
Plotting Graphs in SVG format.

Jake Voytko
Paul A. Bristow

Copyright © 2007 to 2018 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/quan/meas.hpp>	94
Header <boost/quan/meas2.hpp>	100
Header <boost/quan/pair_io.hpp>	101
Header <boost/quan/si_units.hpp>	101
Header <boost/quan/type_erasure_printer.hpp>	102
Header <boost/quan/unc.hpp>	107
Header <boost/quan/unc_init.hpp>	131
Header <boost/quan/uncdata.hpp>	131
Header <boost/quan/uncts.hpp>	132
Header <boost/quan/xiostream.hpp>	132
Header <boost/svg_plot/detail/auto_axes.hpp>	135
Header <boost/svg_plot/detail/axis_plot_frame.hpp>	144

Header <boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp>	145
Header <boost/svg_plot/detail/fp_compare.hpp>	147
Header <boost/svg_plot/detail/functors.hpp>	151
Header <boost/svg_plot/detail/numeric_limits_handling.hpp>	151
Header <boost/svg_plot/detail/pair.hpp>	151
Header <boost/svg_plot/detail/svg_boxplot_detail.hpp>	151
Header <boost/svg_plot/detail/svg_style_detail.hpp>	152
Header <boost/svg_plot/detail/svg_tag.hpp>	152
Header <boost/svg_plot/quantile.hpp>	210
Header <boost/svg_plot/show_1d_settings.hpp>	212
Header <boost/svg_plot/show_2d_settings.hpp>	213
Header <boost/svg_plot/svg_1d_plot.hpp>	214
Header <boost/svg_plot/svg_2d_plot.hpp>	311
Header <boost/svg_plot/svg_boxplot.hpp>	424
Header <boost/svg_plot/svg_color.hpp>	528
Header <boost/svg_plot/svg_style.hpp>	534
Header <boost/svg_plot/uncertain.hpp>	560
Using Inkscape	570
Implementation, History & Rationale	576
History	576
Compilers and Examples	576
Implementation and other notes	577
To Do List	582
Acknowledgements	583
Class Index	583
Function Index	586
Macro Index	621
typedef Index	622
Index	622

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 (&#x221A;)").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 (&#x221A;)").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 (&#x221A;)").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()

```

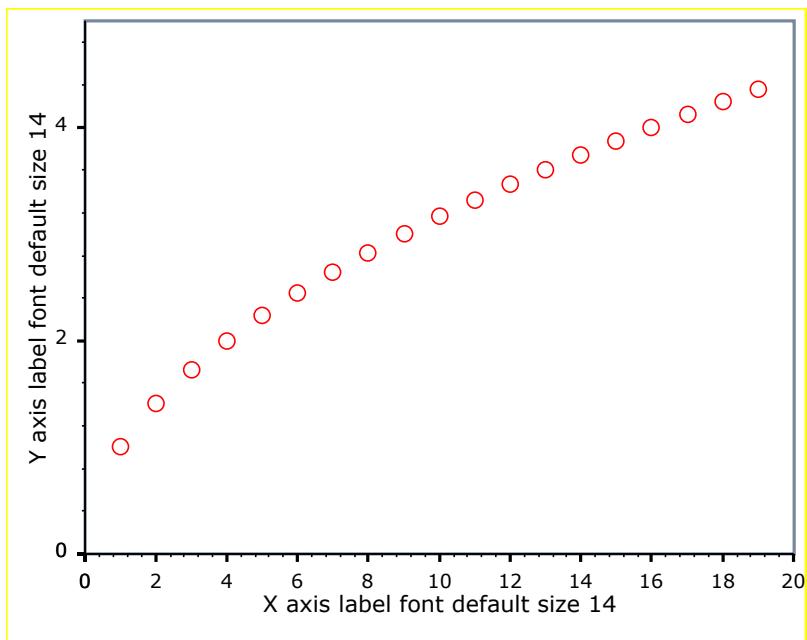
Output:

```

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.

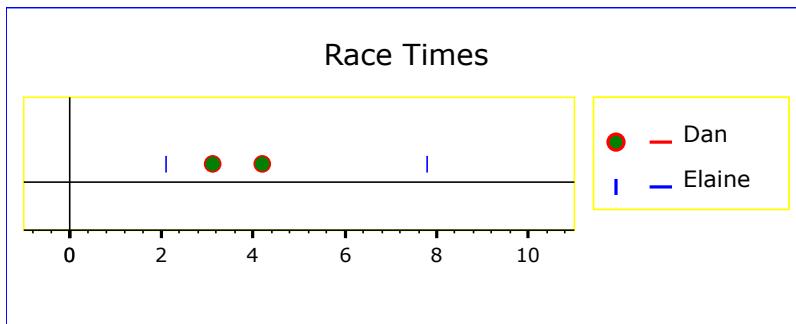
```
my_plot.plot(dan_times, "Dan").shape(circlet).size(10).stroke_color(red).fill_color(green);
my_plot.plot(elaine_times, "Elaine").shape(vertical_line).stroke_color(blue);
```

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");
```



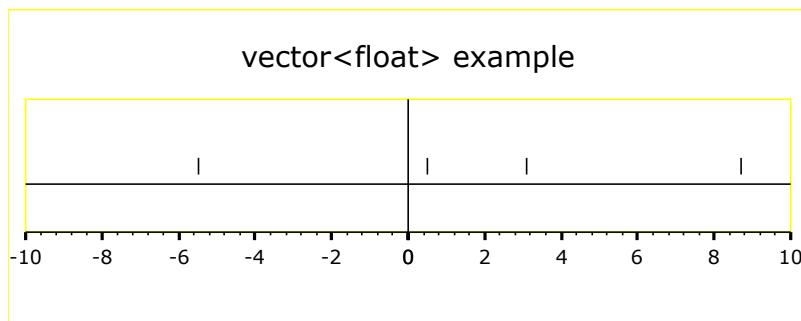
Note

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(), std::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);

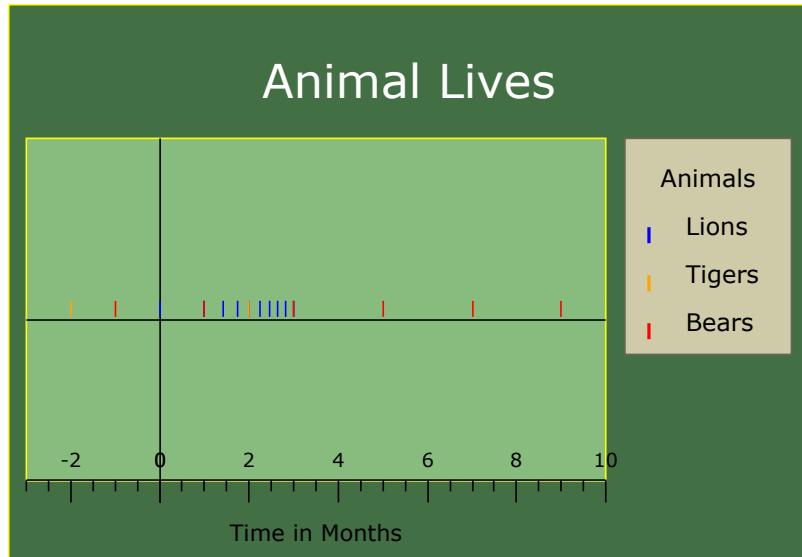
// Legend settings.
my_plot.legend_title_font_size(15);

// 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`:

```

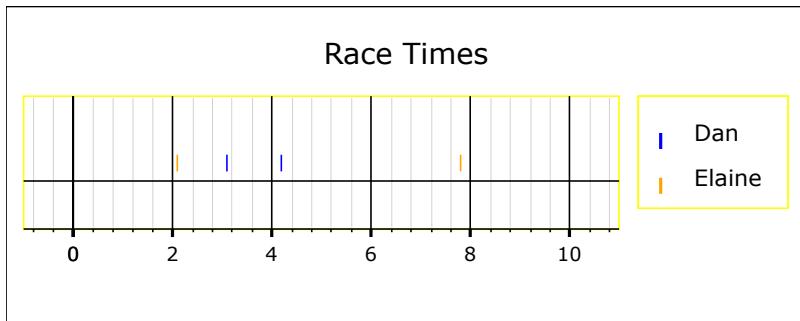
my_plot.x_major_grid_on(true)
    .x_minor_grid_on(true);

```

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.

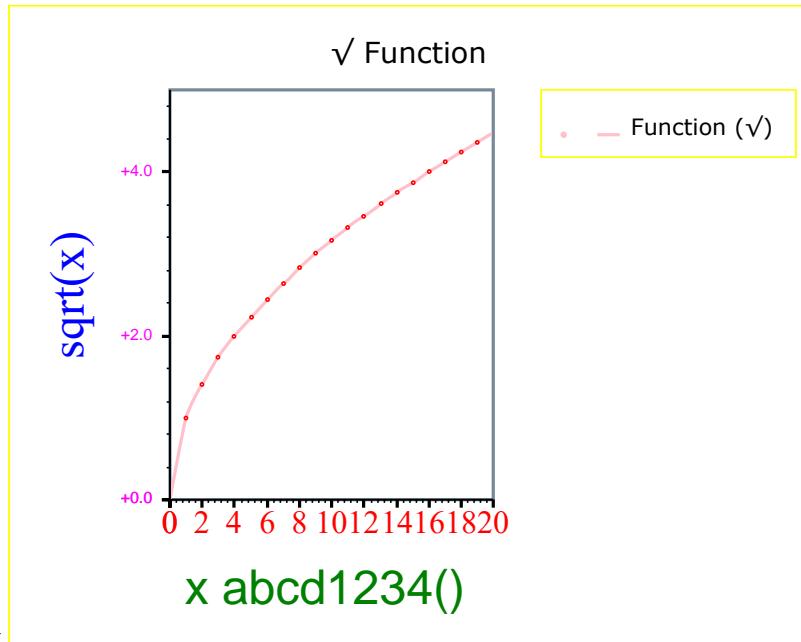
// Show the ticks styling:
// X ticks.
cout << "my_plot.x_ticks_values_color() " << my_plot.x_ticks_values_color() << endl;
cout << "my_plot.x_ticks_values_font_family() " << my_plot.x_ticks_values_font_family() << endl;
cout << "my_plot.x_ticks_values_font_size() " << my_plot.x_ticks_values_font_size() << 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;
}

```



producing this plot

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/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 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)

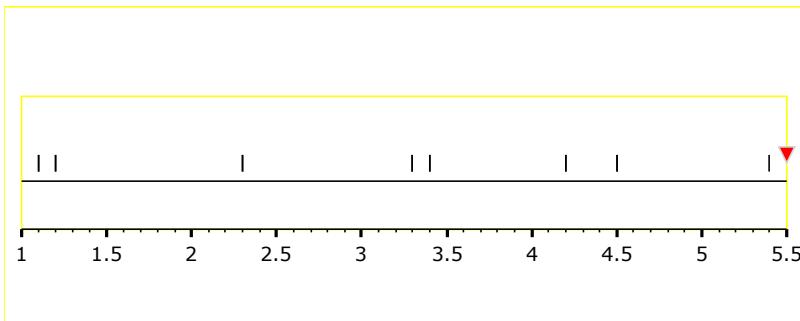
```

```

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;
}

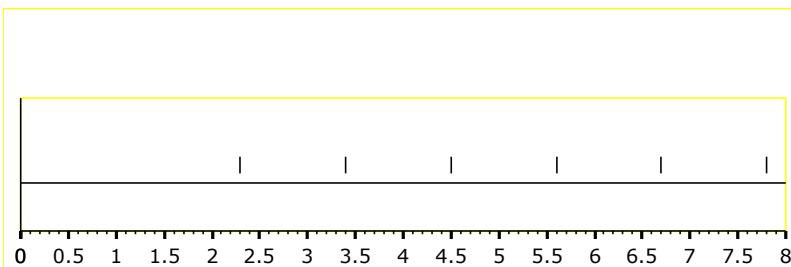
```

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 be 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:

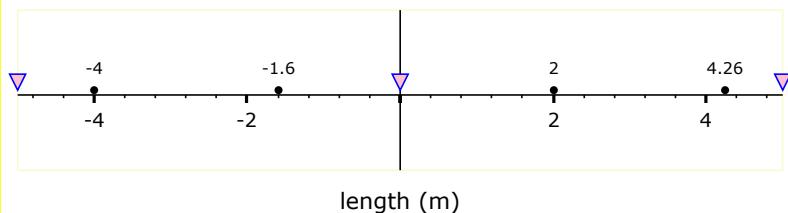
```
cout << "limit points stroke color " << my_1d_plot.limit_color() << endl;
cout << "limit points fill color " << my_1d_plot.limit_fill_color() << endl;
```

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");

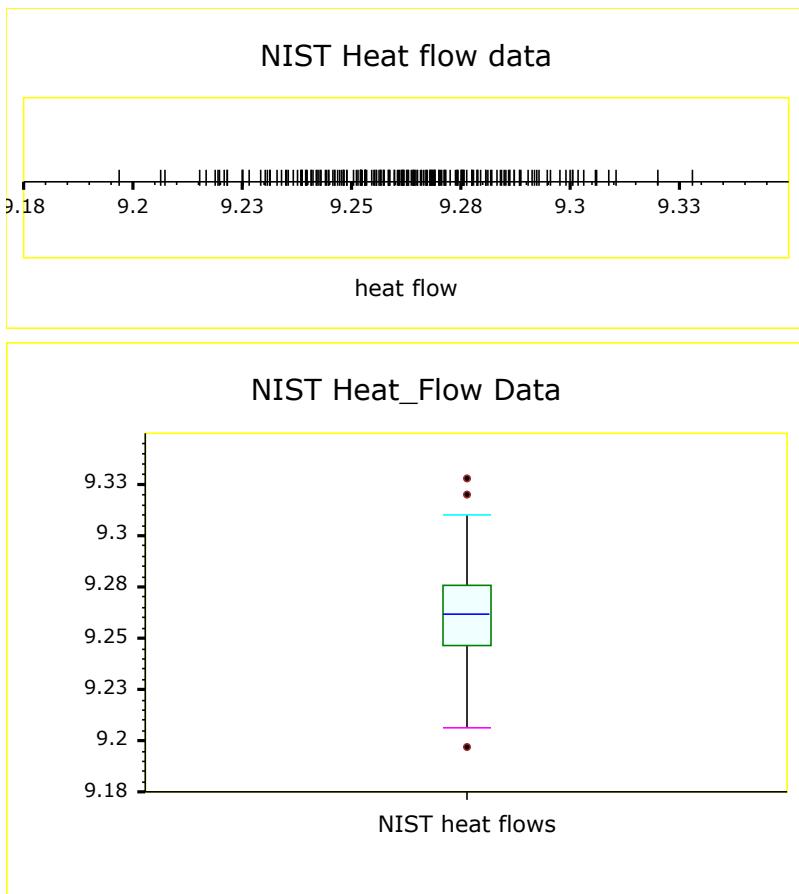
```

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.

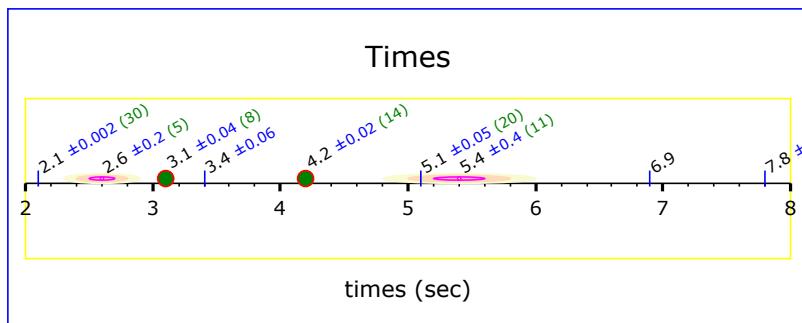
```
my_plot.plot(A_times, "A").shape(circlet).size(10).stroke_color(red).fill_color(green);
my_plot.plot(B_times, "B").shape(vertical_line).stroke_color(blue);
my_plot.plot(C_times, "C").shape(unc_ellipse).fill_color(lightyellow).stroke_color(magenta);
```

