples, 67

Class axis_line_style, 323, 324

Class svg_1d_plot, 199, 201, 202, 203, 208, 209, 211, 212

Class svg_2d_plot, 237, 239, 240, 241, 242, 246, 247, 248, 251, 252, 253, 254, 255, 256, 261, 263

Class svg_boxplot, 292, 294, 295, 296, 297, 302, 303, 304, 306, 307

Class ticks_labels_style, 339, 340, 341

Class value_style, 342

Definitions of the Quartiles, 86

Demonstration of using 1D data that includes information about its Uncertainty, 54

Demonstration of using 2D data that includes information about its uncertainty, 76

Fonts, 8

Global sin45, 103

Header < boost/svg_plot/detail/axis_plot_frame.hpp >, 102

Header < boost/svg_plot/svg_style.hpp >, 321

Implementation and other notes, 363

Real-life Heat flow data, 48

Simple Example, 84

To Do List, 368

Tutorial: 1D Gridlines & Axes - more examples, 22

Tutorial: 1D More Layout Examples, 20

Tutorial: 2D Special Features, 62

label_length

Class ticks_labels_style, 340, 341

label_on

Class axis_line_style, 323, 324

label_units_on

Class axis_line_style, 323, 324

layout

Class svg_2d_plot, 215

Header < boost/svg_plot/svg_1d_plot.hpp >, 178

Header < boost/svg_plot/svg_2d_plot.hpp >, 215

Header < boost/svg_plot/svg_boxplot.hpp >, 267

Tutorial: 1D Gridlines & Axes - more examples, 22

Tutorial: 1D More Layout Examples, 20

Tutorial: Fuller Layout Example, 60

legend

1-D Auto scaling Examples, 31

1-D Vector Example, 14

Class g_element, 123

Class svg_1d_plot, 179, 188, 189, 190, 191

Class svg_1d_plot_series, 213

Class svg_2d_plot, 216, 223, 227, 228, 229, 230

Class svg_2d_plot_series, 264, 266

Class svg_boxplot, 268, 280, 281, 282, 283

Demonstration of using 1D data that includes information about its Uncertainty, 54

Fonts, 8
Function `default_font`, 344
Header <`boost/svg_plot/detail/axis_plot_frame.hpp`>, 102
Implementation and other notes, 363
Simple Code Example, 58, 59
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 1D More Layout Examples, 20
Tutorial: Fuller Layout Example, 60

`legend_background_color`
 Class `svg_1d_plot`, 179, 188
 Class `svg_2d_plot`, 216, 227
 Class `svg_boxplot`, 268, 280

`legend_border_color`
 Class `svg_1d_plot`, 179, 188
 Class `svg_2d_plot`, 216, 228
 Class `svg_boxplot`, 268, 280

`legend_box_fill_on`
 Class `svg_1d_plot`, 179, 189
 Class `svg_2d_plot`, 216, 228
 Class `svg_boxplot`, 268, 281

`legend_color`
 Class `svg_1d_plot`, 179, 189
 Class `svg_2d_plot`, 216, 228
 Class `svg_boxplot`, 268, 281

`legend_font_family`
 Class `svg_1d_plot`, 179, 189
 Class `svg_2d_plot`, 216, 228
 Class `svg_boxplot`, 268, 281

`legend_font_weight`
 Class `svg_1d_plot`, 179, 189
 Class `svg_2d_plot`, 216, 228
 Class `svg_boxplot`, 268, 281

`legend_header_font_size`
 Class `svg_1d_plot`, 179, 189
 Class `svg_2d_plot`, 216, 228
 Class `svg_boxplot`, 268, 281

`legend_lines`
 Class `svg_1d_plot`, 179, 189
 Class `svg_2d_plot`, 216, 228, 229
 Class `svg_boxplot`, 268, 281

`legend_on`
 Class `svg_1d_plot`, 179, 190
 Class `svg_2d_plot`, 216, 229
 Class `svg_boxplot`, 268, 281, 282

`legend_outside`
 Class `svg_1d_plot`, 179, 190
 Class `svg_2d_plot`, 216, 229
 Class `svg_boxplot`, 268, 282

`legend_place`
 Class `svg_1d_plot`, 179, 190
 Class `svg_2d_plot`, 216, 229
 Class `svg_boxplot`, 268, 282

`legend_title`
 Class `svg_1d_plot`, 179, 190
 Class `svg_2d_plot`, 216, 229
 Class `svg_boxplot`, 268, 282

`legend_title_font_size`
 Class `svg_1d_plot`, 179, 190

Class `svg_2d_plot`, 216, 229
Class `svg_boxplot`, 268, 282
`legend_top_left`
 Class `svg_1d_plot`, 179, 190
 Class `svg_2d_plot`, 216, 229, 230
 Class `svg_boxplot`, 268, 282
`legend_width`
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 282, 283
`license`
 Class `svg`, 172, 176
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 283
`license_attribution`
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 283
 Implementation and other notes, 363
`license_commercialuse`
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 283
 Implementation and other notes, 363
`license_distribution`
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 283
 Implementation and other notes, 363
`license_on`
 Class `svg`, 172, 176
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 283
`license_reproduction`
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230
 Class `svg_boxplot`, 268, 283
 Implementation and other notes, 363
`limits_count`
 Class `svg_2d_plot_series`, 263, 265
`limit_color`
 Class `svg_1d_plot`, 179, 191
 Class `svg_2d_plot`, 216, 230, 231
 Class `svg_boxplot`, 268, 283
`limit_fill_color`
 Class `svg_1d_plot`, 179, 191, 192
 Class `svg_2d_plot`, 216, 231
 Class `svg_boxplot`, 268, 283
`limit_point_color`
 Class `svg_1d_plot_series`, 212, 213
`line`
 Class `axis_line_style`, 323
 Class `bar_style`, 325
 Class `box_style`, 327
 Class `g_element`, 119, 121
 Class `plot_line_style`, 330, 331

Class `svg`, 172, 176
Class `svg_1d_plot`, 203
Class `svg_2d_plot`, 256
Class `svg_2d_plot_series`, 264, 265
Class `svg_boxplot`, 292, 293, 297, 306
Class `value_style`, 342
Demonstration of adding lines and curves, typically a least squares fit, 80
`line_color`
 Class `svg_1d_plot_series`, 212, 213
 Class `svg_2d_plot_series`, 263, 265
`line_element`
 Class `line_element`, 126
`line_on`
 Class `plot_line_style`, 330, 331
 Class `svg_1d_plot_series`, 212, 214
 Class `svg_2d_plot_series`, 263, 266
`line_style`
 Class `svg_2d_plot_series`, 263, 266
`line_width`
 Class `svg_1d_plot_series`, 212, 214
 Class `svg_2d_plot_series`, 263, 266
`longest_label`
 Class `ticks_labels_style`, 340, 341
`l_or_r`
 Header <`boost/svg_plot/show_1d_settings.hpp`>, 169
`l_path`
 Struct `l_path`, 125

M

`m`
 Class `path_element`, 129, 131
`M`
 Class `path_element`, 129, 132
`main`
 Fonts, 8
 Real-life Heat flow data, 48
 Simple Code Example, 58
 Simple Example, 84
 Tutorial: 1D Gridlines & Axes - more examples, 22
 Tutorial: 1D More Layout Examples, 20
 Tutorial: Fuller Layout Example, 60
`major_value_labels_side`
 Class `ticks_labels_style`, 340, 341
`margin`
 Class `box_style`, 327, 328
`marker`
 1-D Data Values Examples, 43
 1-D Vector Example, 14
 2-D Data Values Examples, 67
 Class `plot_point_style`, 331
 Class `svg_1d_plot`, 198, 200, 201, 206, 208, 209, 210, 211
 Class `svg_1d_plot_series`, 213, 214, 215
 Class `svg_2d_plot`, 215, 236, 238, 239, 244, 245, 247, 248, 249
 Class `svg_2d_plot_series`, 264, 265, 266
 Class `svg_boxplot`, 279, 285, 286, 291, 293, 294, 300, 303, 304, 305
 Class `svg_boxplot_series`, 312, 314
 Demonstration of using 1D data that includes information about its Uncertainty, 54