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);

```



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

If the plot does not meet your expectations, you could list all the hundreds of options with function `show_1d_settings`.

And if all else fails, defining the macro `BOOST_SVG_DIAGNOSIS` will trigger output of more information about which points are being ignored as being outside the plot window, a common cause of missing lines.

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.;

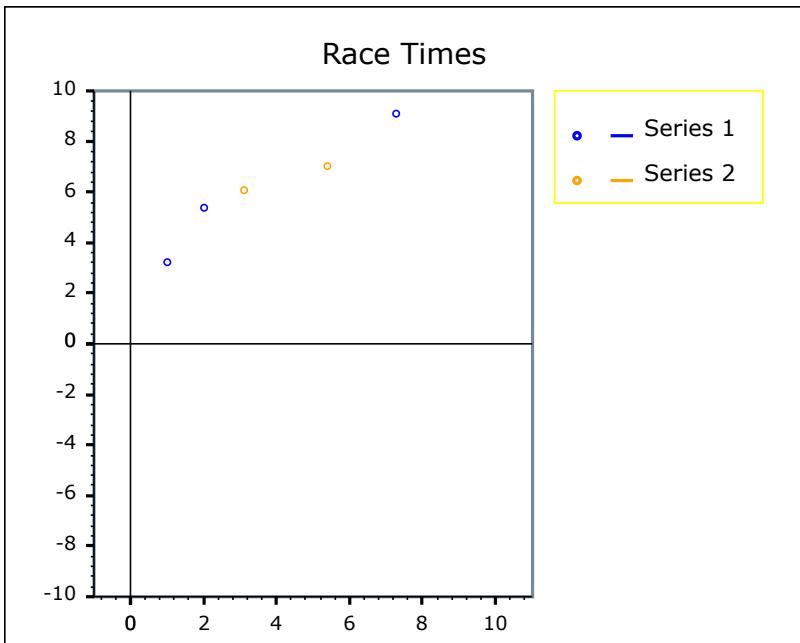
    svg_1d_plot my_plot;

    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.

The container type chosen above is a `std::map` used for type `double` here. The data type must be convertible to type `double`, so `float`, `double`, and `long double`, as well as some `_UDT` are suitable, but not `_multiprecision` types. If your data is one of these types, it must be converted to `double` before adding to the `std::map`.

Some demonstrations are at [convertible_to_double.cpp](#)

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);

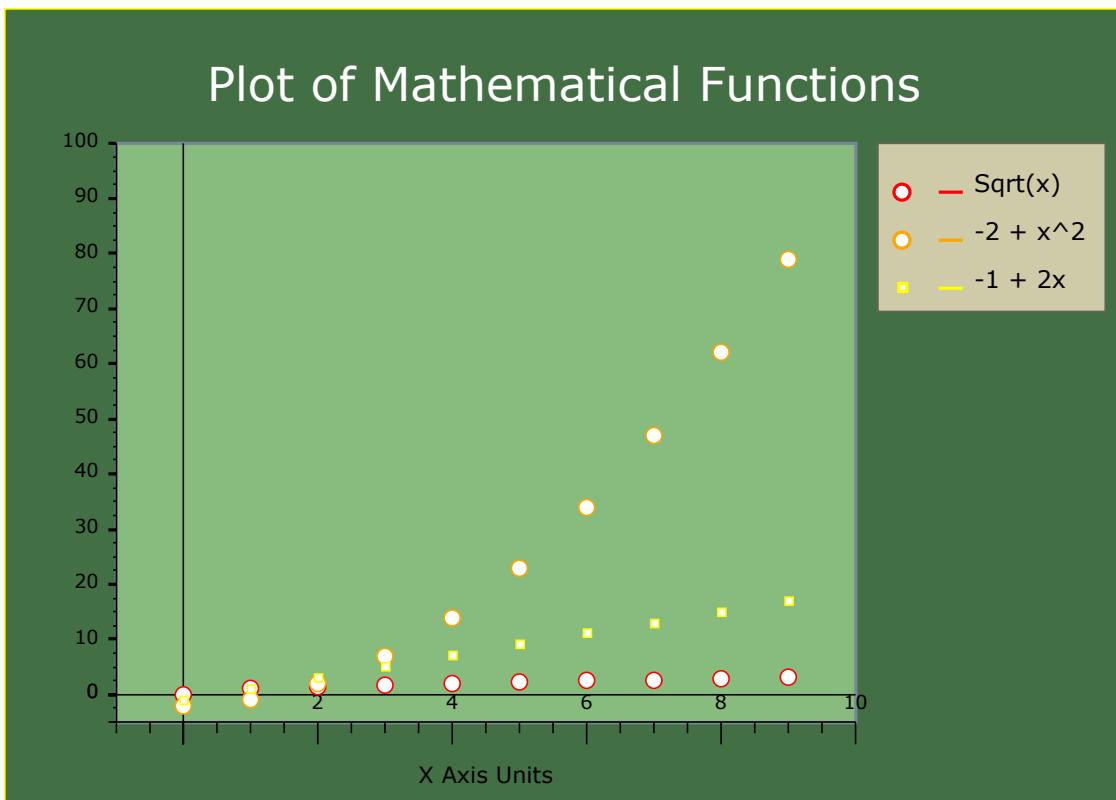
// Add the 3 data series to the plot, using different markers and line colors.
my_plot.plot(data1, "Sqrt(x)").fill_color(red);
my_plot.plot(data2, "-2 + x^2").fill_color(orange).size(5);
my_plot.plot(data3, "-1 + 2x").fill_color(yellow).bezier_on(true).line_color(blue).shape(square);
// Note how the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

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

return 0;
}

```

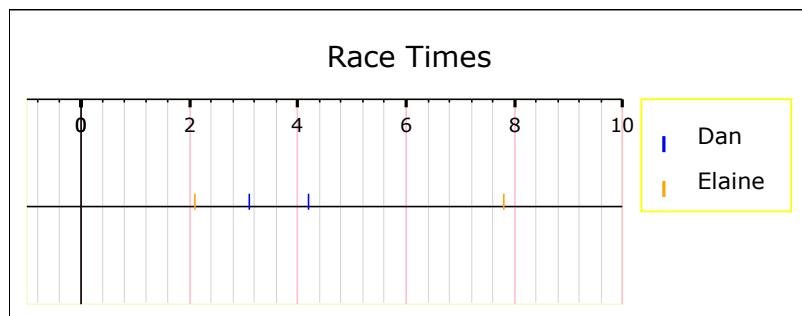
This produces the following image:



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;

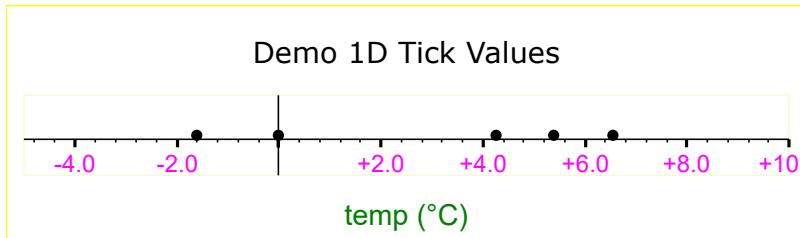
cout << "my_1d_plot.x_axis_label_color() " << my_1d_plot.x_axis_label_color() << endl;
cout << "my_1d_plot.x_label_font_family() " << my_1d_plot.x_label_font_family() << endl;
cout << "my_1d_plot.x_label_font_size() " << my_1d_plot.x_label_font_size() << endl;

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

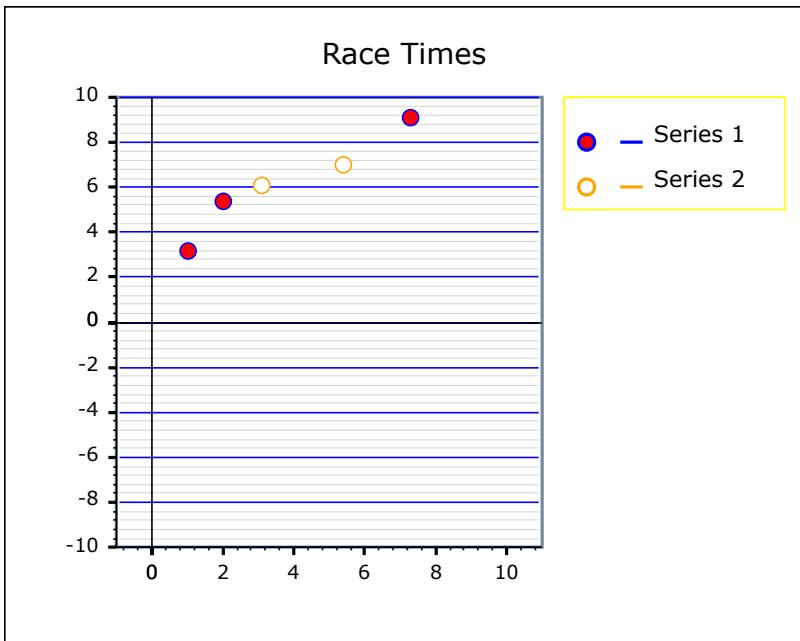
If you would like horizontal Y grid lines that go across the graph, you can make the following call to `svg_2d_plot`:

```
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)
```

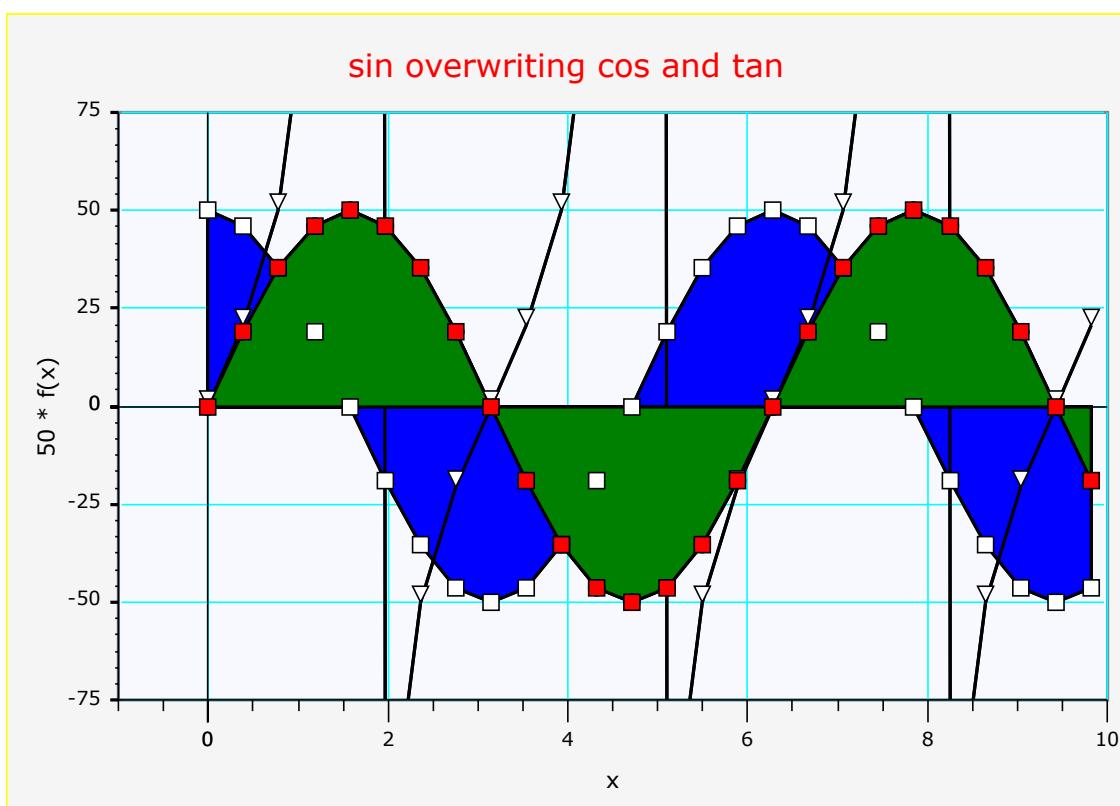
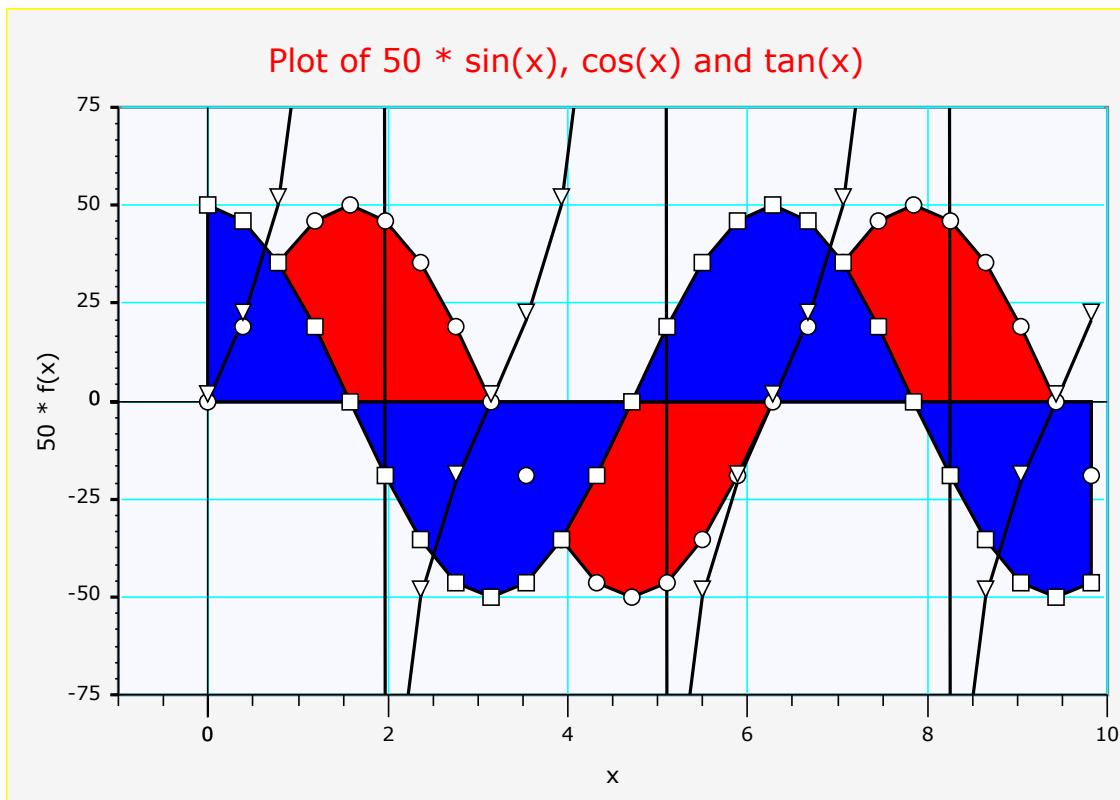
Fill the area between the plot and the axis

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:



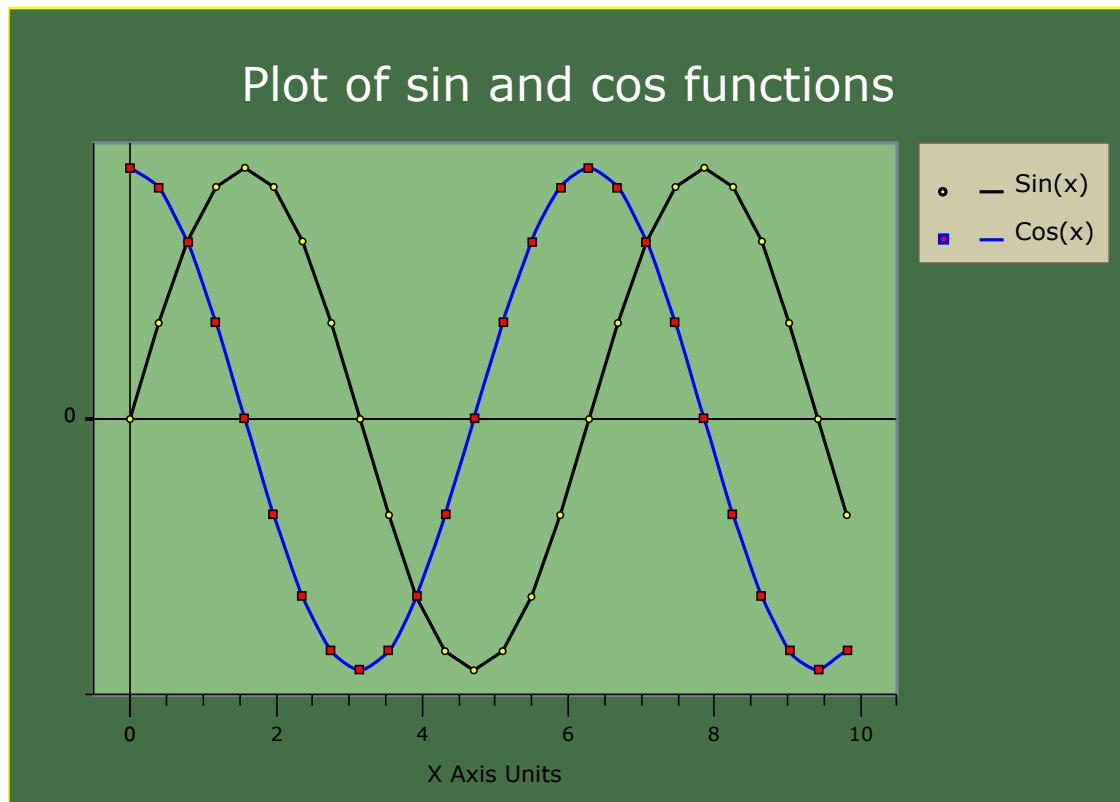
from source at [../../example/2d_area_fill.cpp](#)

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)`.)

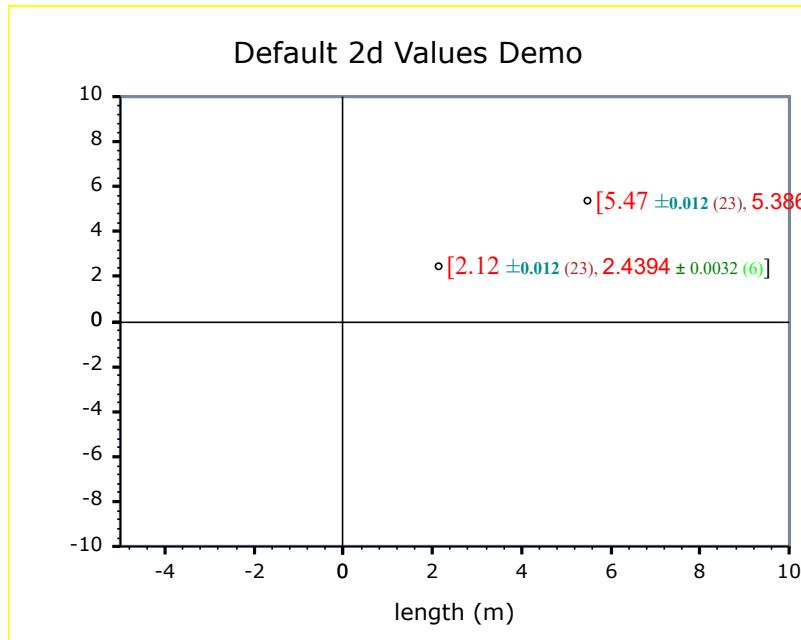
```
cout << "my_2d_plot.x_values_font_size() " << my_2d_plot.x_values_font_size() << endl;
cout << "my_2d_plot.x_values_font_family() " << my_2d_plot.x_values_font_family() << endl;
cout << "my_2d_plot.x_values_color() " << my_2d_plot.x_values_color() << endl;
cout << "my_2d_plot.x_values_precision()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

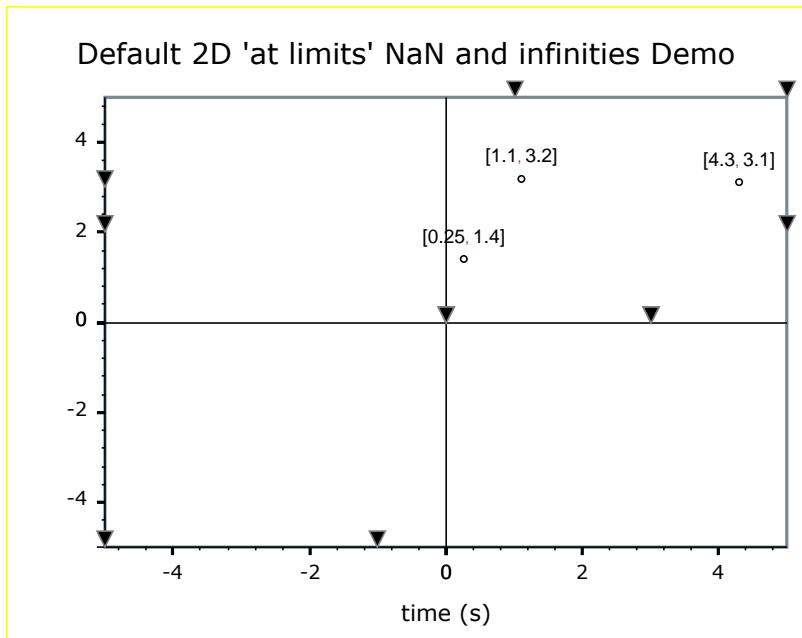
To echo the new marker colors chosen:

```
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.
```

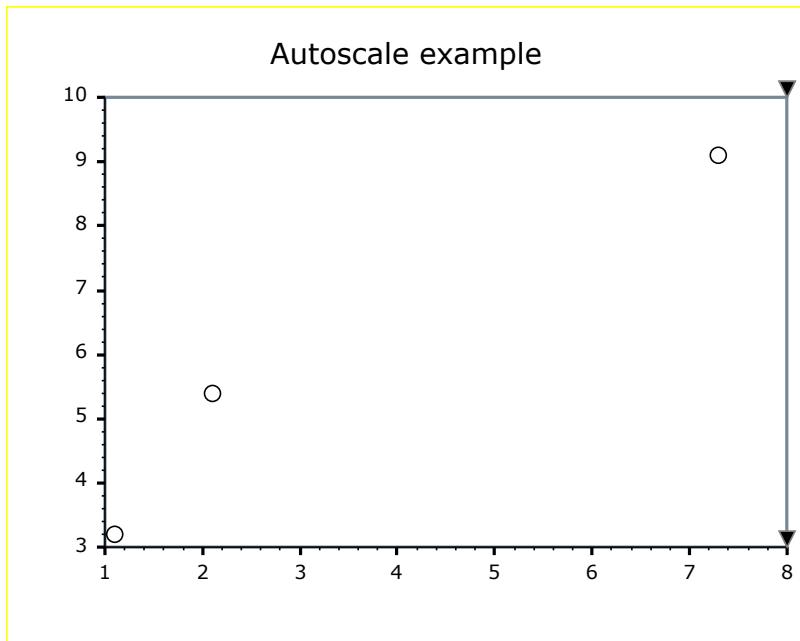
We can show the ranges chosen by autoscaling;

```
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

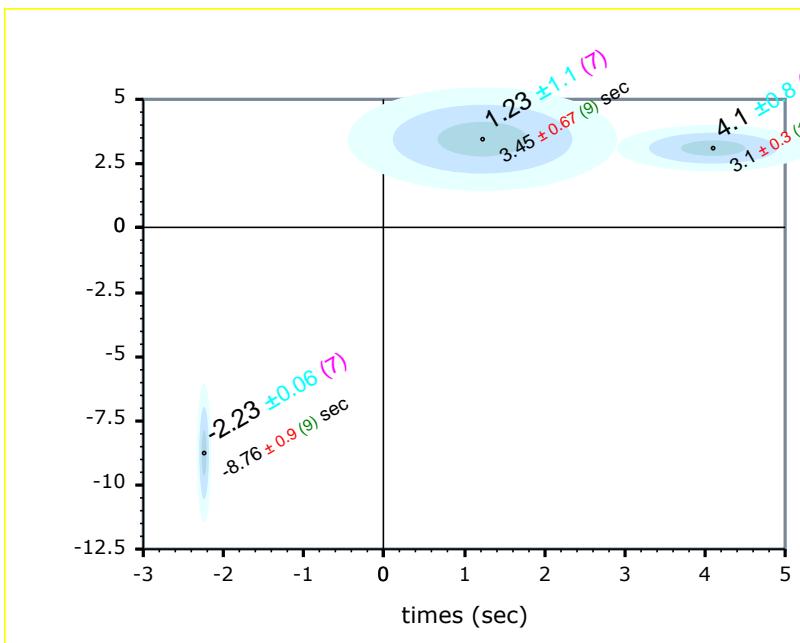
my_plot.plot(data1, "data1").shape(unc_ellipse);

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.

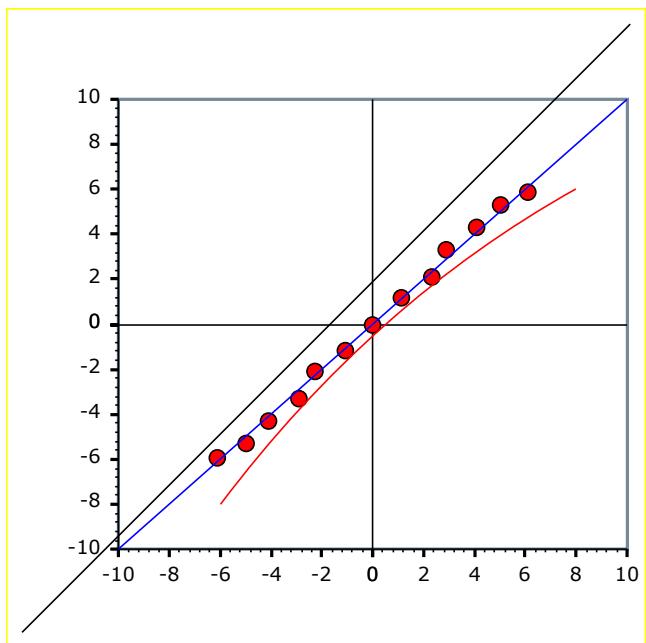


Note

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

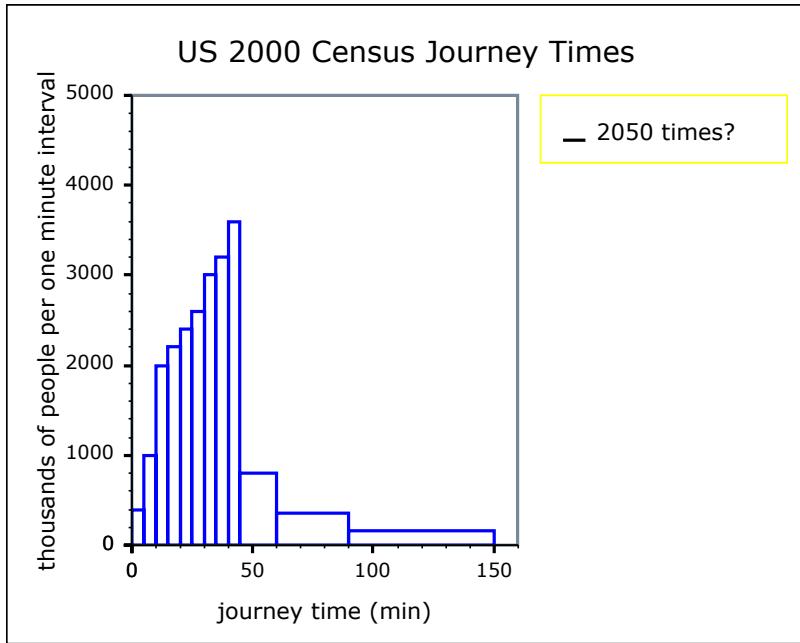
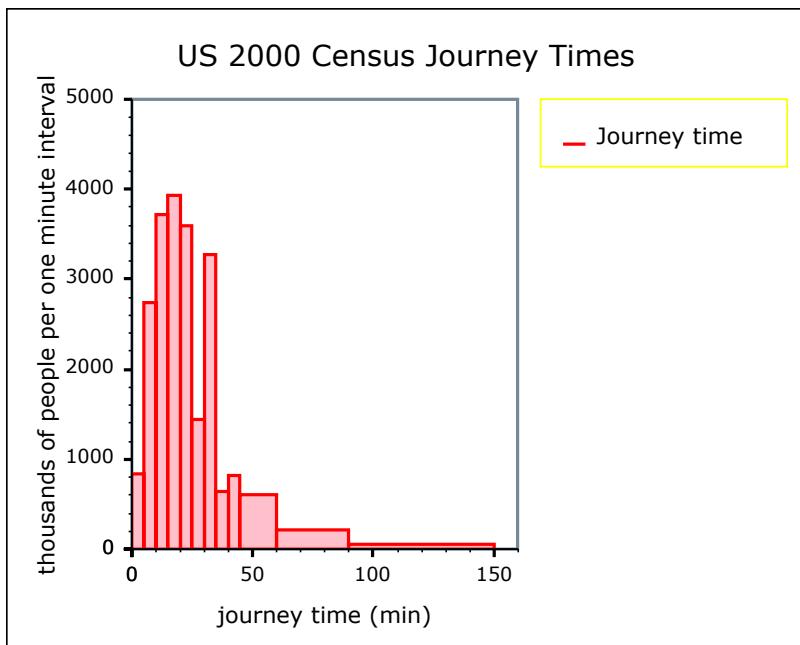


See [demo_2d_lines.cpp](#) for full source code and sample output.

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.
}
```

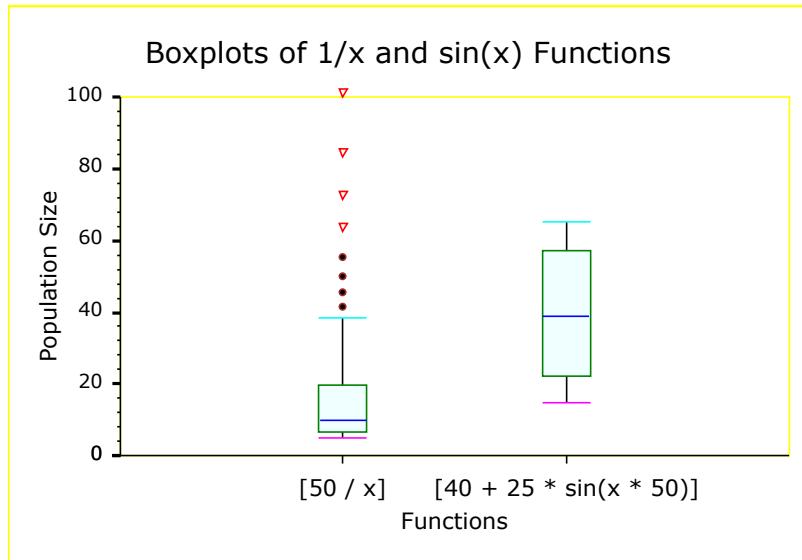
```

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.);
```

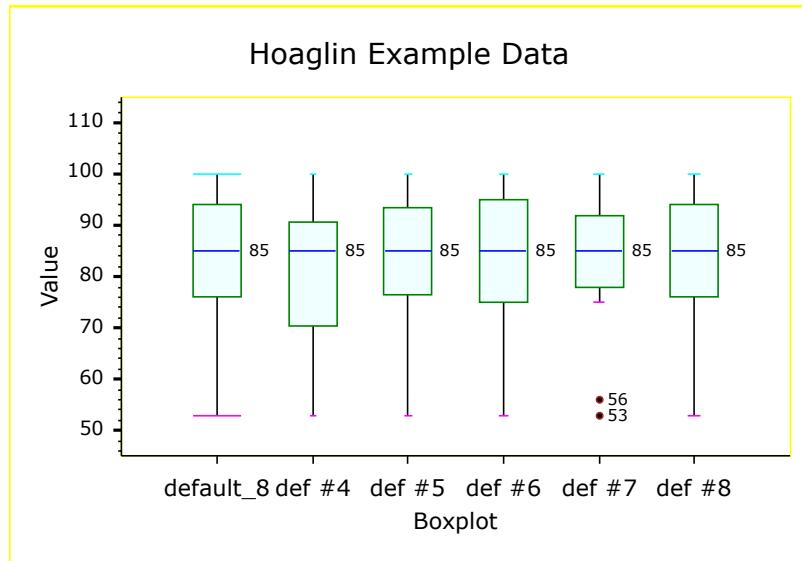
Write the entire SVG plot to a file.

```
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.

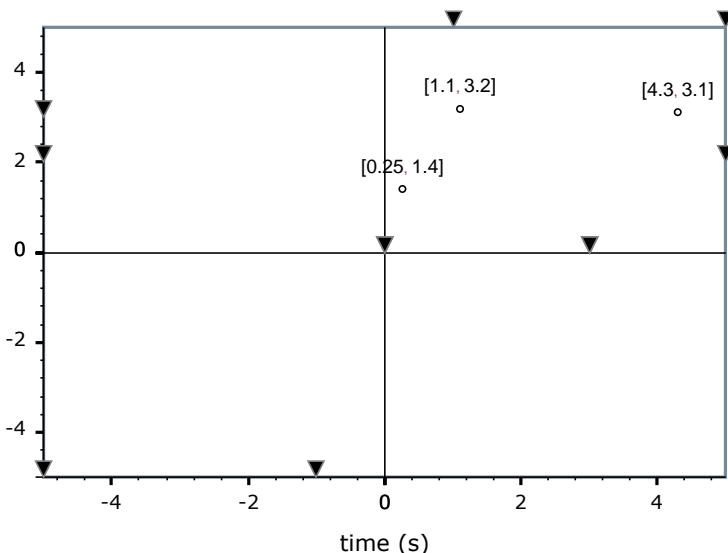
Negative Infinity

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()
```

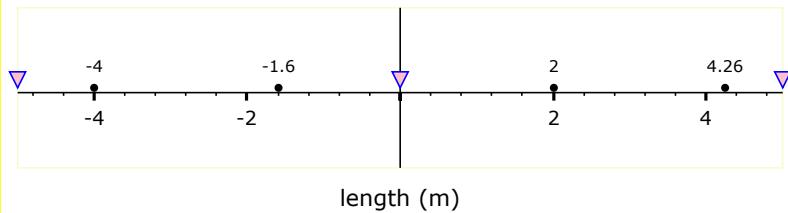
When plotted, these values will appear at the **bottom** of your plot window. If the plot window is not turned on, these points will appear at the bottom of the image.

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/quan/meas.hpp>

Class for measurement using uncertain class.

class meas with uncertain type UReal and measurement order & time-stamp.

```
class Meas;  
  
typedef unc< false > uncurl;  
ostream & operator<<(ostream & os, const Meas & m);  
istream & operator>>(istream & is, Meas & m);
```

Class Meas

Meas — Measured uncertain value AND its id and order and/or time-date stamp.

Synopsis

// In header: <boost/quan/meas.hpp>

```
class Meas : public unc< false > {
public:
    // construct/copy/destruct
    Meas(double const = 0.);
    Meas(uncun, string = "",
        boost::posix_time::ptime = (boost::posix_time::not_a_date_time),
        int = -1);
    Meas(const Meas &);
    Meas & operator=(const Meas &);
    ~Meas();

    // public member functions
    Meas & abs(const Meas &);

    Meas & abs(Meas &);

    unsigned short int deg_free();
    void deg_free(short unsigned int);
    unsigned short int degFree(void) const;
    float deviation(void) const;
    double mean(void) const;
    operator value_type() const;
    bool operator!=(const Meas &) const;
    bool operator!=(const unc< is_correlated > &) const;
    unc< is_correlated > & operator*=(const unc< is_correlated > &);

    unc< is_correlated > & operator*=(const double &);

    Meas operator+(const Meas &) const;
    unc< is_correlated > operator+(void) const;
    unc< is_correlated > operator++(void);
    unc< is_correlated > operator++(int);
    unc< is_correlated > & operator+=(const unc< is_correlated > &);

    unc< is_correlated > & operator+=(const double &);

    Meas operator-(void) const;
    Meas operator-(void);

    unc< is_correlated > operator--(void);
    unc< is_correlated > operator--(int);
    unc< is_correlated > & operator-=(const unc< is_correlated > &);

    unc< is_correlated > & operator-=(const double &);

    unc< is_correlated > & operator/=(const unc< is_correlated > &);

    unc< is_correlated > & operator/=(const double &);

    bool operator<(const Meas &) const;
    bool operator<(const unc< is_correlated > &) const;
    bool operator==(const Meas &) const;
    bool operator==(const unc< is_correlated > &) const;
    bool operator>(const Meas &) const;
    void setUncTypes(short unsigned int);
    float std_dev();
    void std_dev(float);
    unsigned short int types();
    void types(short unsigned int);
    unsigned short int uncFlags(void) const;
    double value();
    void value(double);

    // public static functions
    static bool earlier(const Meas &, const Meas &);

    static bool equal_toUnc(const Meas &, const Meas &);

    static bool equalU(const unc< is_correlated > &,
                     const unc< is_correlated > &);
```

```

static bool equalU2(const unc< is_correlated > &,
                   const unc< is_correlated > &);
static bool greater2U(const Meas &, const Meas &);
static bool greaterU(const Meas &, const Meas &);
static bool less(const Meas &, const Meas &);
static bool less2U(const Meas &, const Meas &);
static bool lessAbsM(const Meas &, const Meas &);
static bool lessU(const Meas &, const Meas &);
static bool lessU2(const unc< is_correlated > &,
                  const unc< is_correlated > &);
static bool moreU(const unc< is_correlated > &,
                  const unc< is_correlated > &);
static bool precedes(const Meas &, const Meas &);

// public data members
std::string id_;
int order_;
boost::posix_time::ptime time_;
};

```

Description

Meas public construct/copy/destruct

1. Meas(double const d = 0.);
2. Meas(uncun u, string id = "",
 boost::posix_time::ptime ti = (boost::posix_time::not_a_date_time),
 int o = -1);
3. Meas(const Meas &);
4. Meas & operator=(const Meas & rhs);

Assignment operator.

5. ~Meas();

Meas public member functions

1. Meas & abs(const Meas & rhs);
2. Meas & abs(Meas & rhs);
3. unsigned short int deg_free();

Returns: Estimate of number of degrees of freedom, usually = number of observations -1.

4. `void deg_free(short unsigned int df);`

Parameters: `df` Number of degrees of freedom, usually = number of observations -1.

5. `unsigned short int degFree(void) const;`

6. `float deviation(void) const;`

7. `double mean(void) const;`

8. `operator value_type() const;`

9. `bool operator!=(const Meas & p) const;`

10. `bool operator!=(const unc< is_correlated > & x) const;`

11. `unc< is_correlated > & operator*=(const unc< is_correlated > & ud);`

12. `unc< is_correlated > & operator*=(const double & a);`

13. `Meas operator+(const Meas &) const;`

14. `unc< is_correlated > operator+(void) const;`

Destructors. Two versions defined in unc.ippp to provide diagnostic output.

15. `unc< is_correlated > operator++(void);`

16. `unc< is_correlated > operator++(int);`

17. `unc< is_correlated > & operator+=(const unc< is_correlated > & ud);`

18. `unc< is_correlated > & operator+=(const double & a);`

19. `Meas operator-(void) const;`

20. `Meas operator-(void);`

21. `unc< is_correlated > operator--(void);`

22. `unc< is_correlated > operator--(int);`

23. `unc< is_correlated > & operator-=(const unc< is_correlated > & ud);`

24. `unc< is_correlated > & operator-=(const double & a);`

25. `unc< is_correlated > & operator/=(const unc< is_correlated > & ud);`

26. `unc< is_correlated > & operator/=(const double & a);`

`operator /=`



Note

Dividing by constant doesn't change degrees of freedom or uncertainty. All indicators & restrictions on value removed. Must assume double is irrational and not integer, NOT int/int becomes rational as operator /=.

27. `bool operator<(const Meas & rhs) const;`

28. `bool operator<(const unc< is_correlated > & x) const;`

29. `bool operator==(const Meas & p) const;`

30. `bool operator==(const unc< is_correlated > & x) const;`

31. `bool operator>(const Meas & rhs) const;`

32. `void setUncTypes(short unsigned int type);`

Parameters: type bits indicating type of uncertain real.

33. `float std_dev();`

Returns: Estimate of uncertainty as standard deviation of value of uncertain type.

34. `void std_dev(float unc);`

Parameters: unc Estimate of uncertainty as standard deviation of value of uncertain type.

35. `unsigned short int types();`

Returns: Types of uncertain real, encoded as a bitmap.

36. `void types(short unsigned int type);`

37. `unsigned short int uncFlags(void) const;`

38. `double value();`



Note

These sizes mean that total size of a unc is $64 = 32 + 32 = 128$ bits

Returns: Central estimate of value of uncertain type.

39. `void value(double value);`

Parameters: value Central estimate of value of uncertain type.

Meas public static functions

1. `static bool earlier(const Meas & l, const Meas & r);`

2. `static bool equal_toUnc(const Meas & l, const Meas & r);`

3. `static bool equalU(const unc< is_correlated > & l,
const unc< is_correlated > & r);`

4. `static bool equalU2(const unc< is_correlated > & l,
const unc< is_correlated > & r);`

5. `static bool greater2U(const Meas & l, const Meas & r);`

6. `static bool greaterU(const Meas & l, const Meas & r);`

7. static bool less(const Meas & l, const Meas & r);

8. static bool less2U(const Meas & l, const Meas & r);

9. static bool lessAbsM(const Meas & l, const Meas & r);

10. static bool lessU(const Meas & l, const Meas & r);

11. static bool lessU(const unc< is_correlated > & l,
 const unc< is_correlated > & r);

12. static bool lessU2(const unc< is_correlated > & l,
 const unc< is_correlated > & r);

13. static bool moreU(const unc< is_correlated > & l,
 const unc< is_correlated > & r);

14. static bool precedes(const Meas & l, const Meas & r);

Header <boost/quan/meas2.hpp>

Class for measurement using **two** uncertain class items.

class **meas2** with **two** uncertain type UReal and measurement order & time-stamp.

```
class meas2;  
ostream & operator<<(ostream & os, const meas2 & m);  
istream & operator>>(istream & is, meas2 & m);
```

Class **meas2**

meas2

Synopsis

```
// In header: <boost/quan/meas2.hpp>

class meas2 {
public:
    // construct/copy/destruct
    meas2(uncun uncn, uncn, string = "",
          boost::posix_time::ptime = (boost::posix_time::not_a_date_time),
          int = -1);

    // public data members
    std::string m_id;
    int m_order;
    boost::posix_time::ptime m_time;
    uncn m_uncx;
    uncn m_uncy;
};
```

Description

meas2 public construct/copy/destruct

1. meas2(uncn ux, uncn uy, string id = "",
 boost::posix_time::ptime ti = (boost::posix_time::not_a_date_time),
 int o = -1);

Header <boost/quan/pair_io.hpp>

Provide operator<< to output pair, surrounded by <> and separated by comma, for example: "<1.23, 4.56>". Typically used to show confidence intervals around a mean.

Polite to keep this a private detail namespace to avoid clash with other implementations!

Paul A. Bristow

Oct 2009, 2012

```
template<typename charT, typename traits, typename T1, typename T2>
std::basic_ostream<charT, traits> &
operator<<(std::basic_ostream<charT, traits> & os,
            const std::pair<T1, T2> & p);
template<typename charT, typename traits>
std::basic_ostream<charT, traits> &
operator<<(std::basic_ostream<charT, traits> & os,
            const std::pair<double, double> & p);
```

Header <boost/quan/si_units.hpp>

SI Unit, names and abbreviations & conversion factors.

See NIST 811 Special publication, Barry N Taylor (1995) "Guide for the Use of the International System of Units (SI) & ISO 31, a closely aligned document. NIST version contains more on non-SI units and many useful notes.

```

struct unit;
const unit &
findAnyUnit(const string, unsigned int &, unsigned int &, unsigned int &);
unsigned int
findUnit(const string, const unit *, unsigned int &, unsigned int &);
void showAllNonSIunits(const unit *, ostream &);
void showAllUnits(const unit *, ostream &);
void showNonSIunits(const unit *, ostream &);
void showSIunit(const unit *, ostream &);
void showSIunits(const unit *, ostream &);

```

Struct unit

unit

Synopsis

// In header: <[boost/quan/si_units.hpp](#)>

```

struct unit {

    // public data members
    const char * SIunitAbbrev;
    const char ** SIunitNames;
    unsigned char unitAbbrevCount;
    const int * unitLengths;
    const char *** unitNames;
    unsigned char unitNamesCount;
    const char * unitOf;
    unsigned char unitSINameCount;
    unsigned char unitsNamesCount;
    const double * unitToSIfactors;
};

```

Header <[boost/quan/type_erasure_printer.hpp](#)>

```

struct _iter;
struct _os;
struct _t;

template<typename CharT = char, typename Traits = std::char_traits<CharT> >
class abstract_printer;

template<typename T, typename U = _self> struct base_and_derived;

class decor_printer;namespace boost {
    namespace type_erasure {
        template<typename T, typename U, typename Base>
            struct concept_interface<base_and_derived< T, U >, Base, U>;
        template<typename Os, typename T1, typename T2>
            struct ostreamable<Os, std::pair< T1, T2 >>;
        template<typename T1, typename T2>
            std::ostream & operator<<(std::ostream &, const std::pair< T1, T2 > &);
    }
}

```

Struct _iter

_iter

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>
```

```
struct _iter : public placeholder {  
};
```

Struct _os

_os

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>
```

```
struct _os : public placeholder {  
};
```

Struct _t

_t

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>
```

```
struct _t : public placeholder {  
};
```

Class template abstract_printer

abstract_printer

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>

template<typename CharT = char, typename Traits = std::char_traits<char>>
class abstract_printer {
public:
    // construct/copy/destruct
    ~abstract_printer();

    // public member functions
    template<typename Range, typename CharT, typename Traits>
    void print(const Range &,
               std::basic_ostream<CharT, Traits> & os = std::cout) const;

    // protected member functions
    virtual void do_print(iterator_type first, iterator_type last, ostream_type os) const = 0;
};
```

Description

An abstract sequence printer - a 'template' that is inherited to implement the examples of actual printers defined below.

abstract_printer public construct/copy/destruct

1. `~abstract_printer();`

abstract_printer public member functions

1. `template<typename Range, typename CharT, typename Traits>
void print(const Range & r,
 std::basic_ostream<CharT, Traits> & os = std::cout) const;`

Parameters: `os` `std::ostream` for printing, for example `std::cout`.

`r` Container to be printed, for example, a C array, `std::vector`, `std::list` ...

Template Parameters: `Range` must be a Forward Range whose elements can be printed to `ostream os`.

abstract_printer protected member functions

1. `virtual void
do_print(iterator_type first, iterator_type last, ostream_type os) const = 0;`

Struct template base_and_derived

`base_and_derived`

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>

template<typename T, typename U = _self>
struct base_and_derived {
    // public static functions
    static T & apply(U &);

};
```

Description

base_and_derived public static functions

1. static T & apply(U & arg);

Class `decor_printer`

`decor_printer`

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>

class decor_printer : public abstract_printer<> {
public:
    // construct/copy/destruct
    explicit decor_printer(std::size_t = 0, std::size_t = 0);

    // public member functions
    void print(const Range &,
               std::basic_ostream<char, std::char_traits<char>> &= std::cout) const;

    // protected member functions
    virtual void do_print(iterator_type, iterator_type, ostream_type) const;
};
```

Description

Outputs items in a specified number of columns across rows with a separator (often comma and/or space(s)), and a suffix (usually newline) every `num_columns` items.

Usage: `decor_printer simple_printer(3, 0, "\n", ", ", "\n", "\n");`

Provide a container to be printed: `double da[] = {1., 2., 3., 4., 5., 6.};`

Print to `std::cout` using: `simple_printer.print(std::cout, da);`

Output:

1, 2, 3,
4, 5, 6,
7, 8, 9,
10, 11, 12

The order of parameters is chosen to try to allow use of the defaults as much as possible, including all defaults placing all items on one line or row separated by spaces.

decor_printer public construct/copy/destruct

1. `explicit decor_printer(std::size_t num_columns = 0, std::size_t wid = 0);`

Constructor.

Usage: for example: 3, 10, "double testd[] = {\n , , , "\n , "\n };\n" 3 columns with width 10, prefix "double testd[] = {\n ", separator string comma space, suffix (newline at the end of a column), and a terminator string "\n };\n".

Defaults are provided for all parameters, so can contruct: my_default_printer `decor_printer`; that places all items on one line or row with space between items, and a final newline.

decor_printer public member functions

1. `void print(const Range & r, std::basic_ostream< char, std::char_traits< char > > & os = std::cout) const;`

Parameters: os std::ostream for printing, for example std::cout.
 r Container to be printed, for example, a C array, std::vector, std::list ...

decor_printer protected member functions

1. `virtual void do_print(iterator_type first, iterator_type last, ostream_type os) const;`

Struct template concept_interface<base_and_derived< T, U >, Base, U>

`boost::type_erasure::concept_interface<base_and_derived< T, U >, Base, U>`

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>

template<typename T, typename U, typename Base>
struct concept_interface<base_and_derived< T, U >, Base, U> : public Base {

    // public member functions
    operator typename rebind_any< Base, const T & >::type() const;
    operator typename rebind_any< Base, T & >::type();
};
```

Description**concept_interface public member functions**

1. `operator typename rebind_any< Base, const T & >::type() const;`
2. `operator typename rebind_any< Base, T & >::type();`

Struct template ostreamable<Os, std::pair< T1, T2 >>

`boost::type_erasure::ostreamable<Os, std::pair< T1, T2 >>`

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>

template<typename Os, typename T1, typename T2>
struct ostreamable<Os, std::pair< T1, T2 >> {

    // public static functions
    static void apply(Os &, const std::pair< T1, T2 > &);

};
```

Description

ostreamable public static functions

1. static void apply(Os & out, const std::pair< T1, T2 > & arg);

Function template operator<<

boost::type_erasure::operator<<

Synopsis

```
// In header: <boost/quan/type_erasure_printer.hpp>

template<typename T1, typename T2>
std::ostream & operator<<(std::ostream & os, const std::pair< T1, T2 > & p);
```

Description

Output a pair of values with space as delimiter.

For example: "1.23 3.45".

Header <[boost/quan/unc.hpp](#)>

Class for simple Propagation of Uncertainties according to a pure Gaussian model.

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 at <http://www.ddj.com/cpp/184403147> . A fuller collection of even fancier classes also given in unc.h.

Standard deviation & its uncertainty added Paul A Bristow 31 Mar 98 to Dec 2013.

```

template<typename Type> struct lessAbs;

class resetMaskedUncFlags;
class resetUncFlags;
class setAllUncFlags;
class setConfidence;
class setMaskedUncFlags;
class setRoundingLoss;
class setScale;
class setSigDigits;
class setUncFlags;
class setUncSigDigits;
class setUncWidth;
class showUncFlags;
class showUncTypes;
template<bool is_correlated = false> class unc;
template<bool is_correlated = false> class unc;namespace std {
    template<typename Type> Type abs(const Type & a);
}

typedef unc< true > unccorr; // Uncertainties are NOT correlated. Uncorrelated is the normal case when uncertainties add.
typedef unc< false > uncurl;

// Add degrees of freedom as (99).
std::ios_base & adddegfree(std::ios_base &);

// Add lower and upper limits a 0.95 > 1.00 > 1.05.
std::ios_base & addlimits(std::ios_base &);

// Add an extra 'noisy' digit for less risk of loss.
std::ios_base & addnoisyDigit(std::ios_base &);

// Add SI prefix like kilo, micro ..
std::ios_base & addsiprefix(std::ios_base &);

// Add SI symbol like M, k, m.
std::ios_base & addsisymbol(std::ios_base &);
autoprefix_norm_impl< unc< false >, true >::type
autoprefix_norm(const unc< false > & arg);

// Automatically Scale to suitable prefix symbol like M, k, m...
std::ios_base & autoscale(std::ios_base &);

// Calculate sigdigits from uncertainty (default).
std::ios_base & autosigdigits(std::ios_base &);

// Calculate stdDev sig digits from uncertainty.
std::ios_base & autouncsigdigits(std::ios_base &);
BOOST_STATIC_ASSERT(std::numeric_limits< double >::is_iec559);
template<typename Type> Type cube(const Type & a);

// Firm fixed layout using setUncWidth.
std::ios_base & firmform(std::ios_base &);

// Flexible free format.
std::ios_base & flexform(std::ios_base &);
std::ios_base & noautoscale(std::ios_base &);

// Do not add degrees of freedom to output of value.
std::ios_base & nodegfree(std::ios_base &);

```

```

// Do not add lower and upper limits.
std::ios_base & nolimits(std::ios_base &);

// No noisy to suit human reading.
std::ios_base & nonoisyDigit(std::ios_base &);

// Do not add +/- uncertainty.
std::ios_base & noplusminus(std::ios_base &);

// or not scale.
std::ios_base & noscale(std::ios_base &);

// (Takes precedence over SI symbol if both set).
std::ios_base & nosiprefix(std::ios_base &);

// Do not add SI symbol like M, k, m.
std::ios_base & nosisymbol(std::ios_base &);

// Set acceptable loss due to rounding.
std::ostream operator<<(std::ostream, const showUncFlags &);
std::ostream operator<<(std::ostream, const setAllUncFlags &);
std::ostream operator<<(std::ostream, const setUncFlags &);
std::ostream operator<<(std::ostream, const setMaskedUncFlags &);
std::ostream operator<<(std::ostream, const resetUncFlags &);
std::ostream operator<<(std::ostream, const resetMaskedUncFlags &);
std::ostream operator<<(std::ostream, const setUncWidth &);
std::ostream operator<<(std::ostream, const setScale &);
std::ostream operator<<(std::ostream, const setUncSigDigits &);
std::ostream operator<<(std::ostream, const setRoundingLoss &);
std::ostream operator<<(std::ostream, const setConfidence &);
std::ostream operator<<(std::ostream, const setSigDigits &);

template<bool correlated>
std::ostream &
operator<<(std::ostream &,
            const std::pair<unc<correlated>, unc<correlated>> &);

std::istream operator>>(std::istream, const setAllUncFlags &);
std::istream operator>>(std::istream, const setUncFlags &);
std::istream operator>>(std::istream, const setMaskedUncFlags &);
std::istream operator>>(std::istream, const resetUncFlags &);
std::istream operator>>(std::istream, const resetMaskedUncFlags &);
std::istream operator>>(std::istream, const setUncWidth &);
std::istream operator>>(std::istream, const setScale &);
std::istream operator>>(std::istream, const setUncSigDigits &);
std::istream operator>>(std::istream, const setSigDigits &);

void outFpClass(double, std::ostream &);
void outIosFlags(long, std::ostream &);
void outUncFlags(long unclags, std::ostream);
void outUncTypes(unsigned short int, std::ostream &);
void outUncValues(std::ostream & os, std::ostream & log);

// Add +/- uncertainty.
std::ios_base & plusminus(std::ios_base &);

// Quaded function, notationally convenient for x^4, but overflow possible? Used by Welch-Satterthwaite formula.
template<typename Type> Type pow4(const Type & a);

// Reset/clear specific flags = 0.
long resetuFlags(std::ios_base &, long);
std::ios_base & scale(std::ios_base &);

// Use sig digits stored with << setSigDigits(6) for value.
std::ios_base & setsigdigits(std::ios_base &);

```

```

// Set specific flags = 1.
long setuFlags(std::ios_base &, long);
void setUncDefaults(std::ios_base & os);
template<typename Type> Type sqr(const Type & a);
template<typename Type> Type sqrtSumSqr(const Type & a, const Type & b);
long uFlags(std::ios_base &);

// Assigns all uncertain flags & returns previous.
long uFlags(std::ios_base &, long);

// Uncertainties ARE correlated. This is the unusual case where the sum must be exact, so the uncertainties are □
// subtracted. Example: two blocks which fit into a box perfectly. So if both have an uncertainties, they must cancel □
// when the uncertainties are added. Also applies to items like concentrations which must add up to 100% or unity.
void unc_input(double & mean, double & stdDev,
               unsigned short int & degreesOfFreedom,
               unsigned short int & uncTypes, std::istream & is);
template<typename T> double unc_of(T);
template<typename T> std::pair< double, double > uncs_of(std::pair< T, T >);
template<typename T1, typename T2>
std::pair< double, double > uncs_of(std::pair< T1, T2 >);

// Use stdDev sigDigits stored with << useSetUncSigDigits(2) ...
std::ios_base & uncsgidigits(std::ios_base &);

template<typename T> double value_of(T);
template<typename T> std::pair< double, double > values_of(std::pair< T, T >);
template<typename T1, typename T2>
std::pair< double, double > values_of(std::pair< T1, T2 >);

```

Struct template lessAbs

lessAbs

Synopsis

```

// In header: <boost/quan/unc.hpp>

template<typename Type>
struct lessAbs : public std::binary_function< Type, Type, bool > {

    // public member functions
    bool operator()(const Type &, const Type &) const;
};

```

Description

lessAbs public member functions

1. `bool operator()(const Type & a, const Type & b) const;`

Class resetMaskedUncFlags

resetMaskedUncFlags

Synopsis

// In header: <boost/quan/unc.hpp>