Function operator<<, 345
Function scale_axis, 99
Header < boost/svg_plot/svg_1d_plot.hpp >, 178
Header < boost/svg_plot/svg_2d_plot.hpp >, 215
Header < boost/svg_plot/svg_boxplot.hpp >, 267
Implementation and other notes, 363
Simple Code Example, 58
Tutorial: Fuller Layout Example, 60

maximum
 1-D Auto scaling Examples, 31, 36
 1-D Autoscaling Various Containers Examples, 40
 1-D Axis Scaling, 26
 2-D Autoscaling Examples, 73
 2-D Data Values Examples, 67
 Class axis_line_style, 323
 Class svg_1d_plot, 194, 198, 204, 207
 Class svg_2d_plot, 233, 236, 242, 245, 250, 256, 258, 259
 Class svg_boxplot, 285, 286, 287, 291, 298, 300, 305, 306, 308
 Class svg_boxplot_series, 309, 310, 312, 313, 315
 Class template unc, 348
 Class ticks_labels_style, 340, 341
Colors, 6
Definitions of the Quartiles, 86
Function median, 168
Function quantile, 169
Function scale_axis, 98, 99
Function template mnnmx, 95
Function template range_all, 95
Function template range_mx, 96
Function template scale_axis, 99, 100, 101
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
Header < boost/svg_plot/detail/fp_compare.hpp >, 104
Header < boost/svg_plot/detail/numeric_limits_handling.hpp >, 107
Showing data values at Numerical Limits, 92
Tutorial: 1D Autoscale with Multiple Containers, 18

max_value
 Header < boost/svg_plot/detail/fp_compare.hpp >, 104

max_whisker_color
 Class svg_boxplot_series, 309, 312, 313

max_whisker_width
 Class svg_boxplot_series, 309, 313

median
 Definitions of the Quartiles, 87
 Function median, 168
 Header < boost/svg_plot/quantile.hpp >, 168

median_color
 Class svg_boxplot, 268, 284
 Class svg_boxplot_series, 309, 313

median_style
 Class svg_boxplot_series, 309, 313

median_values_on
 Class svg_boxplot, 268, 284

median_width
 Class svg_boxplot, 268, 284
 Class svg_boxplot_series, 309, 313

minimum
 1-D Auto scaling Examples, 31, 36
 1-D Autoscaling Various Containers Examples, 40

1-D Axis Scaling, 26
2-D Autoscaling Examples, 73
Acknowledgements, 369
Class axis_line_style, 323
Class svg_1d_plot, 194, 198, 204, 205, 206, 207, 212
Class svg_2d_plot, 233, 236, 242, 243, 244, 245, 250, 256, 257, 258, 259
Class svg_boxplot, 285, 286, 287, 291, 298, 299, 300, 305, 306, 308
Class svg_boxplot_series, 309, 310, 312, 313, 314, 315
Class template_smallest, 106
Class template_unc, 350
Class text_style, 338
Class ticks_labels_style, 340, 341
Class value_style, 343
Colors, 6
Definitions of the Quartiles, 86, 87
Fonts, 8
Function median, 168
Function quantile, 169
Function scale_axis, 98, 99
Function template_mnnmx, 95
Function template_operator<<, 351
Function template_range_all, 95
Function template_range_mx, 96
Function template_scale_axis, 99, 100, 101
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
Header < boost/svg_plot/detail/fp_compare.hpp >, 104
Header < boost/svg_plot/detail/numeric_limits_handling.hpp >, 107
Header < boost/svg_plot/quantile.hpp >, 168
Header < boost/svg_plot/uncertain.hpp >, 346
Implementation and other notes, 363
Preface, 3
Showing data values at Numerical Limits, 92
Tutorial: 1D Autoscale with Multiple Containers, 18
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 2D Special Features, 62
Type svg_color_constant, 318
Using Inkscape, 356
min_value
 Header < boost/svg_plot/detail/fp_compare.hpp >, 104
min_whisker_color
 Class svg_boxplot_series, 309, 313
min_whisker_width
 Class svg_boxplot_series, 309, 314
my_blank
 Class svg_color, 317
my_color
 Function operator<<, 320
m_path
 Struct m_path, 128

N

neareq
 Header < boost/svg_plot/detail/fp_compare.hpp >, 104

O

one_sd_color
 Class svg_1d_plot, 179, 192

Class `svg_2d_plot`, 216, 231
Class `svg_boxplot`, 268, 284
`origin`
 1-D Auto scaling Examples, 31
 1-D Axis Scaling, 26
 Demonstration of adding lines and curves, typically a least squares fit, 80
 Function `scale_axis`, 98, 99
 Function template `scale_axis`, 99, 100, 101
`outFmtFlags`
 Function `outFmtFlags`, 170
 Header <`boost/svg_plot/show_1d_settings.hpp`>, 169
`outlier`
 Class `svg_boxplot`, 279, 284, 285, 286
 Class `svg_boxplot_series`, 311, 312, 314, 315
`outlier_color`
 Class `svg_boxplot`, 268, 284
 Class `svg_boxplot_series`, 309, 314
`outlier_fill`
 Class `svg_boxplot`, 268, 284, 285
 Class `svg_boxplot_series`, 309, 314
`outlier_shape`
 Class `svg_boxplot`, 268, 285
 Class `svg_boxplot_series`, 309, 314
`outlier_size`
 Class `svg_boxplot`, 268, 285
 Class `svg_boxplot_series`, 309, 314
`outlier_style`
 Class `svg_boxplot`, 268, 285
 Class `svg_boxplot_series`, 309, 315
`outlier_values_on`
 Class `svg_boxplot`, 268, 285

P

`P`
 Class `polyline_element`, 139, 141
 Struct `polygon_element`, 136, 138
`p`
 Function `operator<<`, 167
 Struct `polygon_element`, 138
`p0`
 Function `operator<<`, 167
`path`
 Class `g_element`, 119, 122
 Class `svg`, 172, 176
`path_element`
 Class `path_element`, 129
`path_point`
 Struct `a_path`, 110
 Struct `c_path`, 111
 Struct `h_path`, 124
 Struct `l_path`, 125
 Struct `m_path`, 128
 Struct `path_point`, 134
 Struct `q_path`, 142
 Struct `s_path`, 149
 Struct `t_path`, 152
 Struct `v_path`, 164

Struct z_path, 165
pentagon
 Class g_element, 119, 122
 Class svg, 172, 176
plot
 Class svg_1d_plot, 179, 192, 193, 195
 Class svg_2d_plot, 216, 231, 232, 234
 Class svg_boxplot, 268, 285, 286
 Colors, 6
 Function template scale_axis, 100
 Showing data values at Numerical Limits, 92
 Simple Code Example, 58
 Tutorial: 2D Special Features, 62
Plotting Graphs in SVG format.
 SVG, 2
plot_background_color
 Class svg_1d_plot, 179, 193, 194
 Class svg_2d_plot, 216, 232
 Class svg_boxplot, 268, 286
plot_border_color
 Class svg_1d_plot, 179, 194
 Class svg_2d_plot, 216, 232
 Class svg_boxplot, 268, 286
plot_border_width
 Class svg_1d_plot, 179, 194
 Class svg_2d_plot, 216, 232
 Class svg_boxplot, 268, 287
plot_line_style
 Class plot_line_style, 330
plot_point_style
 Class plot_point_style, 332
plot_window_on
 Class svg_1d_plot, 179, 194
 Class svg_2d_plot, 216, 232, 233
 Class svg_boxplot, 268, 287
plot_window_x
 Class svg_1d_plot, 179, 194
 Class svg_2d_plot, 216, 232
 Class svg_boxplot, 268, 287
plot_window_x_left
 Class svg_1d_plot, 179, 194
 Class svg_2d_plot, 216, 232
 Class svg_boxplot, 268, 287
plot_window_x_right
 Class svg_1d_plot, 179, 194
 Class svg_2d_plot, 216, 232
 Class svg_boxplot, 268, 287
plot_window_y
 Class svg_1d_plot, 179, 194, 195
 Class svg_2d_plot, 216, 232
 Class svg_boxplot, 268, 287
plot_window_y_bottom
 Class svg_1d_plot, 179, 195
 Class svg_2d_plot, 216, 232
 Class svg_boxplot, 268, 287
plot_window_y_top
 Class svg_1d_plot, 179, 195
 Class svg_2d_plot, 216, 232

Class `svg_boxplot`, 268, 287
polygon
 Class `g_element`, 119, 122
 Class `svg`, 172, 176
 Struct `polygon_element`, 136
polygon_element
 Struct `polygon_element`, 136
polyline
 Class `g_element`, 119, 122
 Class `svg`, 172, 176, 177
polyline_element
 Class `polyline_element`, 139
poly_path_point
 Struct `poly_path_point`, 135
position
 Class `axis_line_style`, 323, 324
 Class `svg_1d_plot`, 199
 Class `svg_2d_plot`, 237
precision
 1-D Data Values Examples, 43
 2-D Data Values Examples, 67
 Class `svg`, 174
 Class `svg_1d_plot`, 186, 197, 209, 210
 Class `svg_2d_plot`, 225, 247, 248, 249, 261, 262, 263
 Class `svg_boxplot`, 277, 303, 304, 305
 Class `ticks_labels_style`, 340
 Class `value_style`, 342
 Demonstration of using 1D data that includes information about its Uncertainty, 54
Fonts, 8
 Header < boost/svg_plot/detail/functors.hpp >, 107
 Implementation and other notes, 363, 364
 Real-life Heat flow data, 48
 Showing data values at Numerical Limits, 92
 Tutorial: 1D Gridlines & Axes - more examples, 22
 Tutorial: 2D Special Features, 62
Preface
 `boxplot`, 3
 `container`, 3
 `data`, 3
 `example`, 3
 `Inkscape`, 3
 `minimum`, 3
 `SVG`, 3, 4
 `uncertainty`, 3
prefix
 Class `svg_2d_plot`, 258
push_back
 Class `g_element`, 119, 122