```
class resetMaskedUncFlags {
public:
    // construct/copy/destruct
    resetMaskedUncFlags(int, int);

    // public data members
    int flags;
    int mask;
};
```

Description

resetMaskedUncFlags public construct/copy/destruct

1. resetMaskedUncFlags(int, int);

Class resetUncFlags

resetUncFlags

Synopsis

// In header: <boost/quan/unc.hpp>

```
class resetUncFlags {
public:
    // construct/copy/destruct
    resetUncFlags(int);

    // public data members
    int flags;
};
```

Description

resetUncFlags public construct/copy/destruct

1. resetUncFlags(int);

Class setAllUncFlags

setAllUncFlags

Synopsis

// In header: <boost/quan/unc.hpp>

```
class setAllUncFlags {
public:
    // construct/copy/destruct
    setAllUncFlags(int);

    // public data members
    int flags;
};
```

Description

setAllUncFlags public construct/copy/destruct

1. setAllUncFlags(**int**);

Class setConfidence

setConfidence — **setConfidence(int setConfidence)**; to control the confidence interval. Usage: out << **setConfidence(0.01)** ...

Synopsis

// In header: <boost/quan/unc.hpp>

```
class setConfidence {
public:
    // construct/copy/destruct
    setConfidence(double);

    // public data members
    double confidence_;
};
```

Description

setConfidence public construct/copy/destruct

1. setConfidence(**double**);

Class setMaskedUncFlags

setMaskedUncFlags

Synopsis

```
// In header: <boost/quan/unc.hpp>

class setMaskedUncFlags {
public:
    // construct/copy/destruct
    setMaskedUncFlags(int, int);

    // public data members
    int flags;
    int mask;
};
```

Description

setMaskedUncFlags public construct/copy/destruct

1. setMaskedUncFlags(int, int);

Class setRoundingLoss

setRoundingLoss — `setRoundingLoss(int setRoundingLoss)`; to control the acceptable rounding loss. Usage: out << setRoundingLoss(0.01) ...

Synopsis

```
// In header: <boost/quan/unc.hpp>

class setRoundingLoss {
public:
    // construct/copy/destruct
    setRoundingLoss(double);

    // public data members
    double roundingloss_;
};
```

Description

setRoundingLoss public construct/copy/destruct

1. setRoundingLoss(double);

Class setScale

setScale

Synopsis

// In header: <boost/quan/unc.hpp>

```
class setScale {
public:
    // construct/copy/destruct
    setScale(int);

    // public data members
    int scale;
};
```

Description

setScale public construct/copy/destruct

1. setScale(**int**);

Class setSigDigits

setSigDigits

Synopsis

// In header: <boost/quan/unc.hpp>

```
class setSigDigits {
public:
    // construct/copy/destruct
    setSigDigits(int);

    // public data members
    int sigDigits_;
};
```

Description

setSigDigits public construct/copy/destruct

1. setSigDigits(**int**);

Class setUncFlags

setUncFlags

Synopsis

// In header: <boost/quan/unc.hpp>

```
class setUncFlags {
public:
    // construct/copy/destruct
    setUncFlags(int);

    // public data members
    int flags;
};
```

Description

setUncFlags public construct/copy/destruct

1. setUncFlags(int);

Class setUncSigDigits

setUncSigDigits

Synopsis

// In header: <boost/quan/unc.hpp>

```
class setUncSigDigits {
public:
    // construct/copy/destruct
    setUncSigDigits(int);

    // public data members
    int uncSigDigits_;
};
```

Description

setUncSigDigits(int uncSigDigits); Permits choice of number of uncertain or stddev value:

2 is the ISO recommendation.

Uncertainty of measurement – Part 3: Guide to the expression of uncertainty in measurement (GUM:1995)

ISO Guide 98 (1995) and updated version 2008.

1 is more appropriate for very small degrees of freedom, (but it gives a big step when the value starts with 1 or 2, when the difference between 1.4 (rounded to 1) and 1.6 (rounded to 2.) is a doubling. 3 is appropriate only for large degrees of freedoms, ≥ 1000 .



Warning

Values < 1 or > 3 are silently ignored. -1 passes through to allow dynamic choice based on degrees of freedom.
Usage: out << setUncSigDigits(3) ...

setUncSigDigits public construct/copy/destruct

1. setUncSigDigits(**int**);

Class setUncWidth

setUncWidth

Synopsis

// In header: <boost/quan/unc.hpp>

```
class setUncWidth {
public:
    // construct/copy/destruct
    setUncWidth(int);

    // public data members
    int uncWidth;
};
```

Description**setUncWidth public construct/copy/destruct**

1. setUncWidth(**int**);

Class showUncFlags

showUncFlags

Synopsis

// In header: <boost/quan/unc.hpp>

```
class showUncFlags {
public:
    // construct/copy/destruct
    showUncFlags(unsigned short);

    // friend functions
    friend std::ostream operator<<(std::ostream, const showUncFlags &);

    // public data members
    unsigned short int flags;
};
```

Description

showUncFlags public construct/copy/destruct

1. showUncFlags(unsigned short int);

showUncFlags friend functions

1. friend std::ostream operator<<(std::ostream, const showUncFlags &);

Set acceptable loss due to rounding.

Class showUncTypes

showUncTypes

Synopsis

```
// In header: <boost/quan/unc.hpp>

class showUncTypes {
public:
    // construct/copy/destruct
    showUncTypes(unsigned short);

    // public data members
    unsigned short int types;
};
```

Description

showUncTypes public construct/copy/destruct

1. showUncTypes(unsigned short int);

Class template unc

unc

Synopsis

```
// In header: <boost/quan/unc.hpp>

template<bool is_correlated = false>
class unc : public std::char_traits< char > {
public:
    // construct/copy/destruct
    unc(const double, const float = 0.0f, const short unsigned int = 1,
        const short unsigned int = UNC_KNOWN|UNC_EXPLICIT|DEG_FREE_EXACT|DEG_FREE_KNOWN);
    unc(const int = 0, const float = 0.0f, const short unsigned int = 1,
        const short unsigned int = DEG_FREE_DEF|VALUE_INTEGER|VALUE_RATION
AL|UNC_NOPLUS|UNC_NOMINUS|UNC_KNOWN);
    unc(const unc &);

    // friend functions
    friend void unc_input(double &, double &, unsigned short int &,
                          unsigned short int &, std::istream &);

    // public member functions
    unsigned short int deg_free();
    void deg_free(short unsigned int);
    unsigned short int degFree(void) const;
    float deviation(void) const;
    double mean(void) const;
    operator value_type() const;
    bool operator!=(const unc< is_correlated > &) const;
    unc< is_correlated > & operator*=(const unc< is_correlated > &);
    unc< is_correlated > & operator*=(const double &);
    unc< is_correlated > operator+(void) const;
    unc< is_correlated > operator++(void);
    unc< is_correlated > operator++(int);
    unc< is_correlated > & operator+=(const unc< is_correlated > &);
    unc< is_correlated > & operator+=(const double &);
    unc< is_correlated > operator-(void) const;
    unc< is_correlated > operator--(void);
    unc< is_correlated > operator--(int);
    unc< is_correlated > & operator-=(const unc< is_correlated > &);
    unc< is_correlated > & operator-=(const double &);
    unc< is_correlated > & operator/=(const unc< is_correlated > &);
    unc< is_correlated > & operator/=(const double &);
    bool operator<(const unc< is_correlated > &) const;
    bool operator==(const unc< is_correlated > &) const;
    void setUncTypes(short unsigned int);
    float std_dev();
    void std_dev(float);
    unsigned short int types();
    void types(short unsigned int);
    unsigned short int uncFlags(void) const;
    double value();
    void value(double);

    // public static functions
    static bool equalU(const unc< is_correlated > &,
                      const unc< is_correlated > &);
    static bool equalU2(const unc< is_correlated > &,
                      const unc< is_correlated > &);
    static bool lessU(const unc< is_correlated > &,
                     const unc< is_correlated > &);
    static bool lessU2(const unc< is_correlated > &,
                     const unc< is_correlated > &);
    static bool moreU(const unc< is_correlated > &,
```

```

const unc< is_correlated > &);

// public data members
short unsigned int degFree_;
float uncertainty_; // Standard deviation, if known. Reduced precision (float guarantees 6 decimal digits not □
15) and range e38 not E304 should not be a problem unless value is (near) less than 1e38. Can be zero, mean□
ing exact, and can be negative or anti-correlated, for example when values must add up to a total like 100%. Re□
lative (Fraction) Coefficient of variation = Standard deviation / value (aka % +|- /100) Relative is a problem if □
value near zero. +|- is std deviation, so for input of "1.0" implicit standard deviation and uncertainty limits +|- 0.05.
short unsigned int unctypes_; // Information about the value and uncertainty, encoded as a bitmap 16 bits, 0 □
to 15 See enum unc_types.,
double value_; // aka mean, estimate, or most likely value.
};

```

Description

Uncertain number template class unc, using mean and uncertainty (equivalent to std deviation if pure Gaussian), but also includes information about uncertainty as degrees of freedom & distribution.

Template Parameters

- `bool is_correlated = false`

if true, standard deviation is correlated, else not.

unc public construct/copy/destruct

- `unc(const double val, const float unc = 0.0f, const short unsigned int df = 1,`
`const short unsigned int uncTypeFlags = UNC_KNOWN|UNC_EXPLICIT|DEG_FREE_EXACT|DEG_FREE_KNOWN);`

Default constructor from double value & float uncertainty. Constructor from just a double value assumed exact, so behaves like a normal double. Normal conversion from double to unc. (See also constructor from integer which alone sets integer flag.)

- `unc(const int ivalue = 0, const float unc = 0.0f,`
`const short unsigned int df = 1,`
`const short unsigned int uncTypeFlags = DEG_FREE_DEF|VALUE_INTEGER|VALUE_RATION□`
`AL|UNC_NOPLUS|UNC_NOMINUS|UNC_KNOWN);`

< Constructor from integer value.

- `unc(const unc & ud);`

unc friend functions

- `friend void unc_input(double & mean, double & stdDev,`
`unsigned short int & degreesOfFreedom,`
`unsigned short int & types, std::istream & is);`

Uncertainties ARE correlated. This is the unusual case where the sum must be exact, so the uncertainties are subtracted. Example: two blocks which fit into a box perfectly. So if both have an uncertainties, they must cancel when the uncertainties are added. Also applies to items like concentrations which must add up to 100% or unity.

unc public member functions

- `unsigned short int deg_free();`

Returns: Estimate of number of degrees of freedom, usually = number of observations -1.

2. `void deg_free(short unsigned int df);`

Parameters: `df` Number of degrees of freedom, usually = number of observations -1.

3. `unsigned short int degFree(void) const;`

4. `float deviation(void) const;`

5. `double mean(void) const;`

6. `operator value_type() const;`

7. `bool operator!=(const unc< is_correlated > & x) const;`

8. `unc< is_correlated > & operator*=(const unc< is_correlated > & ud);`

9. `unc< is_correlated > & operator*=(const double & a);`

10. `unc< is_correlated > operator+(void) const;`

Destructors. Two versions defined in unc.cpp to provide diagnostic output.

11. `unc< is_correlated > operator++(void);`

12. `unc< is_correlated > operator++(int);`

13. `unc< is_correlated > & operator+=(const unc< is_correlated > & ud);`

14. `unc< is_correlated > & operator+=(const double & a);`

15. `unc< is_correlated > operator-(void) const;`

16. `unc< is_correlated > operator--(void);`

17. `unc< is_correlated > operator--(int);`

18. `unc< is_correlated > & operator=(const unc< is_correlated > & ud);`

19. `unc< is_correlated > & operator=(const double & a);`

20. `unc< is_correlated > & operator/=(const unc< is_correlated > & ud);`

21. `unc< is_correlated > & operator/=(const double & a);`

operator /=

Note



Dividing by constant doesn't change degrees of freedom or uncertainty. All indicators & restrictions on value removed. Must assume double is irrational and not integer, NOT int/int becomes rational as operator /=.

22. `bool operator<(const unc< is_correlated > & x) const;`

23. `bool operator==(const unc< is_correlated > & x) const;`

24. `void setUncTypes(short unsigned int type);`

Parameters: type bits indicating type of uncertain real.

25. `float std_dev();`

Returns: Estimate of uncertainty as standard deviation of value of uncertain type.

26. `void std_dev(float unc);`

Parameters: unc Estimate of uncertainty as standard deviation of value of uncertain type.

27. `unsigned short int types();`

Returns: Types of uncertain real, encoded as a bitmap.

28. `void types(short unsigned int type);`

29. `unsigned short int uncFlags(void) const;`

30. `double value();`



Note

These sizes mean that total size of a unc is $64 = 32 + 32 = 128$ bits

Returns: Central estimate of value of uncertain type.

31. `void value(double value);`

Parameters: `value` Central estimate of value of uncertain type.

unc public static functions

1. `static bool equalU(const unc< is_correlated > & l,
const unc< is_correlated > & r);`

2. `static bool equalU2(const unc< is_correlated > & l,
const unc< is_correlated > & r);`

3. `static bool lessU(const unc< is_correlated > & l,
const unc< is_correlated > & r);`

4. `static bool lessU2(const unc< is_correlated > & l,
const unc< is_correlated > & r);`

5. `static bool moreU(const unc< is_correlated > & l,
const unc< is_correlated > & r);`

unc public public data members

1. `short unsigned int degFree_;`

degrees of freedom, usually = number of observations; so for 2 observations assign 1 to degFree_ degree of freedom. Range from 1 (usually 1 observation) to $65534 = \text{std::numeric_limits<} \text{unsigned short int} \text{>} ::\text{max}() - 1$. Higher numbers of observations are indistinguishable from infinite observations. Max unsigned value 0xFFFF == 65535 is used to indicate degFree_ is NOT meaningful. Zero is as yet undefined? BUT many programs seem to use NON-integer degrees of freedom, so a float might seem better, but is 32 bits not 16, so use 16 for compact struct. (Might use an explicit 16 bit unsigned integer type?)

2. `double value_;`



Note

It is convenient to use 64-bit floating-point value (so even really accurate values like weights are OK), 32-bit floating-point is ample accuracy for standard deviation fractional variation, leaving two 16-bit for degrees of freedom and other flags, so that total is same as two doubles & can be efficiently aligned.

Class template unc

unc

Synopsis

```
// In header: <boost/quan/unc.hpp>

template<bool is_correlated = false>
class unc : public std::char_traits< char > {
public:
    // construct/copy/destruct
    unc(const double, const float = 0.0f, const short unsigned int = 1,
        const short unsigned int = UNC_KNOWN|UNC_EXPLICIT|DEG_FREE_EXACT|DEG_FREE_KNOWN);
    unc(const int = 0, const float = 0.0f, const short unsigned int = 1,
        const short unsigned int = DEG_FREE_DEF|VALUE_INTEGER|VALUE_RATION
    AL|UNC_NOPLUS|UNC_NOMINUS|UNC_KNOWN);
    unc(const unc &);

    // friend functions
    friend void unc_input(double &, double &, unsigned short int &,
        unsigned short int &, std::istream &);

    // public member functions
    unsigned short int deg_free();
    void deg_free(short unsigned int);
    unsigned short int degFree(void) const;
    float deviation(void) const;
    double mean(void) const;
    operator value_type() const;
    bool operator!=(const unc< is_correlated > &) const;
    unc< is_correlated > & operator*=(const unc< is_correlated > &);
    unc< is_correlated > & operator*=(const double &);
    unc< is_correlated > operator+(void) const;
    unc< is_correlated > operator++(void);
    unc< is_correlated > operator++(int);
    unc< is_correlated > & operator+=(const unc< is_correlated > &);
    unc< is_correlated > & operator+=(const double &);
    unc< is_correlated > operator-(void) const;
    unc< is_correlated > operator--(void);
    unc< is_correlated > operator--(int);
    unc< is_correlated > & operator-=(const unc< is_correlated > &);
    unc< is_correlated > & operator-=(const double &);
    unc< is_correlated > & operator/=(const unc< is_correlated > &);
    unc< is_correlated > & operator/=(const double &);
    bool operator<(const unc< is_correlated > &) const;
    bool operator==(const unc< is_correlated > &) const;
    void setUncTypes(short unsigned int);
    float std_dev();
    void std_dev(float);
    unsigned short int types();
    void types(short unsigned int);
    unsigned short int uncFlags(void) const;
    double value();
    void value(double);

    // public static functions
    static bool equalU(const unc< is_correlated > &,
                      const unc< is_correlated > &);
    static bool equalU2(const unc< is_correlated > &,
                      const unc< is_correlated > &);
    static bool lessU(const unc< is_correlated > &,
```

```

    const unc< is_correlated > &);
static bool lessU2(const unc< is_correlated > &,
                  const unc< is_correlated > &);
static bool moreU(const unc< is_correlated > &,
                 const unc< is_correlated > &);

// public data members
short unsigned int degFree_;
float uncertainty_; // Standard deviation, if known. Reduced precision (float guarantees 6 decimal digits not □
15) and range e38 not E304 should not be a problem unless value is (near) less than 1e38. Can be zero, mean□
ing exact, and can be negative or anti-correlated, for example when values must add up to a total like 100%. Re□
lative (Fraction) Coefficient of variation = Standard deviation / value (aka % +|- /100) Relative is a problem if □
value near zero. +|- is std deviation, so for input of "1.0" implicit standard deviation and uncertainty limits +|- 0.05.
short unsigned int unctypes_; // Information about the value and uncertainty, encoded as a bitmap 16 bits, 0 □
to 15 See enum unc_types.,
double value_; // aka mean, estimate, or most likely value.
};

```

Description

Uncertain number template class unc, using mean and uncertainty (equivalent to std deviation if pure Gaussian), but also includes information about uncertainty as degrees of freedom & distribution.

Template Parameters

1. `bool is_correlated = false`

if true, standard deviation is correlated, else not.

unc public construct/copy/destruct

1. `unc(const double val, const float unc = 0.0f, const short unsigned int df = 1,`
`const short unsigned int uncTypeFlags = UNC_KNOWN|UNC_EXPLICIT|DEG_FREE_EXACT|DEG_FREE_KNOWN);`

Default constructor from double value & float uncertainty. Constructor from just a double value assumed exact, so behaves like a normal double. Normal conversion from double to unc. (See also constructor from integer which alone sets integer flag.)

2. `unc(const int ivalue = 0, const float unc = 0.0f,`
`const short unsigned int df = 1,`
`const short unsigned int uncTypeFlags = DEG_FREE_DEF|VALUE_INTEGER|VALUE_RATION□`
`AL|UNC_NOPLUS|UNC_NOMINUS|UNC_KNOWN);`

< Constructor from integer value.

3. `unc(const unc & ud);`

unc friend functions

1. `friend void unc_input(double & mean, double & stdDev,`
`unsigned short int & degreesOfFreedom,`
`unsigned short int & types, std::istream & is);`

Uncertainties ARE correlated. This is the unusual case where the sum must be exact, so the uncertainties are subtracted. Example: two blocks which fit into a box perfectly. So if both have an uncertainties, they must cancel when the uncertainties are added. Also applies to items like concentrations which must add up to 100% or unity.

unc public member functions

1. `unsigned short int deg_free();`

Returns: Estimate of number of degrees of freedom, usually = number of observations -1.

2. `void deg_free(short unsigned int df);`

Parameters: df Number of degrees of freedom, usually = number of observations -1.

3. `unsigned short int degFree(void) const;`

4. `float deviation(void) const;`

5. `double mean(void) const;`

6. `operator value_type() const;`

7. `bool operator!=(const unc< is_correlated > & x) const;`

8. `unc< is_correlated > & operator*=(const unc< is_correlated > & ud);`

9. `unc< is_correlated > & operator*=(const double & a);`

10. `unc< is_correlated > operator+(void) const;`

Destructors. Two versions defined in unc.cpp to provide diagnostic output.

11. `unc< is_correlated > operator++(void);`

12. `unc< is_correlated > operator++(int);`

13. `unc< is_correlated > & operator+=(const unc< is_correlated > & ud);`

14. `unc< is_correlated > & operator+=(const double & a);`

15. `unc< is_correlated > operator-(void) const;`

16. `unc< is_correlated > operator--(void);`

17. `unc< is_correlated > operator--(int);`

18. `unc< is_correlated > & operator-=(const unc< is_correlated > & ud);`

19. `unc< is_correlated > & operator-=(const double & a);`

20. `unc< is_correlated > & operator/=(const unc< is_correlated > & ud);`

21. `unc< is_correlated > & operator/=(const double & a);`

`operator /=`



Note

Dividing by constant doesn't change degrees of freedom or uncertainty. All indicators & restrictions on value removed. Must assume double is irrational and not integer, NOT int/int becomes rational as operator /=.

22. `bool operator<(const unc< is_correlated > & x) const;`

23. `bool operator==(const unc< is_correlated > & x) const;`

24. `void setUncTypes(short unsigned int type);`

Parameters: `type` bits indicating type of uncertain real.

25. `float std_dev();`

Returns: Estimate of uncertainty as standard deviation of value of uncertain type.

26. `void std_dev(float unc);`

Parameters: `unc` Estimate of uncertainty as standard deviation of value of uncertain type.

27. `unsigned short int types();`

Returns: Types of uncertain real, encoded as a bitmap.

28. `void types(short unsigned int type);`

29. `unsigned short int uncFlags(void) const;`

30. `double value();`



Note

These sizes mean that total size of a unc is $64 = 32 + 32 = 128$ bits

Returns: Central estimate of value of uncertain type.

31. `void value(double value);`

Parameters: `value` Central estimate of value of uncertain type.

unc public static functions

1. `static bool equalU(const unc< is_correlated > & l,
const unc< is_correlated > & r);`

2. `static bool equalU2(const unc< is_correlated > & l,
const unc< is_correlated > & r);`

3. `static bool lessU(const unc< is_correlated > & l,
const unc< is_correlated > & r);`

4. `static bool lessU2(const unc< is_correlated > & l,
const unc< is_correlated > & r);`

5. `static bool moreU(const unc< is_correlated > & l,
const unc< is_correlated > & r);`

unc public public data members

1. `short unsigned int degFree_;`

degrees of freedom, usually = number of observations; so for 2 observations assign 1 to degFree_ degree of freedom. Range from 1 (usually 1 observation) to 65534 = std::numeric_limits<unsigned short int>::max() - 1 Higher numbers of observations are indistinguishable from infinite observations. Max unsigned value 0xFFFF == 65535 is used to indicate degFree_ is NOT meaningful. Zero is as yet undefined? BUT many programs seem to use NON-integer degrees of freedom, so a float might seem better, but is 32 bits not 16, so use 16 for compact struct. (Might use an explicit 16 bit unsigned integer type?)

2. `double value_;`



Note

It is convenient to use 64-bit floating-point value (so even really accurate values like weights are OK), 32-bit floating-point is ample accuracy for standard deviation fractional variation, leaving two 16-bit for degrees of freedom and other flags, so that total is same as two doubles & can be efficiently aligned.

Function template operator<<

operator<<

Synopsis

```
// In header: <boost/quan/unc.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) of uncertain values with (if defined) uncertainty and degrees of freedom.

For example: "1.23 +/- 0.01 (13), 3.45 +/- 0.06 (78)".

Function scale

scale — All ios_base functions declared below are defined in unc.hpp.

Synopsis

```
// In header: <boost/quan/unc.hpp>

std::ios_base & scale(std::ios_base &);
```

Description



Note

that all of these are entirely lower case names, like std::ios manipulators. To use value set with `setScale(int)`

Function uFlags

uFlags — Returns current uncertain flags.

Synopsis

```
// In header: <boost/quan/unc.hpp>

long uFlags(std::ios_base &);
```

Description

< Use stdDev sigDigits stored with << setconfidence(0.01) ... or alpha to control estimation of confidence interval. 0.01 means 99% confidence.

Functions to change uncertain flags on specified ios_base. Usage:

```
f = uFlags(out); f = uFlags(out, 0xFF);
f = setuFlags(out, 0xFF); f = resetuFlags(out, 0xFF);
```

Function template unc_of

unc_of

Synopsis

```
// In header: <boost/quan/unc.hpp>

template<typename T> double 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).
Returns:	zero	always (because no uncertainty information is available for built-in double, float, or long double).

Function template uncs_of

uncs_of — Get uncertainties (standard deviation) of a pair of values.

Synopsis

```
// In header: <boost/quan/unc.hpp>

template<typename T> std::pair< double, double > uncs_of(std::pair< T, T >);
```

Description

Template Parameters:	T	Built-in floating-point type, float, double, long double or unc or Meas.
		Built-in floating-point type or unc.

Builtin-floating point type or unc.
 Returns: uncertainty parts (if any) as a pair of floats.

Function template uncs_of

uncs_of

Synopsis

```
// In header: <boost/quan/unc.hpp>

template<typename T1, typename T2>
std::pair< double, double > uncs_of(std::pair< T1, T2 >);
```

Description

Function template value_of

value_of — Two helper functions to provide values and uncertainties as pairs.

Synopsis

```
// In header: <boost/quan/unc.hpp>

template<typename T> double value_of(T);
```

Description



Note

Names value_of and plural valueS_of. Allow value part of variables of class unc to be assigned to, and compared with double.

Two helper functions to provide values and uncertainties as pairs.

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.

Built-in floating-point type, float, double or long double, or uncertain type unc.

value type convertible to double.

Returns: value as a double.

Returns: value as a double.

Function template values_of

values_of

Synopsis

```
// In header: <boost/quan/unc.hpp>

template<typename T> std::pair< double, double > values_of(std::pair< T, T >);
```

Description

<

Template Parameters:

T	Built-in floating-point type, float, double, long double or unc or Meas .
	Built-infloating-point type, float, double, long double or unc.

Returns:

values of a pair of double values.

Function template values_of

values_of

Synopsis

```
// In header: <boost/quan/unc.hpp>

template<typename T1, typename T2>
std::pair< double, double > values_of(std::pair< T1, T2 >);
```

Description

Header <[boost/quan/unc_init.hpp](#)>

Initialisation of std::stream iword to store uncertainty information.

This sets the index values for data stored using the xalloc mechanism. This is included from unc.hpp and accessed from unc.ipp for many functions. Function setUncdefaults is used to set default values for these items for an iostream.

Header <[boost/quan/uncdata.hpp](#)>

Summary of uncFlagsIndex, unc_types and uncertainflags.

Header <boost/quan/uncs.hpp>

```
ios_base & addNoisyDigit(ios_base &);  
ios_base & addSIprefix(ios_base &);  
ios_base & addSIsymbol(ios_base &);  
ios_base & autoscale(ios_base &);  
ios_base & autosigfigs(ios_base &);  
ios_base & autouncsigfigs(ios_base &);  
ios_base & firmform(ios_base &);  
ios_base & flexform(ios_base &);  
ios_base & noautoscale(ios_base &);  
ios_base & noNoisyDigit(ios_base &);  
ios_base & noplusminus(ios_base &);  
ios_base & noscale(ios_base &);  
ios_base & noSIprefix(ios_base &);  
ios_base & noSIsymbol(ios_base &);  
ios_base & plusminus(ios_base &);  
long resetuFlags(ios_base &, long);  
ios_base & scale(ios_base &);  
ios_base & setsigdigits(ios_base &);  
long setuFlags(ios_base &, long);  
ios_base & setuncsigfigs(ios_base &);  
long uFlags(ios_base &);  
long uFlags(ios_base &, long);  
void uncertainPrint(double, float, unsigned short int, unsigned short int,  
                  ostream &);  
void uncertainRead(double & mean, double & stdDeviation,  
                  unsigned short int & degreesOfFreedom,  
                  unsigned short int & types, istream & is);
```

Header <boost/quan/xiostream.hpp>

Extra iostream manipulators.

Definitions in xiostream.cpp

Paul A. Bristow

Sep 2009

```
class chars;  
class setupperbase;  
class spaces;  
class stars;  
class tabs;  
void outFmtFlags(std::ios_base::fmtflags = std::cout.flags(),  
                 std::ostream & = std::cerr, const char * = ". ");  
void outFpClass(double value, std::ostream & os);  
void outIosFlags(long flags, std::ostream & os);  
void outIOStates(std::ios_base::iostate rdstate = std::cout.rdstate(),  
                 std::ostream & os = std::cerr, const char * term = ". ");  
std::ostream & showformat(std::ostream & os);  
std::ostream & showiostate(std::ostream & os);
```

Class chars

chars

Synopsis

```
// In header: <boost/quan/xiostream.hpp>
```

```
class chars {
public:
    // construct/copy/destruct
    chars(int, char);
};
```

Description

chars public **construct/copy/destruct**

1. `chars(int, char);`

Class setupperbase

setupperbase

Synopsis

```
// In header: <boost/quan/xiostream.hpp>
```

```
class setupperbase {
public:
    // construct/copy/destruct
    setupperbase(int);
};
```

Description

setupperbase public **construct/copy/destruct**

1. `setupperbase(int);`

Class spaces

spaces

Synopsis

```
// In header: <boost/quan/xiostream.hpp>
```

```
class spaces {
public:
    // construct/copy/destruct
    spaces(int);
};
```

Description

spaces public construct/copy/destruct

1. spaces(int);

Class stars

stars

Synopsis

```
// In header: <boost/quan/xiostream.hpp>

class stars {
public:
    // construct/copy/destroy
    stars(int);

    // public data members
    int num;
};
```

Description

stars public construct/copy/destruct

1. stars(int);

Class tabs

tabs

Synopsis

```
// In header: <boost/quan/xiostream.hpp>

class tabs {
public:
    // construct/copy/destroy
    tabs(int);
};
```

Description

tabs public construct/copy/destruct

1. tabs(int);

Function outFmtFlags

outFmtFlags

Synopsis

```
// In header: <boost/quan/xiostream.hpp>

void outFmtFlags(std::ios_base::fmtflags fmtFlags = std::cout.flags(),
                 std::ostream & os = std::cerr, const char * term = ".\n");
```

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");
```

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 *, int *,
        bool = false, double = 0., int = 6, int = 0);
    void scale_axis(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 *, 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 interator 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 autoscale_plusminus axis_max_value axis_min_value axis_tick_increment check_limits container min_ticks origin steps tight	Computed number of ticks, updated by scale_axis. Multiplier of uncertainty or standard deviations to allow for confidence ellipses. Computed maximum value for the axis, updated by scale_axis. Computed minimum value for the axis, updated by scale_axis. Computed tick increment for the axis, updated by scale_axis. Whether to check all values for infinity, NaN etc. STL container, usually of a data series. Minimum number of major ticks. If false, do not include the origin unless the range min_value <= 0 <= max_value. 0, or 2 for 2, 4, 6, 8, 10, 5 for 1, 5, 10, or 10 (2, 5, 10). 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 minimum 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`.

#define BOOST_SVG_LEGEND_DIAGNOSTICS for diagnostics of plot legend. #define BOOST_SVG_TITLE_DIAGNOSTICS for diagnostics of plot title. #define BOOST_SVG_POINT_DIAGNOSTICS for diagnostics of data point markers.

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,
            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,
            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,
            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/axis_plot_frame_14Aug18_1644_dud.hpp](#)>

SVG Plot functions common to 1D, 2D and Boxplots.

Functions are derived from base class `axis_plot_frame`.

Functions are derived from base class `axis_plot_frame`.

Jacob Voytko and Paul A. Bristow

```

namespace boost {
namespace svg {

enum legend_places { nowhere == 0, inside == -1, outside_left == 1,
                     outside_right == +2, outside_top == +3,
                     outside_bottom == +4, somewhere == +5,
                     nowhere == 0, inside == -1, outside_left == 1,
                     outside_right == +2, outside_top == +3,
                     outside_bottom == +4, somewhere == +5 };

enum x_axis_intersect { bottom == -1, x_intersects_y == 0, top == +1,
                        bottom == -1, x_intersects_y == 0, top == +1 };

enum y_axis_intersect { left == -1, y_intersects_x == 0, right == +1,
                        left == -1, y_intersects_x == 0, right == +1 };

static const double aspect_ratio; // Guess at average height to width of font. used to estimate the length of a title or header string from the font size. (This can only be approximate as varies on type of font (narrow or bold) and the mix of characters widths (unless monospace font).
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 spacing_factor; // Spacing factor to control the spacing of the legend data series descriptors.
static const double text_plusminus;
template<typename T> T maxof3(T a, T b, T c);
}
}

```

Global aspect_ratio

boost::svg::aspect_ratio — Guess at average height to width of font. used to estimate the length of a title or header string from the font size. (This can only be approximate as varies on type of font (narrow or bold) and the mix of characters widths (unless monospace font).

Synopsis

```
// In header: <boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp>

static const double aspect_ratio;
```

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_14Aug18_1644_dud.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_14Aug18_1644_dud.hpp>
static const double sin45;
```

Global spacing_factor

boost::svg::spacing_factor — Spacing factor to control the spacing of the legend data series descriptors.

Synopsis

```
// In header: <boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp>
static const double spacing_factor;
```

Global text_plusminus

boost::svg::text_plusminus

Synopsis

```
// In header: <boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.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 Douglis 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 ≠ 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 so higher precision data type will therefore lose information. This seems a reasonable design decision as any real data to be plotted is unlikely to have more than double precision (about 16 decimal digits).

"`svg_plot\example\convertible_to_double.cpp`" demonstrates that built-in types `float`, `double`, `long double` @ Boost.Multiprecision `cpp_bin_float_quad` work as expected, as well as a sample User Defined Type (UDT) a fixed-point type.

Types that cannot be converted to double nor constructible from double provoke a compile-time message: "Uncertain types must be convertible to double!"

(This uses checks using http://www.cplusplus.com/reference/type_traits/is_constructible/ `BOOST_STATIC_ASSERT_MSG(std::is_constructible<T, double>::value, "Uncertain types must be convertible to double!");`).

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_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.

< 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, double = 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, 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, double 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.

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/destruct
    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/destruct**

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 &rhs);
    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 &id);
    std::string class_id();
    void clip_id(const std::string &id);
    std::string clip_id();
    void id(const std::string &id);
    std::string id();
    bool operator!=(const svg_element &rhs);
    std::ostream &operator<<(std::ostream &os);
    bool operator==(const svg_element &rhs);
    polygon_element & P(double, double);
    svg_style & style();
    const svg_style & style() const;
    virtual void write(std::ostream &os);

    // protected member functions
    void write_attributes(std::ostream &os);

    // 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, double x6, double y6, 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 = 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);
    curve_element(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 std::string` of a `text_element`.

17. `text_style & textstyle();`

Get `text_element` textstyle for font size, family, decoration ...

18. `const text_style & textstyle() const;`

Get `text_element` textstyle for font size, family, decoration ...

19. `text_element & textstyle(text_style & ts);`

Set `text_element` 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 SVG `text_element`, style & attributes to a `std::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 constructor.

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(int);
    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;
    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(int size);`

font size of 1st single character of text string to use with SVG tspan command.

Returns: `tspan_element&` to make chainable.

15. `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_elements` for inequality, useful for Boost.Test.

24. `bool operator==(const svg_element & lhs);`

Compare `svg_elements`, 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's estimated length for a text string. This length may be used to expand or contract the SVG text to fit into this width, if `text_length_ == 0`, then has no effect as is not output by `write` below. If `< 0` would be an error if output like `textLength=-1`

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. `virtual void write(std::ostream & rhs);`

`write` functions output SVG commands.

Output SVG XML for a `tspan_element`

37. `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.

38. `double x();`

Get absolute X position for `tspan` element.

39. `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.