Q

q
 Class `path_element`, 129, 132
Q
 Class `path_element`, 129, 132
quantile
 Definitions of the Quartiles, 86
 Function `quantile`, 169

Header < boost/svg_plot/quantile.hpp >, 168
quartile
 Class svg_boxplot, 268, 286, 288
 Class svg_boxplot_series, 309, 310, 311, 315
 Definitions of the Quartiles, 86, 87
 Function quantile, 169
 Header < boost/svg_plot/quantile.hpp >, 168
 Header < boost/svg_plot/svg_boxplot.hpp >, 267
 Real-life Heat flow data, 48
quartile_definition
 Class svg_boxplot, 268, 288
 Class svg_boxplot_series, 309, 315
curve_element
 Class curve_element, 143
q_path
 Struct q_path, 142

R

r
 Function operator<<, 166
Real-life Heat flow data
 boxplot, 48
 data, 48
 example, 48
 label, 48
 main, 48
 precision, 48
 quartile, 48
 scaling, 48
 SVG, 48
 title, 48
rect
 Class g_element, 119, 122
 Class svg, 172, 177
rect_element
 Class rect_element, 146
reproduction
 Class svg, 172, 177
rhombus
 Class g_element, 119, 123
 Class svg, 172, 177
rotation
 Class svg_1d_plot, 210, 211
 Class svg_2d_plot, 249
 Class svg_boxplot, 305
 Class text_element, 153, 155
 Class tspan_element, 159, 162
rounddown10
 Function rounddown10, 96
 Header < boost/svg_plot/detail/auto_axes.hpp >, 94
rounddown2
 Function rounddown2, 96
 Header < boost/svg_plot/detail/auto_axes.hpp >, 94
rounddown5
 Function rounddown5, 97
 Header < boost/svg_plot/detail/auto_axes.hpp >, 94
roundup10

Function roundup10, 97
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
roundup2
Function roundup2, 97
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
roundup5
Function roundup5, 98
Header < boost/svg_plot/detail/auto_axes.hpp >, 94

S

s
2-D Autoscaling Examples, 73
Class path_element, 129, 132
s
Class path_element, 129, 132
safe_fpt_division
Function template safe_fpt_division, 107
Header < boost/svg_plot/detail/fp_compare.hpp >, 104
scale_axis
Function scale_axis, 98, 99
Function template scale_axis, 99, 100, 101
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
scaling
1-D Auto scaling Examples, 31
1-D Axis Scaling, 26
1-D Vector Example, 14
Class axis_line_style, 323
Class path_element, 129
Class svg, 177
Class svg_1d_plot, 194, 197, 201, 208
Class svg_2d_plot, 233, 239, 246, 249, 250, 253, 260, 262
Class svg_boxplot, 287, 294, 302, 305, 306
Class svg_boxplot_series, 309, 311
Class template unc, 347
Definitions of the Quartiles, 86, 87
Demonstration of adding lines and curves, typically a least squares fit, 80
Fonts, 8
Function template scale_axis, 99, 100
Global sin45, 103
Header < boost/svg_plot/detail/axis_plot_frame.hpp >, 102
Header < boost/svg_plot/quantile.hpp >, 168
Implementation and other notes, 363
Real-life Heat flow data, 48
Showing data values at Numerical Limits, 92
Struct polygon_element, 136
To Do List, 368
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 2D Special Features, 62
Using Inkscape, 356
series
1-D Auto scaling Examples, 31
1-D Autoscaling Various Containers Examples, 40
1-D Axis Scaling, 26
1-D Data Values Examples, 43
1-D STL Containers Examples, 15
1-D Vector Example, 14
2-D Autoscaling Examples, 73

2-D Data Values Examples, 67
Class `plot_line_style`, 329, 330
Class `svg_1d_plot`, 189, 192, 193
Class `svg_1d_plot_series`, 212, 213, 214
Class `svg_2d_plot`, 228, 231, 232, 249, 250
Class `svg_2d_plot_series`, 263, 264, 265, 266
Class `svg_boxplot`, 267, 281, 285, 286
Class `svg_boxplot_series`, 310, 311, 315
Class `value_style`, 342
Definitions of the Quartiles, 86
Demonstration of using 1D data that includes information about its Uncertainty, 54
Demonstration of using 2D data that includes information about its uncertainty, 76
Function template `range_mx`, 96
Function template `scale_axis`, 99, 100, 101
Implementation and other notes, 363
Simple Code Example, 58, 59
Simple Example, 84
To Do List, 368
Tutorial: 1D Autoscale with Multiple Containers, 18
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 2D Special Features, 62
Tutorial: Fuller Layout Example, 60
`series_count`
 Class `svg_1d_plot_series`, 212, 214
`series_limits_count`
 Class `svg_1d_plot_series`, 212, 214
`set_ids`
 Header <`boost/svg_plot/detail/svg_boxplot_detail.hpp`>, 108
`shape`
 Class `plot_point_style`, 332, 333
 Class `svg`, 176
 Class `svg_1d_plot_series`, 212, 214
 Class `svg_2d_plot_series`, 263, 266
`show`
 Function template `show`, 102
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 94
Showing data values at Numerical Limits
 color, 92
 data, 92
 example, 92
 fill, 92
 isnan, 92
 maximum, 92
 minimum, 92
 plot, 92
 precision, 92
 scaling, 92
`show_1d_plot_settings`
 1-D Auto scaling Examples, 31
 1-D Data Values Examples, 43
 Function `show_1d_plot_settings`, 170
 Header <`boost/svg_plot/show_1d_settings.hpp`>, 169
 Tutorial: 2D Special Features, 62
`show_2d_plot_settings`
 2-D Data Values Examples, 67
 Function `show_2d_plot_settings`, 171
 Header <`boost/svg_plot/show_2d_settings.hpp`>, 170
`show_all`

Function template show_all, 102
Header < boost/svg_plot/detail/auto_axes.hpp >, 94

Simple Code Example
background, 58, 59
border, 59
color, 58, 59
container, 58
data, 58
example, 58
legend, 58, 59
main, 58
marker, 58
plot, 58
series, 58, 59
SVG, 58
title, 58, 59

Simple Example
background, 84
border, 84
boxplot, 84
color, 84
data, 84
example, 84
f, 84
fill, 84
g, 84
label, 84
main, 84
series, 84
sin, 84
SVG, 84
title, 84

sin
 Simple Example, 84
 To Do List, 368

size
 Class g_element, 119, 123
 Class plot_point_style, 332, 333
 Class svg, 172, 177
 Class svg_1d_plot, 179, 189, 195, 201, 207, 208, 209, 212
 Class svg_1d_plot_series, 212, 214
 Class svg_2d_plot, 216, 228, 233, 239, 240, 245, 247, 259
 Class svg_2d_plot_series, 263, 266
 Class svg_boxplot, 268, 281, 288, 295, 301, 303, 308
 Class template close_to, 105
 Class template smallest, 106
 Class text_style, 337, 338
 Using Inkscape, 356

smallest
 Class template smallest, 106

sqrt
 Tutorial: 1D More Layout Examples, 20
 Tutorial: Fuller Layout Example, 60

strength
 Class template close_to, 105

string_svg_length
 Function string_svg_length, 346
 Header < boost/svg_plot/svg_style.hpp >, 321

strip_e0s
Function strip_e0s, 346
Header < boost/svg_plot/svg_style.hpp >, 321

stroke
1-D Auto scaling Examples, 31
1-D Data Values Examples, 43
2-D Data Values Examples, 67
Class axis_line_style, 323
Class bar_style, 325
Class box_style, 327, 328
Class circle_element, 114
Class clip_path_element, 116
Class ellipse_element, 118
Class g_element, 123
Class line_element, 128
Class path_element, 133
Class plot_line_style, 330
Class plot_point_style, 332, 333
Class polyline_element, 141
Class curve_element, 145
Class rect_element, 148
Class svg_1d_plot, 188, 189, 191
Class svg_1d_plot_series, 213, 214, 215
Class svg_2d_plot, 228, 230, 231, 251
Class svg_2d_plot_series, 265, 266
Class svg_boxplot, 280, 283
Class svg_element, 150, 151
Class svg_style, 334, 335, 336
Class text_element, 157
Class ticks_labels_style, 340
Class tspan_element, 164
Class value_style, 342
Function operator<<, 344
Header < boost/svg_plot/svg_style.hpp >, 321
Implementation and other notes, 363
Struct polygon_element, 138

stroke_color
Class plot_point_style, 332, 333
Class svg_1d_plot_series, 212, 214, 215
Class svg_2d_plot_series, 263, 266
Class svg_style, 334, 335

stroke_on
Class svg_style, 334, 335

stroke_width
Class svg_style, 334, 335, 336

Struct a_path
a_path, 110
data, 110
path_point, 110
SVG, 110
svg, 110
write, 110

Struct c_path
c_path, 111
data, 111
path_point, 111
SVG, 110, 111
svg, 111

```
    write, 111
Struct h_path
    data, 124
    h_path, 124
    path_point, 124
    SVG, 124
    svg, 124
    write, 124
Struct l_path
    data, 125
    l_path, 125
    path_point, 125
    SVG, 124, 125
    svg, 125
    write, 125
Struct m_path
    data, 128
    example, 128
    m_path, 128
    path_point, 128
    SVG, 128
    svg, 128
    write, 128
Struct path_point
    data, 134
    path_point, 134
    SVG, 133, 134
    write, 134
Struct polygon_element
    class_id, 136, 137
    clip_id, 136, 137
    color, 136, 138
    container, 136
    data, 136
    example, 137, 138
    fill, 136, 138
    id, 136, 137, 138
    P, 136, 138
    p, 138
    polygon, 136
    polygon_element, 136
    scaling, 136
    stroke, 138
    style, 136, 138
    SVG, 135, 136, 137, 138
    svg, 136
    svg_element, 136
    write, 136, 138
    write_attributes, 136, 138
Struct poly_path_point
    data, 135
    example, 135
    poly_path_point, 135
    SVG, 134, 135
    write, 135
Struct q_path
    data, 142
    path_point, 142
```

q_path, 142
SVG, 141, 142
svg, 142
write, 142

Struct s_path
 data, 149
 path_point, 149
 SVG, 148, 149
 svg, 149
 s_path, 149
 write, 149

Struct t_path
 data, 152
 path_point, 152
 SVG, 152
 svg, 152
 t_path, 152
 write, 152

Struct v_path
 data, 164
 path_point, 164
 SVG, 164, 165
 svg, 164
 v_path, 164
 write, 164, 165

Struct z_path
 data, 165
 path_point, 165
 SVG, 165
 svg, 165
 write, 165
 z_path, 165

style
 Class circle_element, 112, 113
 Class clip_path_element, 114, 115
 Class ellipse_element, 116, 117, 118
 Class g_element, 119, 123
 Class line_element, 126, 127
 Class path_element, 129, 132
 Class plot_point_style, 332, 333
 Class polyline_element, 139, 141
 Class curve_element, 143, 144
 Class rect_element, 146, 148
 Class svg_element, 150, 152
 Class text_element, 153, 155
 Class ticks_labels_style, 340
 Class tspan_element, 159, 162, 163
 Class value_style, 342
 Function operator<<, 166
 Implementation and other notes, 363
 Struct polygon_element, 136, 138

SVG
 1-D Auto scaling Examples, 31
 1-D Autoscaling Various Containers Examples, 40
 1-D Axis Scaling, 26
 1-D Data Values Examples, 43
 1-D STL Containers Examples, 15, 17, 18
 1-D Vector Example, 14

2-D Autoscaling Examples, 73
2-D Data Values Examples, 67
Boost.SVG plot C++ Reference, 94
Class axis_line_style, 322, 323
Class bar_style, 325
Class box_style, 326
Class circle_element, 111, 112, 113
Class clip_path_element, 114, 115
Class ellipse_element, 116, 117, 118
Class g_element, 118, 119, 120, 121, 123
Class histogram_style, 328
Class line_element, 125, 126, 127
Class path_element, 128, 129, 130, 131, 133
Class plot_line_style, 329
Class plot_point_style, 331
Class polyline_element, 139, 140, 141
Class quurve_element, 142, 143, 144, 145
Class rect_element, 145, 146, 147, 148
Class svg, 172, 173, 174, 175, 176, 177, 178
Class svg_1d_plot, 179, 185, 186, 187, 188, 189, 190, 191, 194, 195, 196, 197, 201, 204, 205, 206, 207, 208, 209, 212
Class svg_1d_plot_series, 212
Class svg_2d_plot, 215, 223, 225, 226, 227, 228, 229, 230, 232, 233, 235, 236, 239, 240, 242, 244, 245, 246, 247, 253, 259, 260, 262
Class svg_2d_plot_series, 263
Class svg_boxplot, 267, 275, 276, 277, 278, 279, 280, 281, 282, 283, 287, 288, 289, 290, 291, 294, 295, 298, 299, 301, 302, 303, 308, 309
Class svg_boxplot_series, 309
Class svg_color, 316, 317
Class svg_element, 149, 150, 151, 152
Class svg_style, 334, 335, 336
Class template_unc, 347
Class text_element, 153, 154, 155, 156
Class text_element_text, 157
Class text_parent, 157, 158
Class text_style, 336, 337, 338, 339
Class ticks_labels_style, 339, 340
Class tspan_element, 158, 159, 160, 161, 162, 163, 164
Class value_style, 342
Colors, 6
Compilers and Examples, 362
Definitions of the Quartiles, 86
Demonstration of adding lines and curves, typically a least squares fit, 80
Demonstration of using 1D data that includes information about its Uncertainty, 54
Demonstration of using 2D data that includes information about its uncertainty, 76
Fonts, 8
Function constant_to_rgb, 319
Function default_font, 343
Function is_blank, 319
Function median, 168
Function operator!=, 165, 166, 320, 344
Function operator<<, 166, 167, 320, 344, 345
Function operator==, 167, 320, 345
Function outFmtFlags, 170
Function quantile, 168
Function rounddown10, 96
Function rounddown2, 96
Function rounddown5, 97
Function roundup10, 97

Function roundup2, 97
Function roundup5, 98
Function scale_axis, 98
Function show_1d_plot_settings, 170
Function show_2d_plot_settings, 171
Function string_svg_length, 346
Function strip_e0s, 346
Function template mnmx, 95
Function template operator<<, 350, 351
Function template range_all, 95
Function template range_mx, 96
Function template scale_axis, 99, 100
Function template show, 101, 102
Function template show_all, 102
Function template uncs_of, 352, 353
Function template unc_of, 351, 352
Function template values_of, 354, 355
Function template value_of, 353, 354
Global color_array, 318
Global document_ids_, 108
Global fmtFlagWords, 169
Global no_style, 343
Global package_info, 178
Global plusminus, 350
Global reducer, 103
Global sin45, 103
Global text_plusminus, 103
Global wh, 343
Header < boost/svg_plot/detail/auto_axes.hpp >, 94
Header < boost/svg_plot/detail/axis_plot_frame.hpp >, 102
Header < boost/svg_plot/detail/functors.hpp >, 107
Header < boost/svg_plot/detail/svg_boxplot_detail.hpp >, 108
Header < boost/svg_plot/detail/svg_tag.hpp >, 108
Header < boost/svg_plot/quantile.hpp >, 168
Header < boost/svg_plot/show_1d_settings.hpp >, 169
Header < boost/svg_plot/show_2d_settings.hpp >, 170
Header < boost/svg_plot/svg.hpp >, 171
Header < boost/svg_plot/svg_1d_plot.hpp >, 178
Header < boost/svg_plot/svg_2d_plot.hpp >, 215
Header < boost/svg_plot/svg_boxplot.hpp >, 267
Header < boost/svg_plot/svg_color.hpp >, 315
Header < boost/svg_plot/svg_style.hpp >, 321
Header < boost/svg_plot/uncertain.hpp >, 346
Implementation and other notes, 363
Plotting Graphs in SVG format., 2
Preface, 3, 4
Real-life Heat flow data, 48
Simple Code Example, 58
Simple Example, 84
Struct a_path, 110
Struct c_path, 110, 111
Struct h_path, 124
Struct l_path, 124, 125
Struct m_path, 128
Struct path_point, 133, 134
Struct polygon_element, 135, 136, 137, 138
Struct poly_path_point, 134, 135
Struct q_path, 141, 142