40 `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 &);
```

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!

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. Example:

```
#include <boost/svg_plot/show_2d_settings.hpp>
my_plot.show_2d_plot_settings();
```

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!

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_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;
        const std::string strip_e0s(std::string);
    }
}
```

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:
    // construct/copy/destruct
    svg_1d_plot();

    // public member functions
    bool autoscale();
    svg_1d_plot & autoscale(bool);
    bool autoscale();
    svg_1d_plot & autoscale(bool);
    svg_1d_plot & autoscale_check_limits(bool);
    bool autoscale_check_limits();
    svg_1d_plot & autoscale_check_limits(bool);
    bool autoscale_check_limits();
    svg_1d_plot & autoscale_plusminus(double);
    double autoscale_plusminus();
    svg_1d_plot & autoscale_plusminus(double);
    double autoscale_plusminus();
    svg_1d_plot & axes_on(bool);
    bool axes_on();
    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_color(const svg_color &);
    svg_color background_border_color();
    svg_1d_plot & background_border_width(double);
    double background_border_width();
    svg_1d_plot & background_border_width(double);
    double background_border_width();
    svg_color background_color();
    svg_1d_plot & background_color(const svg_color &);
    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 & boost_license_on(bool);
    bool boost_license_on();
    void calculate_plot_window();
    void calculate_transform();
    svg_1d_plot & confidence(double);
    double confidence();
    svg_1d_plot & confidence(double);
    double confidence();
    svg_1d_plot & coord_precision(int);
    int coord_precision();
    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_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 & copyright_holder(const std::string);
```

```

const std::string copyright_holder();
svg_1d_plot & data_lines_width(double);
double data_lines_width();
svg_1d_plot & data_lines_width(double);
double data_lines_width();
svg_1d_plot & derived();
const svg_1d_plot & derived() const;
svg_1d_plot & derived();
const svg_1d_plot & derived() const;
svg_1d_plot & description(const std::string);
const std::string & description();
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 & document_title(const std::string);
std::string document_title();
void draw_axes();
svg_1d_plot &
draw_line(double, double, double, const svg_color & = black);
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_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,
                const svg_color & = black);
svg_1d_plot &
draw_plot_curve(double, double, double, double, double,
                const svg_color & = black);
svg_1d_plot &
draw_plot_line(double, double, double, double, const svg_color & = black);
svg_1d_plot &
draw_plot_line(double, double, double, double, const svg_color & = black);
svg_1d_plot & image_border_margin(double);
double image_border_margin();
svg_1d_plot & image_border_margin(double);
double image_border_margin();
svg_1d_plot & image_border_width(double);
double image_border_width();
svg_1d_plot & image_border_width(double);
double image_border_width();
unsigned int image_x_size();
svg_1d_plot & image_x_size(unsigned int);
int image_x_size();
svg_1d_plot & image_x_size(int);
unsigned int image_y_size();
svg_1d_plot & image_y_size(unsigned int);
int image_y_size();
svg_1d_plot & image_y_size(int);
svg_1d_plot & legend_background_color(const svg_color &);
svg_color legend_background_color();
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();
svg_1d_plot & legend_border_color(const svg_color &);

```

```

svg_color legend_border_color();
const std::pair< double, double > legend_bottom_right();
const std::pair< double, double > legend_bottom_right();
bool legend_box_fill_on();
bool legend_box_fill_on();
svg_1d_plot & legend_color(const svg_color &);
svg_color legend_color();
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_family(const std::string &);
const std::string & legend_font_family();
svg_1d_plot & legend_font_size(int);
int legend_font_size();
svg_1d_plot & legend_font_weight(const std::string &);
const std::string & legend_font_weight();
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_lines(bool);
bool legend_lines();
svg_1d_plot & legend_on(bool);
bool legend_on();
svg_1d_plot & legend_on(bool);
bool legend_on();
bool legend_outside();
bool legend_outside();
svg_1d_plot & legend_place(legend_places);
legend_places legend_place();
svg_1d_plot & legend_place(legend_places);
legend_places legend_place();
svg_1d_plot & legend_text_font_size(int);
int legend_text_font_size();
svg_1d_plot & legend_text_font_weight(const std::string &);
const std::string & legend_text_font_weight();
svg_1d_plot & legend_title(const std::string);
const std::string legend_title();
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_title_font_size(int);
int legend_title_font_size();
svg_1d_plot & legend_title_font_weight(const std::string &);
const std::string & legend_title_font_weight();
svg_1d_plot & legend_top_left(double, double);
const std::pair< double, double > legend_top_left();
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 & 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");
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_attribution();
const std::string license_commercialuse();
const std::string license_commercialuse();
const std::string license_distribution();
const std::string license_distribution();
svg_1d_plot & license_on(bool);
bool license_on();
svg_1d_plot & license_on(bool);
bool license_on();
const std::string license_reproduction();
const std::string license_reproduction();
svg_1d_plot & limit_color(const svg_color &);
svg_color limit_color();
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 & 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();
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_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_color(const svg_color &);
    svg_color plot_border_color();
    svg_1d_plot & plot_border_width(double);
    double plot_border_width();
    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_on(bool);
    bool plot_window_on();
    svg_1d_plot & plot_window_x(double, double);
    std::pair< double, double > plot_window_x();
    svg_1d_plot & plot_window_x(double, double);
    std::pair< double, double > plot_window_x();
    double plot_window_x_left();
    double plot_window_x_left();
    double plot_window_x_right();
    double plot_window_x_right();
    svg_1d_plot & plot_window_y(double, double);
    std::pair< double, double > plot_window_y();
    svg_1d_plot & plot_window_y(double, double);

```

```

std::pair< double, double > plot_window_y();
double plot_window_y_bottom();
double plot_window_y_bottom();
double plot_window_y_top();
double plot_window_y_top();
void set_ids();
svg_1d_plot & size(unsigned int, unsigned int);
std::pair< double, double > size();
svg_1d_plot & size(int, int);
std::pair< double, double > size();
svg_1d_plot & three_sd_color(const svg_color &);
svg_color three_sd_color();
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(const std::string);
const std::string title();
svg_1d_plot & title_color(const svg_color &);
svg_color title_color();
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_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_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_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_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_size(int);
int title_font_size();
svg_1d_plot & title_font_stretch(const std::string &);
const std::string & title_font_stretch();
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_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_font_weight(const std::string &);
const std::string & title_font_weight();
svg_1d_plot & title_on(bool);
bool title_on();
svg_1d_plot & title_on(bool);
bool title_on();
text_style & title_style();
svg_1d_plot & title_text_length(double);
double title_text_length();
svg_1d_plot & two_sd_color(const svg_color &);
svg_color two_sd_color();

```

```

svg_1d_plot & two_sd_color(const svg_color &);
svg_color two_sd_color();
void update_image();
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_color(const svg_color &);
svg_color x_addlimits_color();
svg_1d_plot & x_addlimits_on(bool);
bool x_addlimits_on();
svg_1d_plot & x_addlimits_on(bool);
bool x_addlimits_on();
double x_auto_max_value();
double x_auto_max_value();
double x_auto_min_value();
double x_auto_min_value();
double x_auto_tick_interval();
double x_auto_tick_interval();
int x_auto_ticks();
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 &);
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_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_label_color(const svg_color &);
svg_color x_axis_label_color();
svg_1d_plot & x_axis_on(bool);
bool x_axis_on();
svg_1d_plot & x_axis_on(bool);
bool x_axis_on();
const std::string x_axis_position();
const std::string x_axis_position();
svg_1d_plot & x_axis_vertical(double);
bool x_axis_vertical();
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_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_color(const svg_color &);
svg_color x_datetime_color();
svg_1d_plot & x_datetime_on(bool);
bool x_datetime_on();
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_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_color(const svg_color &);  

svg_color x_df_color();  

svg_1d_plot & x_df_on(bool);  

bool x_df_on();  

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_color(const svg_color &);  

svg_color x_id_color();  

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_id_on(bool);  

bool x_id_on();  

svg_1d_plot & x_label(const std::string &);  

std::string x_label();  

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_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_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_font_size(int);  

int x_label_font_size();  

svg_1d_plot & x_label_on(bool);  

bool x_label_on();  

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(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_units_on(bool);  

bool x_label_units_on();  

svg_1d_plot & x_label_width(double);  

double x_label_width();  

svg_1d_plot & x_label_width(double);  

double x_label_width();  

svg_1d_plot & x_labels_strip_e0s(bool);  

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_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_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_grid_width(double);
double x_major_grid_width();
svg_1d_plot & x_major_interval(double);
double x_major_interval();
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_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_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(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_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_length(double);
double x_major_tick_length();
svg_1d_plot & x_major_tick_width(double);
double x_major_tick_width();
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_max(double);
double x_max();
svg_1d_plot & x_min(double);
double x_min();
svg_1d_plot & x_min(double);
double x_min();
svg_1d_plot & x_min_ticks(int);
int x_min_ticks();
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_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_on(bool);
bool x_minor_grid_on();
svg_1d_plot & x_minor_grid_width(double);
double x_minor_grid_width();
svg_1d_plot & x_minor_grid_width(double);
double x_minor_grid_width();
double x_minor_interval();
double x_minor_interval();
svg_1d_plot & x_minor_interval(double);
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_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_length(double);
double x_minor_tick_length();
svg_1d_plot & x_minor_tick_width(double);
double x_minor_tick_width();
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_num_minor_ticks(int);
int x_num_minor_ticks();
svg_1d_plot & x_order_color(const svg_color &);
svg_color x_order_color();
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_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_color(const svg_color &);
svg_color x_plusminus_color();
svg_1d_plot & x_plusminus_on(bool);
bool x_plusminus_on();
svg_1d_plot & x_plusminus_on(bool);
bool x_plusminus_on();
const std::string x_prefix();
const std::string x_prefix();
svg_1d_plot & x_range(double, double);
std::pair< double, double > x_range();
svg_1d_plot & x_range(double, double);
std::pair< double, double > x_range();
const std::string x_separator();
const std::string x_separator();
svg_1d_plot & x_size(unsigned int);
unsigned int x_size();
svg_1d_plot & x_size(int);
int x_size();
svg_1d_plot & x_steps(int);
int x_steps();
svg_1d_plot & x_steps(int);
int x_steps();
const std::string x_suffix();
const std::string x_suffix();
svg_1d_plot & x_ticks_down_on(bool);
bool x_ticks_down_on();
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_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_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_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_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_font_size(int);
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_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_ticks_values_precision(int);
int x_ticks_values_precision();
svg_1d_plot & x_tight(double);
double x_tight();
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_font_size(int);
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_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_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_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_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_font_size(int);
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_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_on(bool);
bool x_values_on();
svg_1d_plot & x_values_precision(int);
int x_values_precision();
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_values_rotation(rotate_style);
int x_values_rotation();
svg_1d_plot & x_with_zero(bool);
bool x_with_zero();
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_color(const svg_color &);
svg_color y_axis_color();
svg_1d_plot & y_axis_on(bool);
bool y_axis_on();
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(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_color(const svg_color &);
svg_color y_label_color();
svg_1d_plot & y_label_units(const std::string &);
std::string y_label_units();
svg_1d_plot & y_label_units(const std::string &);
std::string y_label_units();
bool y_labels_strip_e0s();
bool y_labels_strip_e0s();
double y_minor_interval();
double y_minor_interval();
unsigned int y_size();
svg_1d_plot & y_size(unsigned int);
int y_size();
svg_1d_plot & y_size(int);

// protected member functions
void adjust_limits(double &, double &);
void adjust_limits(double &, double &);
void clear_all();
void clear_all();
void clear_background();
void clear_background();
void clear_grids();
void clear_grids();
void clear_legend();
void clear_legend();
void clear_plot_background();
void clear_plot_background();
void clear_points();
void clear_points();
void clear_title();
void clear_title();
void clear_x_axis();
void clear_x_axis();
void clear_y_axis();
void clear_y_axis();
void draw_legend();
void draw_legend();
void draw_plot_point(double, double, g_element &, const plot_point_style &);
void draw_plot_point(double, double, g_element &, plot_point_style &,
                     unc< false >, unc< false >);
void draw_plot_point(double, double, g_element &, const plot_point_style &);
void draw_plot_point(double, double, g_element &, plot_point_style &,
                     unc< false >, unc< false >);
void draw_plot_point_value(double, double, g_element &, value_style &,
                           plot_point_style &, Meas);
void draw_plot_point_value(double, double, g_element &, value_style &,
                           plot_point_style &, Meas);
void draw_plot_point_values(double, double, g_element &, g_element &,
                           value_style &,
                           plot_point_style &, Meas);

```

```

        const value_style &, const value_style &, Meas,
        Meas);
void draw_plot_point_values(double, double, g_element &, g_element &,
                           const value_style &, const value_style &, Meas,
                           unc< false >);
void draw_plot_point_values(double, double, g_element &, g_element &,
                           const value_style &, const value_style &, Meas,
                           Meas);
void draw_plot_point_values(double, double, g_element &, g_element &,
                           const value_style &, const value_style &, Meas,
                           unc< false >);
void draw_title();
void draw_title();
void draw_x_axis();
void draw_x_axis();
void draw_x_axis_label();
void draw_x_axis_label();
void draw_x_major_tick(double, path_element &, path_element &);
void draw_x_major_tick(double, path_element &, path_element &);
void draw_x_minor_tick(double, path_element &, path_element &);
void draw_x_minor_tick(double, path_element &, path_element &);
void place_legend_box();
void place_legend_box();
void size_legend_box();
void size_legend_box();
void transform_point(double &, double &);
void transform_point(double &, double &);
void transform_x(double &);
void transform_x(double &);
void transform_y(double &);
void transform_y(double &);

// public data members
text_style a_style_; // Default text style that contains font size & type etc.
double alpha_;
bool autoscale_check_limits_; // If true, then check autoscale values for infinity, NaN, max, min.
double autoscale_plusminus_; // For uncertain values, allow for text_plusminus ellipses showing 67%, 95% □
and 99% confidence limits.

```

For example, if a max value is 1.2 +or- 0.02, then 1.4 will be used for autoscaling the maximum.

Similarly, if a min value is 1.2 +or- 0.02, then 1.0 will be used for autoscaling the minimum.

```

double epsilon_;
svg image_; // Stored so as to avoid rewriting style information constantly.
box_style image_border_; // rectangular border of all image width, color...
bool isNoisyDigit_;
double legend_bottom_; // bottom of legend box.
box_style legend_box_; // rectangular box of legend width, color...
double legend_height_; // Height of legend box (in pixels). Size of legend box is controlled by its contents, but □
helpful to store computed coordinates.
double legend_left_; // Left of legend box.
bool legend_lines_; // If true, include data colored line type in legend box.
size_t legend_longest_; // longest (both header & data) string in legend box,
bool legend_on_; // If true include a legend box.
legend_places legend_place_; // Place for any legend box.
double legend_right_; // SVG Coordinates of right of legend box,.
text_style legend_text_style_; // style (font etc of legend).
text_element legend_title_; // legend box header or title (if any).
double legend_top_; // Top of legend box.
double legend_width_; // Width of legend box (pixels).
bool outside_legend_on_; // If true, place legend box outside the plot window.
double plot_bottom_; // svg bottom of plot window (calculate_plot_window() sets these values).
double plot_left_; // svg left of plot window (calculate_plot_window() sets these values).
double plot_right_; // svg right of plot window (calculate_plot_window() sets these values).
double plot_top_; // svg top of plot window (calculate_plot_window() sets these values).
box_style plot_window_border_; // rectangular border of plot window width, color...
std::string plot_window_clip_; // = "clip_plot_window" id for clip□
path http://www.w3.org/TR/SVG/masking.html#ClipPathElement 14.1 Introduction clipping paths, aspect_ratioi□
ch uses any combination of 'path', 'text' and basic shapes to serve as the outline aspect_ratioere everything on □
the "inside" of the outline is allowed to show through but everything on the outside is masked out. So the plot_win□
dow_clip_limits display to a plot_window rectangle.
bool plot_window_on_; // Use a separate plot window (not aspect_ratioole image).
text_style point_symbols_style_; // Used for data point marking.
std::vector< svg_1d_plot_series > serieses_; // The (perhaps several) series of data points for transformation. □
These are sorted into two vectors for normal and abnormal (max, inf and NaN).
double text_margin_; // Marginal space around text items like title,.
double text_plusminus_;
text_element title_info_; // Title of aspect_ratioole plot.
bool title_on_; // If true include a title for the aspect_ratioole plot.
text_style title_style_; // style (font etc) of title.
int uncSigDigits_;
text_style value_style_; // Used for data point value label.
double x_auto_max_value_; // X maximum value calculated by autoscaling.
double x_auto_min_value_; // X minimum value calculated by autoscaling.
double x_auto_tick_interval_; // X axis tick major interval.
int x_auto_ticks_; // Number of X axis ticks.
bool x_autoscale_; // If true, use any autoscale values for scaling the X axis.
axis_line_style x_axis_; // style of X axis line.
text_style x_axis_label_style_; // style of X axis label.
int x_axis_position_;
double x_axis_vertical_; // Vertical position of 1D horizontal X-axis line as fraction of window. 0.5 is at middle(use□
ful if no labels) (default), 0.8 is near bottom (useful if value labels go upward), 0.2 is near top (useful if value la□
bels go downward).
bool x_include_zero_; // If autoscaled, include zero.
text_element x_label_info_; // X-axis label, Example: "length of widget".
int x_min_ticks_; // If autoscaled, set a minimum number of ticks.
double x_scale_; // Scale used for transform from Cartesian to SVG coordinates.
double x_shift_; // Shift from SVG origin is top left, Cartesian is bottom right.
int x_steps_; // If autoscaled, set any prescaling to decimal 1, 2, 5, 10 etc.
ticks_labels_style x_ticks_; // style of X axis tick value labels.
bool x_ticks_on_; // Ticks on X axis will be shown.
double x_tight_; // How much a value can go beyond the tick value before another tick is required.
text_element x_units_info_; // For example, to display, "length (meter)".

```