Struct s_path, 148, 149
Struct t_path, 152
Struct v_path, 164, 165
Struct z_path, 165
SVG tutorial, 91
To Do List, 368
Tutorial: 1D Autoscale with Multiple Containers, 18
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 1D More Layout Examples, 20
Tutorial: 2D Special Features, 62
Tutorial: Fuller Layout Example, 60
Type svg_color_constant, 317, 318
Using Inkscape, 356

svg
 Class circle_element, 112
 Class clip_path_element, 114
 Class ellipse_element, 116
 Class g_element, 119
 Class line_element, 126
 Class path_element, 129
 Class polyline_element, 139
 Class quurve_element, 143
 Class rect_element, 146
 Class svg, 172
 Class text_element, 153
 Class text_element_text, 157
 Class tspan_element, 159
 Struct a_path, 110
 Struct c_path, 111
 Struct h_path, 124
 Struct l_path, 125
 Struct m_path, 128
 Struct polygon_element, 136
 Struct q_path, 142
 Struct s_path, 149
 Struct t_path, 152
 Struct v_path, 164
 Struct z_path, 165

SVG tutorial
 example, 91
 SVG, 91
svg_1d_plot
 Class svg_1d_plot, 179
svg_1d_plot_series
 Class svg_1d_plot_series, 212
svg_2d_plot
 Class svg_2d_plot, 216
svg_2d_plot_series
 Class svg_2d_plot_series, 263

svg_boxplot
 Class svg_boxplot, 268
svg_boxplot_series
 Class svg_boxplot_series, 309

svg_color
 Class svg_color, 316
 Function constant_to_rgb, 319
svg_element
 Class circle_element, 112

Class clip_path_element, 114
 Class ellipse_element, 116
 Class g_element, 119
 Class line_element, 126
 Class path_element, 129
 Class polyline_element, 139
 Class quurve_element, 143
 Class rect_element, 146
 Class svg_element, 150
 Class text_element, 153
 Class tspan_element, 159
 Struct polygon_element, 136
svg_style
 Class svg_style, 334
symbols
 Class plot_point_style, 332, 333, 334
 Class svg_1d_plot_series, 212, 215
s_path
 Struct s_path, 149

T

t
 Class path_element, 129, 132
 Function operator<<, 166

T
 Class path_element, 129, 133

text
 Class g_element, 119, 123
 Class svg, 172, 177
 Class text_element, 153, 155, 156
 Class tspan_element, 159, 163
 Header < boost/svg_plot/svg_style.hpp >, 321

textstyle
 Class text_element, 153, 156
 Class tspan_element, 159, 163

text_element
 Class text_element, 153

text_element_text
 Class text_element_text, 157

text_length
 Class tspan_element, 159, 163

text_parent
 Class text_element_text, 157
 Class text_parent, 158
 Class tspan_element, 159

text_style
 Class text_style, 337

three_sd_color
 Class svg_1d_plot, 179, 195
 Class svg_2d_plot, 216, 233, 234
 Class svg_boxplot, 268, 288

ticks_labels_style
 Class ticks_labels_style, 340

tiny
 Header < boost/svg_plot/detail/fp_compare.hpp >, 104
title
 1-D Auto scaling Examples, 31

1-D Autoscaling Various Containers Examples, 40
1-D Data Values Examples, 43
1-D STL Containers Examples, 15
1-D Vector Example, 14
2-D Autoscaling Examples, 73
2-D Data Values Examples, 67
Class `svg`, 174, 175
Class `svg_1d_plot`, 179, 187, 189, 190, 192, 193, 195, 196, 197
Class `svg_1d_plot_series`, 213
Class `svg_2d_plot`, 216, 223, 226, 228, 229, 231, 232, 234, 235
Class `svg_2d_plot_series`, 264
Class `svg_boxplot`, 268, 278, 281, 282, 285, 286, 288, 289, 290, 295
Class `svg_boxplot_series`, 309, 310, 315
Class `svg_style`, 334
Definitions of the Quartiles, 86
Demonstration of using 1D data that includes information about its Uncertainty, 54
Fonts, 8
Function `default_font`, 344
Global package_info, 178
Implementation and other notes, 363
Real-life Heat flow data, 48
Simple Code Example, 58, 59
Simple Example, 84
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 1D More Layout Examples, 20
Tutorial: 2D Special Features, 62
Tutorial: Fuller Layout Example, 60

`title_color`
 Class `svg_1d_plot`, 179, 195
 Class `svg_2d_plot`, 216, 234
 Class `svg_boxplot`, 268, 288, 289

`title_font_alignment`
 Class `svg_1d_plot`, 179, 196
 Class `svg_2d_plot`, 216, 234
 Class `svg_boxplot`, 268, 289

`title_font_decoration`
 Class `svg_1d_plot`, 179, 196
 Class `svg_2d_plot`, 216, 234
 Class `svg_boxplot`, 268, 289

`title_font_family`
 Class `svg_1d_plot`, 179, 196
 Class `svg_2d_plot`, 216, 234
 Class `svg_boxplot`, 268, 289

`title_font_rotation`
 Class `svg_1d_plot`, 179, 196
 Class `svg_2d_plot`, 216, 234, 235
 Class `svg_boxplot`, 268, 289

`title_font_size`
 Class `svg_1d_plot`, 179, 196
 Class `svg_2d_plot`, 216, 235
 Class `svg_boxplot`, 268, 289

`title_font_stretch`
 Class `svg_1d_plot`, 179, 196
 Class `svg_2d_plot`, 216, 235
 Class `svg_boxplot`, 268, 289

`title_font_style`
 Class `svg_1d_plot`, 179, 196, 197
 Class `svg_2d_plot`, 216, 235

Class `svg_boxplot`, 268, 290

`title_font_weight`

Class `svg_1d_plot`, 179, 197

Class `svg_2d_plot`, 216, 235

Class `svg_boxplot`, 268, 290

`title_on`

Class `svg_1d_plot`, 179, 197

Class `svg_2d_plot`, 216, 235

Class `svg_boxplot`, 268, 290

`title_size`

Class `svg_boxplot`, 268, 290

To Do List

`container`, 368

`data`, 368

`example`, 368

`Inkscape`, 368

`label`, 368

`scaling`, 368

`series`, 368

`sin`, 368

`SVG`, 368

`triangle`

Class `g_element`, 119, 123

Class `svg`, 172, 177

`ts`

Function operator`<<`, 345

`tspan`

Class `text_element`, 153, 156

`tspan_element`

Class `tspan_element`, 159

Tutorial: 1D Autoscale with Multiple Containers

`color`, 18

`container`, 18

`data`, 18

`example`, 18

`maximum`, 18

`minimum`, 18

`series`, 18

`SVG`, 18

`x_autoscale`, 18

`x_range`, 18

Tutorial: 1D Gridlines & Axes - more examples

`background`, 22

`color`, 22

`container`, 22

`data`, 22

`example`, 22

`font`, 22

`grid`, 22

`label`, 22

`layout`, 22

`legend`, 22

`main`, 22

`minimum`, 22

`precision`, 22

`scaling`, 22

`series`, 22

`SVG`, 22

title, 22
Unicode, 22
Tutorial: 1D More Layout Examples
background, 20
color, 20
container, 20
data, 20
deque, 20
example, 20
f, 20
fill, 20
g, 20
h, 20
label, 20
layout, 20
legend, 20
main, 20
sqrt, 20
SVG, 20
title, 20
Tutorial: 2D Special Features
color, 62
container, 62
data, 62
example, 62
fill, 62
font, 62
grid, 62
ioflags, 62
label, 62
minimum, 62
plot, 62
precision, 62
scaling, 62
series, 62
show_1d_plot_settings, 62
SVG, 62
title, 62
Unicode, 62
value, 62
Tutorial: Boxplot
boxplot, 84
Tutorial: Fuller Layout Example
background, 60
color, 60
data, 60
example, 60
f, 60
g, 60
h, 60
layout, 60
legend, 60
main, 60
marker, 60
series, 60
sqrt, 60
SVG, 60
title, 60

two_sd_color
 Class `svg_1d_plot`, 179, 197
 Class `svg_2d_plot`, 216, 235, 236
 Class `svg_boxplot`, 268, 290
Type `svg_color_constant`
 `color`, 317, 318
 `minimum`, 318
 `SVG`, 317, 318
types
 Class template `unc`, 348, 349
t_or_b
 Header <`boost/svg_plot/show_1d_settings.hpp`>, 169
t_path
 Struct `t_path`, 152

U

u1
 Demonstration of using 2D data that includes information about its uncertainty, 76
u2
 Demonstration of using 2D data that includes information about its uncertainty, 76
unc
 Class template `unc`, 348
uncertainty
 2-D Data Values Examples, 67
 Class `plot_point_style`, 332
 Class `svg_1d_plot_series`, 213
 Class `svg_2d_plot`, 258
 Class `svg_2d_plot_series`, 264
 Class `svg_boxplot_series`, 310
 Class template `unc`, 347, 348, 349, 350
 Class `value_style`, 342, 343
 Compilers and Examples, 362
 Demonstration of using 1D data that includes information about its Uncertainty, 54
 Demonstration of using 2D data that includes information about its uncertainty, 76
 Function `scale_axis`, 99
 Function template `operator<<`, 351
 Function template `scale_axis`, 99, 100, 101
 Function template `uncs_of`, 352, 353
 Function template `unc_of`, 351, 352
 Function template `values_of`, 354, 355
 Function template `value_of`, 353, 354
 Global `plusminus`, 350
 Global reducer, 103
 Header <`boost/svg_plot/detail/axis_plot_frame.hpp`>, 102
 Header <`boost/svg_plot/detail/functors.hpp`>, 107
 Header <`boost/svg_plot/uncertain.hpp`>, 346
 History, 362
 Preface, 3
uncun
 Demonstration of using 1D data that includes information about its Uncertainty, 54
 Demonstration of using 2D data that includes information about its uncertainty, 76
unc_of
 Function template `unc_of`, 351, 352
 Header <`boost/svg_plot/uncertain.hpp`>, 346
Unicode
 1-D Data Values Examples, 43
 Class `plot_point_style`, 332

Class `svg_1d_plot`, 187, 195, 196, 200, 201, 207, 208
Class `svg_2d_plot`, 226, 234, 238, 239, 245, 246, 252, 253, 259, 260, 262
Class `svg_boxplot`, 278, 288, 289, 293, 294, 301, 302
Class `template unc`, 350
Class `text_element`, 153
Class `text_style`, 337, 338
Demonstration of using 2D data that includes information about its uncertainty, 76
Fonts, 8
Function `default_font`, 344
Function template `operator<<`, 351
Header < `boost/svg_plot/svg_style.hpp` >, 321
Implementation and other notes, 363
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 2D Special Features, 62
`use_down_ticks`
 Class `ticks_labels_style`, 340, 341
`use_style`
 Class `tspan_element`, 159, 163
`use_up_ticks`
 Class `ticks_labels_style`, 340, 341
Using Inkscape
 background, 356
 color, 356
 container, 356
 example, 356
 file, 356
 font, 356
 Inkscape, 356
 minimum, 356
 scaling, 356
 size, 356
 SVG, 356

V

`v`
 Class `path_element`, 129, 133
`V`
 Class `path_element`, 129, 133
`value`
 2-D Data Values Examples, 67
 Class `axis_line_style`, 323
 Class `svg_boxplot`, 279, 284, 285
 Class `template unc`, 348, 349
 Class `ticks_labels_style`, 340
 Tutorial: 2D Special Features, 62
`values_count`
 Class `svg_2d_plot_series`, 263, 266
`value_of`
 Function template `value_of`, 353, 354
 Header < `boost/svg_plot/uncertain.hpp` >, 346
`value_style`
 Class `value_style`, 342
`vector_iterator`
 1-D Auto scaling Examples, 31
 1-D Axis Scaling, 26
`v_path`
 Struct `v_path`, 164

W

whisker_length

- Class svg_boxplot, 268, 290
- Class svg_boxplot_series, 309, 315

width

- Class axis_line_style, 323, 324
- Class bar_style, 325, 326
- Class box_style, 327, 328
- Class plot_line_style, 330, 331
- Class rect_element, 146, 148
- Class svg_1d_plot, 188, 202, 207
- Class svg_2d_plot, 227, 240, 245
- Class svg_boxplot, 280, 288, 296
- Class svg_style, 335

Demonstration of using 1D data that includes information about its Uncertainty, 54

width_on

- Class svg_style, 334, 336

write

- Class circle_element, 112, 113
- Class clip_path_element, 114, 115
- Class ellipse_element, 116, 118
- Class g_element, 119, 123
- Class line_element, 126, 127
- Class path_element, 129, 133
- Class polyline_element, 139, 141
- Class quurve_element, 143, 145
- Class rect_element, 146, 148
- Class svg, 172, 177
- Class svg_1d_plot, 179, 197
- Class svg_2d_plot, 216, 236
- Class svg_boxplot, 268, 291
- Class svg_color, 316, 317
- Class svg_element, 150, 152
- Class svg_style, 334, 336
- Class text_element, 153, 156
- Class text_element_text, 157
- Class text_parent, 158
- Class tspan_element, 159, 163
- Struct a_path, 110
- Struct c_path, 111
- Struct h_path, 124
- Struct l_path, 125
- Struct m_path, 128
- Struct path_point, 134
- Struct polygon_element, 136, 138
- Struct poly_path_point, 135
- Struct q_path, 142
- Struct s_path, 149
- Struct t_path, 152
- Struct v_path, 164, 165
- Struct z_path, 165

write_attributes

- Class circle_element, 112, 114
- Class clip_path_element, 114, 116
- Class ellipse_element, 116, 118
- Class g_element, 119, 123
- Class line_element, 126, 128

Class path_element, 129, 133
Class polyline_element, 139, 141
Class quurve_element, 143, 145
Class rect_element, 146, 148
Class svg_element, 150, 151
Class text_element, 153, 157
Class tspan_element, 159, 164
Struct polygon_element, 136, 138

X

x

Class rect_element, 146, 148
Class text_element, 153, 156
Class tspan_element, 159, 163, 164

xy_autoscale

Class svg_2d_plot, 216, 249

xy_values_on

Class svg_2d_plot, 216, 249

x_addlimits_color

Class svg_1d_plot, 179, 198

Class svg_2d_plot, 216, 236

Class svg_boxplot, 268, 291

x_addlimits_on

Class svg_1d_plot, 179, 198

Class svg_2d_plot, 216, 236

Class svg_boxplot, 268, 291

x_autoscale

1-D Auto scaling Examples, 31

Class svg_1d_plot, 179, 198, 199, 204, 207

Class svg_2d_plot, 216, 236, 237, 242, 245

Class svg_boxplot, 268, 291, 292, 298, 301

Tutorial: 1D Autoscale with Multiple Containers, 18

x_auto_max_value

Class svg_1d_plot, 179, 198

Class svg_2d_plot, 216, 236

Class svg_boxplot, 268, 291

x_auto_min_value

Class svg_1d_plot, 179, 198

Class svg_2d_plot, 216, 236

Class svg_boxplot, 268, 291

x_auto_ticks

Class svg_1d_plot, 179, 198

Class svg_2d_plot, 216, 236

Class svg_boxplot, 268, 291

x_auto_tick_interval

Class svg_1d_plot, 179, 198

Class svg_2d_plot, 216, 236

Class svg_boxplot, 268, 291

x_axis

Class svg_2d_plot, 216, 237

x_axis_color

1-D Auto scaling Examples, 31

Class svg_1d_plot, 179, 199

Class svg_2d_plot, 216, 237

Class svg_boxplot, 268, 292

x_axis_label_color

Class svg_1d_plot, 179, 199

Class `svg_2d_plot`, 216, 237
Class `svg_boxplot`, 268, 292

`x_axis_on`
Class `svg_1d_plot`, 179, 199
Class `svg_2d_plot`, 216, 237
Class `svg_boxplot`, 268, 292

`x_axis_position`
Class `svg_1d_plot`, 179, 199
Class `svg_2d_plot`, 216, 237
Class `svg_boxplot`, 268, 292

`x_axis_vertical`
Class `svg_1d_plot`, 179, 199
Class `svg_2d_plot`, 216, 238
Class `svg_boxplot`, 268, 293

`x_axis_width`
Class `svg_1d_plot`, 179, 200
Class `svg_2d_plot`, 216, 238
Class `svg_boxplot`, 268, 293

`x_datetime_color`
Class `svg_1d_plot`, 179, 200
Class `svg_2d_plot`, 216, 238
Class `svg_boxplot`, 268, 293

`x_datetime_on`
Class `svg_1d_plot`, 179, 200
Class `svg_2d_plot`, 216, 238
Class `svg_boxplot`, 268, 293

`x_decor`
Class `svg_1d_plot`, 179, 200
Class `svg_2d_plot`, 216, 238
Class `svg_boxplot`, 268, 293

`x_df_color`
Class `svg_1d_plot`, 179, 200
Class `svg_2d_plot`, 216, 238
Class `svg_boxplot`, 268, 293

`x_df_on`
Class `svg_1d_plot`, 179, 200
Class `svg_2d_plot`, 216, 238, 239
Class `svg_boxplot`, 268, 293, 294

`x_id_color`
Class `svg_1d_plot`, 179, 200, 201
Class `svg_2d_plot`, 216, 239
Class `svg_boxplot`, 268, 294

`x_id_on`
Class `svg_1d_plot`, 179, 201
Class `svg_2d_plot`, 216, 239
Class `svg_boxplot`, 268, 294

`x_label`
Class `svg_1d_plot`, 179, 201
Class `svg_2d_plot`, 216, 239
Class `svg_boxplot`, 268, 294

`x_labels_strip_e0s`
Class `svg_1d_plot`, 179, 202
Class `svg_2d_plot`, 216, 240
Class `svg_boxplot`, 268, 296

`x_label_color`
Class `svg_1d_plot`, 179, 201
Class `svg_2d_plot`, 216, 239
Class `svg_boxplot`, 268, 294

x_label_font_family
 Class svg_1d_plot, 179, 201
 Class svg_2d_plot, 216, 239
 Class svg_boxplot, 268, 294, 295
x_label_font_size
 Class svg_1d_plot, 179, 201
 Class svg_2d_plot, 216, 239, 240
 Class svg_boxplot, 268, 295
x_label_on
 Class svg_1d_plot, 179, 201, 202
 Class svg_2d_plot, 216, 239, 240
 Class svg_boxplot, 268, 295
x_label_size
 Class svg_boxplot, 268, 295
x_label_text
 Class svg_boxplot, 268, 295
x_label_units
 Class svg_1d_plot, 179, 202
 Class svg_2d_plot, 216, 240
 Class svg_boxplot, 268, 295
x_label_units_on
 Class svg_1d_plot, 179, 202
 Class svg_2d_plot, 216, 240
 Class svg_boxplot, 268, 295, 296
x_label_width
 Class svg_1d_plot, 179, 202
 Class svg_2d_plot, 216, 240
 Class svg_boxplot, 268, 296
x_major_grid_color
 Class svg_1d_plot, 179, 202
 Class svg_2d_plot, 216, 241
 Class svg_boxplot, 268, 296
x_major_grid_on
 Class svg_1d_plot, 179, 202, 203
 Class svg_2d_plot, 216, 241
 Class svg_boxplot, 268, 296
x_major_grid_width
 Class svg_1d_plot, 179, 203
 Class svg_2d_plot, 216, 241
 Class svg_boxplot, 268, 296
x_major_interval
 Class svg_1d_plot, 179, 203
 Class svg_2d_plot, 216, 241
 Class svg_boxplot, 268, 296
x_major_labels
 Class svg_boxplot, 268, 297
x_major_labels_on
 Class svg_boxplot, 268, 297
x_major_labels_side
 Class svg_1d_plot, 179, 203
 Class svg_2d_plot, 216, 241, 242
 Class svg_boxplot, 268, 297
x_major_label_rotation
 Class svg_1d_plot, 179, 203
 Class svg_2d_plot, 216, 241
 Class svg_boxplot, 268, 297
x_major_tick
 Class svg_1d_plot, 179, 203

Class `svg_2d_plot`, 216, 242
Class `svg_boxplot`, 268, 297
`x_major_tick_color`
 Class `svg_1d_plot`, 179, 204
 Class `svg_2d_plot`, 216, 242
 Class `svg_boxplot`, 268, 297
`x_major_tick_length`
 Class `svg_1d_plot`, 179, 204
 Class `svg_2d_plot`, 216, 242
 Class `svg_boxplot`, 268, 298
`x_major_tick_width`
 Class `svg_1d_plot`, 179, 204
 Class `svg_2d_plot`, 216, 242
 Class `svg_boxplot`, 268, 298
`x_max`
 Class `svg_1d_plot`, 179, 204
 Class `svg_2d_plot`, 216, 242
 Class `svg_boxplot`, 268, 298
`x_min`
 Class `svg_1d_plot`, 179, 204
 Class `svg_2d_plot`, 216, 242, 243
 Class `svg_boxplot`, 268, 298
`x_minor_grid_color`
 Class `svg_1d_plot`, 179, 205
 Class `svg_2d_plot`, 216, 243
 Class `svg_boxplot`, 268, 298
`x_minor_grid_on`
 Class `svg_1d_plot`, 179, 205
 Class `svg_2d_plot`, 216, 243
 Class `svg_boxplot`, 268, 299
`x_minor_grid_width`
 Class `svg_1d_plot`, 179, 205
 Class `svg_2d_plot`, 216, 243
 Class `svg_boxplot`, 268, 299
`x_minor_interval`
 Class `svg_1d_plot`, 179, 205
 Class `svg_2d_plot`, 216, 243
 Class `svg_boxplot`, 268, 299
`x_minor_tick_color`
 Class `svg_1d_plot`, 179, 205
 Class `svg_2d_plot`, 216, 243
 Class `svg_boxplot`, 268, 299
`x_minor_tick_length`
 Class `svg_1d_plot`, 179, 205
 Class `svg_2d_plot`, 216, 244
 Class `svg_boxplot`, 268, 299
`x_minor_tick_width`
 Class `svg_1d_plot`, 179, 205, 206
 Class `svg_2d_plot`, 216, 244
 Class `svg_boxplot`, 268, 299
`x_min_ticks`
 Class `svg_1d_plot`, 179, 204
 Class `svg_2d_plot`, 216, 243
 Class `svg_boxplot`, 268, 298
`x_num_minor_ticks`
 Class `svg_1d_plot`, 179, 206
 Class `svg_2d_plot`, 216, 244
 Class `svg_boxplot`, 268, 299, 300

Implementation and other notes, 363

x_order_color

Class `svg_1d_plot`, 179, 206
Class `svg_2d_plot`, 216, 244
Class `svg_boxplot`, 268, 300

x_order_on

Class `svg_1d_plot`, 179, 206
Class `svg_2d_plot`, 216, 244
Class `svg_boxplot`, 268, 300

x_plusminus_color

Class `svg_1d_plot`, 179, 206
Class `svg_2d_plot`, 216, 244
Class `svg_boxplot`, 268, 300

x_plusminus_on

Class `svg_1d_plot`, 179, 206
Class `svg_2d_plot`, 216, 245
Class `svg_boxplot`, 268, 300

x_prefix

Class `svg_1d_plot`, 179, 206
Class `svg_2d_plot`, 216, 245
Class `svg_boxplot`, 268, 300

x_range

1-D Auto scaling Examples, 31
1-D Autoscaling Various Containers Examples, 40
1-D Axis Scaling, 26
Class `svg_1d_plot`, 179, 207
Class `svg_2d_plot`, 216, 245
Class `svg_boxplot`, 268, 300
Tutorial: 1D Autoscale with Multiple Containers, 18

x_separator

Class `svg_1d_plot`, 179, 207
Class `svg_2d_plot`, 216, 245
Class `svg_boxplot`, 268, 301

x_size

1-D Data Values Examples, 43
Class `svg`, 172, 177, 178
Class `svg_1d_plot`, 179, 188, 207
Class `svg_2d_plot`, 216, 227, 245
Class `svg_boxplot`, 268, 280, 301

x_steps

Class `svg_1d_plot`, 179, 207
Class `svg_2d_plot`, 216, 245
Class `svg_boxplot`, 268, 301

x_suffix

Class `svg_1d_plot`, 179, 207
Class `svg_2d_plot`, 216, 246
Class `svg_boxplot`, 268, 301

x_ticks

Class `svg_2d_plot`, 216, 246

x_ticks_down_on

Class `svg_1d_plot`, 179, 207
Class `svg_2d_plot`, 216, 246
Class `svg_boxplot`, 268, 302

x_ticks_on_window_or_axis

Class `svg_1d_plot`, 179, 208
Class `svg_2d_plot`, 216, 246
Class `svg_boxplot`, 268, 302

x_ticks_up_on

Class `svg_1d_plot`, 179, 208
Class `svg_2d_plot`, 216, 246
Class `svg_boxplot`, 268, 302
`x_ticks_values_color`
 Class `svg_1d_plot`, 179, 208
 Class `svg_2d_plot`, 216, 246
 Class `svg_boxplot`, 268, 302
`x_ticks_values_font_family`
 Class `svg_1d_plot`, 179, 208
 Class `svg_2d_plot`, 216, 246
 Class `svg_boxplot`, 268, 302, 303
`x_ticks_values_font_size`
 Class `svg_1d_plot`, 179, 208
 Class `svg_2d_plot`, 216, 247
 Class `svg_boxplot`, 268, 303
`x_ticks_values_ioflags`
 Class `svg_1d_plot`, 179, 208, 209
 Class `svg_2d_plot`, 216, 247
 Class `svg_boxplot`, 268, 303
`x_ticks_values_precision`
 Class `svg_1d_plot`, 179, 209
 Class `svg_2d_plot`, 216, 247
 Class `svg_boxplot`, 268, 303
`x_tick_color`
 Class `svg_boxplot`, 268, 301
`x_tick_length`
 Class `svg_boxplot`, 268, 301
`x_tick_width`
 Class `svg_boxplot`, 268, 302
`x_tight`
 Class `svg_1d_plot`, 179, 209
 Class `svg_2d_plot`, 216, 247
 Class `svg_boxplot`, 268, 303
`x_values_color`
 Class `svg_1d_plot`, 179, 209, 210
 Class `svg_2d_plot`, 216, 248
 Class `svg_boxplot`, 268, 304
`x_values_font_family`
 Class `svg_1d_plot`, 179, 210
 Class `svg_2d_plot`, 216, 248
 Class `svg_boxplot`, 268, 304
`x_values_font_size`
 Class `svg_1d_plot`, 179, 210
 Class `svg_2d_plot`, 216, 248
 Class `svg_boxplot`, 268, 304
`x_values_ioflags`
 Class `svg_1d_plot`, 179, 210
 Class `svg_2d_plot`, 216, 248
 Class `svg_boxplot`, 268, 304
`x_values_on`
 Class `svg_1d_plot`, 179, 210
 Class `svg_2d_plot`, 216, 248
 Class `svg_boxplot`, 268, 304
`x_values_precision`
 Class `svg_1d_plot`, 179, 210
 Class `svg_2d_plot`, 216, 249
 Class `svg_boxplot`, 268, 305
`x_values_rotation`

Class `svg_1d_plot`, 179, 210, 211
Class `svg_2d_plot`, 216, 249
Class `svg_boxplot`, 268, 305
`x_value_font_size`
 Class `svg_1d_plot`, 179, 209
 Class `svg_2d_plot`, 216, 247
 Class `svg_boxplot`, 268, 303
`x_value_ioflags`
 Class `svg_1d_plot`, 179, 209
 Class `svg_2d_plot`, 216, 247, 248
 Class `svg_boxplot`, 268, 303, 304
`x_value_precision`
 Class `svg_1d_plot`, 179, 209
 Class `svg_2d_plot`, 216, 248
 Class `svg_boxplot`, 268, 304
`x_with_zero`
 Class `svg_1d_plot`, 179, 211
 Class `svg_2d_plot`, 216, 249
 Class `svg_boxplot`, 268, 305

Y

`y`
 Class `rect_element`, 146, 148
 Class `text_element`, 153, 156
 Class `tspan_element`, 159, 164
`y_addlimits_color`
 Class `svg_2d_plot`, 216, 249, 250
`y_addlimits_on`
 Class `svg_2d_plot`, 216, 250
`y_autoscale`
 Class `svg_2d_plot`, 216, 250
 Class `svg_boxplot`, 268, 305, 306
`y_axis`
 Class `svg_2d_plot`, 216, 250
`y_axis_color`
 Class `svg_1d_plot`, 179, 211
 Class `svg_2d_plot`, 216, 251
 Class `svg_boxplot`, 268, 306
`y_axis_label_color`
 Class `svg_2d_plot`, 216, 251
`y_axis_on`
 Class `svg_1d_plot`, 179, 211
 Class `svg_2d_plot`, 216, 251
 Class `svg_boxplot`, 268, 306
`y_axis_position`
 Class `svg_2d_plot`, 216, 251
 Class `svg_boxplot`, 268, 306
`y_axis_value_color`
 Class `svg_2d_plot`, 216, 251
`y_axis_width`
 Class `svg_2d_plot`, 216, 251, 252
`y_decor`
 Class `svg_2d_plot`, 216, 252
`y_df_color`
 Class `svg_2d_plot`, 216, 252
`y_df_on`
 Class `svg_2d_plot`, 216, 252

y_label
 Class svg_1d_plot, 179, 211
 Class svg_2d_plot, 216, 252
 Class svg_boxplot, 268, 306
y_labels_strip_e0s
 Class svg_1d_plot, 179, 212
 Class svg_2d_plot, 216, 254
 Class svg_boxplot, 268, 307
y_label_axis
 Class svg_2d_plot, 216, 252, 253
y_label_color
 Class svg_1d_plot, 179, 211
 Class svg_2d_plot, 216, 253
 Class svg_boxplot, 268, 306, 307
y_label_font_family
 Class svg_2d_plot, 216, 253
y_label_font_size
 Class svg_2d_plot, 216, 253
 Class svg_boxplot, 268, 307
y_label_on
 Class svg_1d_plot, 211
 Class svg_2d_plot, 216, 252, 253
 Class svg_boxplot, 268, 306, 307
y_label_text
 Class svg_boxplot, 268, 307
y_label_units
 Class svg_1d_plot, 179, 211
 Class svg_2d_plot, 216, 253, 254
 Class svg_boxplot, 268, 307
y_label_units_on
 Class svg_2d_plot, 216, 254
y_label_weight
 Class svg_2d_plot, 216, 254
y_label_width
 Class svg_2d_plot, 216, 254
y_major_grid_color
 Class svg_2d_plot, 216, 254, 255
y_major_grid_on
 Class svg_2d_plot, 216, 255
y_major_grid_width
 Class svg_2d_plot, 216, 255
y_major_interval
 Class svg_2d_plot, 216, 255
 Class svg_boxplot, 268, 307
y_major_labels_on
 Class svg_boxplot, 268, 307
y_major_labels_side
 Class svg_2d_plot, 216, 256
y_major_label_rotation
 Class svg_2d_plot, 216, 255
y_major_tick_color
 Class svg_2d_plot, 216, 256
 Class svg_boxplot, 268, 307
y_major_tick_length
 Class svg_2d_plot, 216, 256
 Class svg_boxplot, 268, 307
y_major_tick_width
 Class svg_2d_plot, 216, 256

Class `svg_boxplot`, 268, 308
`y_max`
 Class `svg_2d_plot`, 216, 256
`y_min`
 Class `svg_2d_plot`, 216, 256
`y_minor_grid_color`
 Class `svg_2d_plot`, 216, 257
`y_minor_grid_on`
 Class `svg_2d_plot`, 216, 257
`y_minor_grid_width`
 Class `svg_2d_plot`, 216, 257
`y_minor_interval`
 Class `svg_1d_plot`, 179, 212
 Class `svg_2d_plot`, 216, 257
 Class `svg_boxplot`, 268, 308
`y_minor_tick_color`
 Class `svg_2d_plot`, 216, 257
 Class `svg_boxplot`, 268, 308
`y_minor_tick_length`
 Class `svg_2d_plot`, 216, 257
 Class `svg_boxplot`, 268, 308
`y_minor_tick_width`
 Class `svg_2d_plot`, 216, 258
 Class `svg_boxplot`, 268, 308
`y_num_minor_ticks`
 Class `svg_2d_plot`, 216, 258
 Class `svg_boxplot`, 268, 308
`y_plusminus_color`
 Class `svg_2d_plot`, 216, 258
`y_plusminus_on`
 Class `svg_2d_plot`, 216, 258
`y_prefix`
 Class `svg_2d_plot`, 216, 258
`y_range`
 Class `svg_2d_plot`, 216, 258, 259
 Class `svg_boxplot`, 268, 308
`y_separator`
 Class `svg_2d_plot`, 216, 259
`y_size`
 1-D Data Values Examples, 43
 Class `svg`, 172, 178
 Class `svg_1d_plot`, 179, 188, 212
 Class `svg_2d_plot`, 216, 227, 259
 Class `svg_boxplot`, 268, 280, 308, 309
`y_suffix`
 Class `svg_2d_plot`, 216, 259
`y_ticks`
 Class `svg_2d_plot`, 216, 259
`y_ticks_left_on`
 Class `svg_2d_plot`, 216, 259
`y_ticks_on_window_or_axis`
 Class `svg_2d_plot`, 216, 259
`y_ticks_right_on`
 Class `svg_2d_plot`, 216, 260
`y_ticks_values_color`
 Class `svg_2d_plot`, 216, 260
`y_ticks_values_font_family`
 Class `svg_2d_plot`, 216, 260

y_ticks_values_font_size
 Class svg_2d_plot, 216, 260
 y_ticks_values_ioflags
 Class svg_2d_plot, 216, 260, 261
 y_ticks_values_precision
 Class svg_2d_plot, 216, 261
 y_values_color
 Class svg_2d_plot, 216, 261
 y_values_font_family
 Class svg_2d_plot, 216, 262
 y_values_font_size
 Class svg_2d_plot, 216, 262
 y_values_ioflags
 Class svg_2d_plot, 216, 262
 y_values_on
 Class svg_2d_plot, 216, 262
 y_values_precision
 Class svg_2d_plot, 216, 262
 y_values_rotation
 Class svg_2d_plot, 216, 263
 y_value_ioflags
 Class svg_2d_plot, 216, 261
 y_value_precision
 Class svg_2d_plot, 216, 261

Z

z
 Class path_element, 129, 133
Z
 Class path_element, 129, 133
z_path
 Struct z_path, 165

**Important**

This is not (yet) an official Boost library. It was a [Google Summer of Code project \(2007\)](#) whose mentor organization was Boost. It remains a library under construction, the code is quite functional, but interfaces, library structure, and names may still be changed without notice.

**Note**

Comments and suggestions (even bugs!) to Paul.A.Bristow (at) htp (dot) u-net (dot) com or Jake Voytko at jake-voytko (at) gmail (dot) com