```

text_element x_value_label_info_; // X-axis tick value label, for example: "1.2" or "1.2e1".
text_style x_value_label_style_; // style of X ticks value label.
bool x_values_on_; // values of data are shown by markers.
value_style x_values_style_; // Used for data point value marking.
bool y_autoscale_; // Always false for 1-D plot because Y axis is not autoscaled.
axis_line_style y_axis_; // style of Y axis line.
text_style y_axis_label_style_; // Not used for 1D but needed by axis_plot_frame.hpp.
double y_scale_; // Scale used for transform from Cartesian to SVG coordinates.
double y_shift_; // Shift from SVG origin is top left, Cartesian is bottom right.
ticks_labels_style y_ticks_; // style of Y axis tick value labels. (Meaningless for 1D but added to permit shared code!)
text_style y_value_label_style_; // Not used for 1D but needed by axis_plot_frame.hpp.
};


```

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 construct/copy/destruct

1. `svg_1d_plot();`

Default constructor. Many (but not all - see below) default values here. See documentation for default settings rationale.

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. `bool autoscale();`

Returns: true if to use autoscale values autoscaling for X-axis.

Returns: true if to use autoscale values for X-axis. `autoscale()` is same as `x_autoscale`.

Returns: true if to use autoscale values for X-axis. `autoscale()` is same as `x_autoscale`.

4. `svg_1d_plot & autoscale(bool b);`

Set true if to use autoscale values for X-axis.

Set whether to use X autoscaled values. Same as `x_autoscale`, and used by boxplot too.

Set whether to use X autoscaled values. Same as `x_autoscale`, and used by boxplot too.

5. `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'.

6. `bool autoscale_check_limits();`

Returns: true if to check that values used for autoscaling are within limits.

7. `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'.

Set to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed.

Set to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed.

8. `bool autoscale_check_limits();`

Returns: true if to check that values used for autoscaling are within limits.

Returns: to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed.

Returns: to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed.

9. `svg_1d_plot & autoscale_plusminus(double);`

Set how many std_dev or standard deviations to allow for ellipses when autoscaling.

10. `double autoscale_plusminus();`

Returns: How many std_dev or standard deviations allowed for ellipses when autoscaling.

11. `svg_1d_plot & autoscale_plusminus(double);`

Set how many std_dev or standard deviations to allow for ellipses when autoscaling.

Set how many std_dev or standard deviation to allow for ellipse when autoscaling. Default is 3 for 99% confidence.

Set how many std_dev or standard deviation to allow for ellipse when autoscaling. Default is 3 for 99% confidence.

12. `double autoscale_plusminus();`

Returns: How many std_dev or standard deviations allowed for ellipses when autoscaling.

Returns: how many std_dev or standard deviation to allow for ellipse when autoscaling. Default is 3 for 99% confidence.

Returns: how many std_dev or standard deviation to allow for ellipse when autoscaling. Default is 3 for 99% confidence.

13. `svg_1d_plot & axes_on(bool is);`

Set true if to draw **both** x and y axes (note plural axes).

14. `bool axes_on();`

Returns: true if to draw **both** x and y axis on.

15. `svg_1d_plot & axes_on(bool is);`

Set true if to draw **both** x and y axes (note plural axes).

If set true, draw **both** x and y axes (note plural axes).

If set true, draw **both** x and y axes (note plural axes).

16. `bool axes_on();`

Returns: true if to draw **both** x and y axis on.

Returns: true if **both** x and y axis on.

Returns: true if **both** x and y axis on.

17. `svg_1d_plot & background_border_color(const svg_color & col);`

Set plot background border color.

18. `svg_color background_border_color();`

Returns: plot background border color.

19. `svg_1d_plot & background_border_color(const svg_color & col);`

Set plot background border color.

Set plot background border color.

`background_border_color`, for example: `svg_2d_plot my_plot(my_data, "My Data").background_border_color(red).background_color(azure);`

Set plot background border color.

`background_border_color`, for example: `svg_2d_plot my_plot(my_data, "My Data").background_border_color(red).background_color(azure);`

20. `svg_color background_border_color();`

Returns: plot background border color.

Returns: plot background border color.

Returns: plot background border color.

21. `svg_1d_plot & background_border_width(double w);`

Set plot background border width.

22. `double background_border_width();`

Returns: Plot background border width.

23. `svg_1d_plot & background_border_width(double w);`

Set plot background border width.

Set plot background border width.

Set plot background border width.

24. `double background_border_width();`

Returns: Plot background border width.

Returns: plot background border width.

Returns: plot background border width.

25. `svg_color background_color();`

Returns: Plot background color.

26. `svg_1d_plot & background_color(const svg_color & col);`

Set plot background color.

27. `svg_color background_color();`

Returns: Plot background color.

Returns: plot background color.

Returns: plot background color.

28. `svg_1d_plot & background_color(const svg_color & col);`

Set plot background color.

Set plot background color.

Set plot background color.

29. `svg_1d_plot & boost_license_on(bool l);`

Set true if the Boost license conditions should be included in the SVG document.

30. `bool boost_license_on();`

Returns: true if the Boost license conditions should be included in the SVG document.

31. `svg_1d_plot & boost_license_on(bool l);`

Set true if the Boost license conditions should be included in the SVG document.

Set if the Boost license conditions should be included in the SVG document.

Set if the Boost license conditions should be included in the SVG document.

32. `bool boost_license_on();`

Returns: true if the Boost license conditions should be included in the SVG document.

Returns: if the Boost license conditions should be included in the SVG document. To set see `axis_plot_frame::boost_license_on(bool)`.

Returns: if the Boost license conditions should be included in the SVG document. To set see `axis_plot_frame::boost_license_on(bool)`.

33. `void calculate_plot_window();`

Calculate the size and position of the plot window, taking account of the length and font size of axes labels, axis ticks, title and legend box. This version is only for 1-D. All calculations use SVG units, pixels by default.

34. `void calculate_transform();`

Calculate scale and shift factors for transforming from Cartesian to SVG plot. SVG image is (0, 0) at top left, Cartesian (0, 0) at bottom left.

35. `svg_1d_plot & confidence(double);`

Set confidence alpha for display of confidence intervals (default 0.05 for 95%).

36. `double confidence();`

Returns: Confidence alpha for display of confidence intervals (default 0.05 for 95%).

37. `svg_1d_plot & confidence(double);`

Set confidence alpha for display of confidence intervals (default 0.05 for 95%).

Set alpha for displaying confidence intervals. Default is 0.05 for 95% confidence.

Set alpha for displaying confidence intervals. Default is 0.05 for 95% confidence.

38. `double confidence();`

Returns: Confidence alpha for display of confidence intervals (default 0.05 for 95%).

Returns: alpha for displaying confidence intervals. Default is 0.05.

Returns: alpha for displaying confidence intervals. Default is 0.05.

39. `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.

40. `int coord_precision();`

Returns: precision of SVG coordinates in decimal digits.

41. `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.

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.

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.

42. `int coord_precision();`

Returns: precision of SVG coordinates in decimal digits.
 Returns: precision of SVG coordinates in decimal digits.
 Returns: precision of SVG coordinates in decimal digits.

43. `svg_1d_plot & copyright_date(const std::string d);`

Writes copyright date to the SVG document. and as metadata:<meta name="date" content="2007" />

44. `const std::string copyright_date();`

Returns: SVG document copyright_date.

45. `svg_1d_plot & copyright_date(const std::string d);`

Writes copyright date to the SVG document. and as metadata:<meta name="date" content="2007" />

Writes copyright date to the document. and asmetadata <meta name="date" content="2007" />

Writes copyright date to the document. and asmetadata <meta name="date" content="2007" />

46. `const std::string copyright_date();`

Returns: SVG document copyright_date.
 Returns: copyright_date.
 Returns: copyright_date.

47. `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" />

48. `const std::string copyright_holder();`

Returns: SVG document copyright holder.

49. `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" />

Writes copyright_holder to the document (as<!-- SVG Plot Copyright Paul A. Bristow 2007 -->) and asmetadata: <meta name="copyright" content="Paul A. Bristow" /meta>

Writes copyright_holder to the document (for header as<!-- SVG Plot Copyright Paul A. Bristow 2007 -->) and asmetadata: <meta name="copyright" content="Paul A. Bristow" /meta>

50. `const std::string copyright_holder();`

Returns: SVG document copyright holder.
 Returns: copyright holder.
 Returns: copyright holder.

51. `svg_1d_plot & data_lines_width(double width);`

Set the width of lines joining data points.

52. `double data_lines_width();`

Returns: the width of lines joining data points.

53. `svg_1d_plot & data_lines_width(double width);`

Set the width of lines joining data points.

Set the width of lines joining data points.

Set the width of lines joining data points.

54. `double data_lines_width();`

Returns: the width of lines joining data points.
 Returns: the width of lines joining data points.
 Returns: the width of lines joining data points.

55. `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. Sadly this has the most unwelcome effect of terminally confusing the Visual Studio Intellisense, and sometimes the debugger as well :-(

56. `const svg_1d_plot & derived() const;`

const version of derived()

57. `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. Sadly this has the most unwelcome effect of terminally confusing the Visual Studio Intellisense, and sometimes the debugger as well :-(

58. `const svg_1d_plot & derived() const;`

const version of derived()

59. `svg_1d_plot & description(const std::string d);`

Writes description to the document for header as.

<desc> My Description </desc>.

60. `const std::string & description();`

Returns: Description of the document for header as<desc> My description </desc>.

61. `svg_1d_plot & description(const std::string d);`

Writes description to the document for header as.

<desc> My Description </desc>.

Writes description to the SVG document for header, for example: <desc> My Data </desc>

Writes description to the document for header, for example: <desc> My Data </desc>

62. `const std::string & description();`

Returns: Description of the document for header as<desc> My description </desc>.

Returns: Description of the document for title as<desc> ... </desc>

Returns: description of the document for header as<desc> ... </desc>

63. `svg_1d_plot & document_title(const std::string d);`

Set document title to the document for header as.

<title> My Title </title>.

64. `std::string document_title();`

Returns: Document title to the document for header as<title> My Title </title>.

65. `svg_1d_plot & document_title(const std::string d);`

Set document title to the document for header as.

<title> My Title </title>.

Write document title to the SVG document for title as<title> My Title </title>

Write document title to the SVG document for header as<title> My Title </title>

66. `std::string document_title();`

Returns: Document title to the document for header as<title> My Title </title>.

Returns: Get document title to the document for title as<title> My Title </title>

Returns: Get document title to the document for header as<title> My Title </title>

67. `void draw_axes();`

Add information to the plot image for X-axis lines. (For 1-D, there is, of course, only the horizontal X-axis, but there can be a vertical Y-axis line at x = 0).

68.

```
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.

69.

```
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.

Annotate plot with a line from SVG Coordinates X1, Y1 to X2, Y2.

Default color black.

Annotate plot with a line from SVG Coordinates X1, Y1 to X2, Y2.

Default color black.

70.

```
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.

71.

```
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.

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.

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.

72.

```
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.

73.

```
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.

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.

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.

74.

```
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.

75.

```
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.

Annotate plot with a line from user's Cartesian Coordinates X1, Y1 to X2, Y2.

For example, -10, -10, +10, +10, Default color 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.

76.

```
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.

77.

```
double image_border_margin();
```

Returns: the margin around the plot window border (svg units, default pixels).

78. `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.

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.

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.

79. `double image_border_margin();`

Returns: the margin around the plot window border (svg units, default pixels).

Returns: the margin around the plot window border (svg units, default pixels).

Returns: the margin around the plot window border (svg units, default pixels).

80. `svg_1d_plot & image_border_width(double w);`

Set the svg image border width (svg units, default pixels).

81. `double image_border_width();`

Returns: the svg image border width (svg units, default pixels).

82. `svg_1d_plot & image_border_width(double w);`

Set the svg image border width (svg units, default pixels).

Set the svg image border width (svg units, default pixels).

Set the svg image border width (svg units, default pixels).

83. `double image_border_width();`

Returns: the svg image border width (svg units, default pixels).

Returns: the svg image border width (svg units, default pixels).

Returns: the svg image border width (svg units, default pixels).

84. `unsigned int image_x_size();`

Obselete - deprecated use x_size()

85. `svg_1d_plot & image_x_size(unsigned int i);`

Obselete - deprecated - use x_size().

Set SVG image X-axis size (SVG units, default pixels).

86. `int image_x_size();`

Obselete - deprecated use x_size()

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).
Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

87. `svg_1d_plot & image_x_size(int i);`

Obselete - deprecated - use x_size().

Set SVG image X-axis size (SVG units, default pixels).

88. `unsigned int image_y_size();`

Obselete - deprecated - use y_size()

89. `svg_1d_plot & image_y_size(unsigned int i);`

Obselete - deprecated - use y_size()

Set SVG image Y-axis size (SVG units, default pixels).

90. `int image_y_size();`

Obselete - deprecated - use y_size()

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).
Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

91. `svg_1d_plot & image_y_size(int i);`

Obselete - deprecated - use y_size()

Set SVG image Y-axis size (SVG units, default pixels).

92. `svg_1d_plot & legend_background_color(const svg_color & col);`

Set the background fill color of the legend box.

93. `svg_color legend_background_color();`

Returns: the background fill color of the legend box.

94. `svg_1d_plot & legend_background_color(const svg_color & col);`

Set the background fill color of the legend box.

Set the background fill color of the legend box.//

Set the background fill color of the legend box.

95. `svg_color legend_background_color();`

Returns: the background fill color of the legend box.

Returns: the background fill color of the legend box.

Returns: the background fill color of the legend box.

96 `svg_1d_plot & legend_border_color(const svg_color & col);`

Set the border stroke color of the legend box.

97. `svg_color legend_border_color();`

Returns: the border stroke color of the legend box.

98 `svg_1d_plot & legend_border_color(const svg_color & col);`

Set the border stroke color of the legend box.

Set the border stroke color of the legend box.

Set the border stroke color of the legend box.

99. `svg_color legend_border_color();`

Returns: the border stroke color of the legend box.

Returns: the border stroke color of the legend box.

Returns: the border stroke color of the legend box.

100 `const std::pair< double, double > legend_bottom_right();`

Returns: SVG coordinate (default pixels) of bottom right of legend box.

101 `const std::pair< double, double > legend_bottom_right();`

Returns: SVG coordinate (default pixels) of bottom right of legend box.

Returns: svg coordinate (default pixels) of Bottom right of legend box.

Returns: svg coordinate (default pixels) of Bottom right of legend box.

102 `bool legend_box_fill_on();`

Returns: true if legend box has a background fill color.

103 `bool legend_box_fill_on();`

Returns: true if legend box has a background fill color.

Returns: true if legend box has a background fill color.

Returns: true if legend box has a background fill color.

104 `svg_1d_plot & legend_color(const svg_color & col);`

Set the color of the title of the legend.

105 `svg_color legend_color();`

Returns: the color of the title of the legend.

16 `svg_1d_plot & legend_color(const svg_color & col);`

Set the color of the title of the legend.

Set the color of the title of the legend.

Set the color of the title of the legend.

17 `svg_color legend_color();`

Returns: the color of the title of the legend.

Returns: the color of the title of the legend.

Returns: the color of the title of the legend.

18 `svg_1d_plot & legend_font_family(const std::string & family);`

Set the font family for the legend title.

19 `const std::string & legend_font_family();`

Returns: the font family for the legend title.

10 `svg_1d_plot & legend_font_family(const std::string & family);`

Set the font family for the legend text.

Set the font family for the legend title.

Set the font family for the legend title.

11 `const std::string & legend_font_family();`

Returns: the font family for the legend title.

Returns: the font family for the legend title.

Returns: the font family for the legend title.

12 `svg_1d_plot & legend_font_size(int size);`

Set Font size for **both** the legend title and legend text (svg units, default pixels).

Returns: Font size for the legend title and legend text.

13 `int legend_font_size();`



Note

If function .legend_font_size(size) has been used to set both title and text font sizes, then these will be the same (unless altered by a call of .legend_text_font_size(size)).

Returns: Font size for the legend title (svg units, default pixels).

Returns: Font size for the legend title (svg units, default pixels).

14 `svg_1d_plot & legend_font_weight(const std::string & weight);`

Set the font weight for the legend title.

15 `const std::string & legend_font_weight();`

Returns: Font weight for the legend title.

16 `svg_1d_plot & legend_font_weight(const std::string & weight);`

Set the font weight for the legend title.

Set the font weight for the legend title.

Set the font weight for the legend title.

17 `const std::string & legend_font_weight();`

Returns: Font weight for the legend text.

Returns: the font weight for the legend title.

Returns: the font weight for the legend title.

18 `svg_1d_plot & legend_header_font_size(int size);`

Set legend header font size (svg units, default pixels).

Set legend header font size (svg units, default pixels).

19 `int legend_header_font_size();`

Returns: legend header font size (svg units, default pixels).

Returns: legend header font size (svg units, default pixels).

20 `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.

21 `bool legend_lines();`

Returns: true if legend should include samples of the lines joining data points.

22 `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.

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.

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.

13 `bool legend_lines();`

Returns: true if legend should include samples of the lines joining data points.
 Returns: true if legend should include samples of the lines joining data points.
 Returns: true if legend should include samples of the lines joining data points.

14 `svg_1d_plot & legend_on(bool cmd);`

Set true if a legend is wanted.

15 `bool legend_on();`

Returns: true if a legend is wanted.

16 `svg_1d_plot & legend_on(bool cmd);`

Set true if a legend is wanted.

Set true if a legend is wanted.

Set true if a legend is wanted.

17 `bool legend_on();`

Returns: true if a legend is wanted.
 Returns: true if a legend is wanted.
 Returns: true if a legend is wanted.

18 `bool legend_outside();`

Returns: if the legend should be outside the plot area.

19 `bool legend_outside();`

Returns: if the legend should be outside the plot area.
 Returns: if the legend should be outside the plot area.
 Returns: if the legend should be outside the plot area.

20 `svg_1d_plot & legend_place(legend_places l);`

Set the position of the legend..

See Also:

`boost::svg::legend_places`

21 `legend_places legend_place();`

See Also:

`boost::svg::legend_places`

Returns: the position of the legend,

12 `svg_1d_plot & legend_place(legend_places l);`

Set the position of the legend.,

See Also:

`boost::svg::legend_places`

Set the position of the legend,

See Also:

`boost::svg::legend_places`

Set the position of the legend,

See Also:

`boost::svg::legend_places`

13 `legend_places legend_place();`

See Also:

`boost::svg::legend_places`

See Also:

`boost::svg::legend_places`

See Also:

`boost::svg::legend_places`

Returns: the position of the legend,

Returns: the position of the legend,

Returns: the position of the legend,

14 `svg_1d_plot & legend_text_font_size(int size);`

Set the font size for the legend title (svg units, default pixels).

Returns: Font size for the legend text.

15 `int legend_text_font_size();`

Returns: Font size for the legend text (svg units, default pixels).

Returns: the font size for the legend title (svg units, default pixels).

16 `svg_1d_plot & legend_text_font_weight(const std::string & weight);`

Set the font weight for the legend text. Example: `my_plot.legend_text_font_size(20);`

17 `const std::string & legend_text_font_weight();`

Returns: the font weight for the legend text.

13 `svg_1d_plot & legend_title(const std::string title);`

Set the title for the legend.

14 `const std::string legend_title();`

Returns: Title for the legend.

15 `svg_1d_plot & legend_title(const std::string title);`

Set the title for the legend.

Set the title for the legend.

Set the title for the legend.

16 `const std::string legend_title();`

Returns: Title for the legend.

Returns: the title for the legend.

Returns: the title for the legend.

17 `svg_1d_plot & legend_title_font_size(unsigned int size);`

Set the font size for the legend title (svg units, default pixels).

Returns: Font family for the legend title.

18 `unsigned int legend_title_font_size();`

Returns: Font size for the legend title (svg units, default pixels).

19 `svg_1d_plot & legend_title_font_size(int size);`

Set the font size for the legend title (svg units, default pixels).

Returns: Font size for the legend title.

20 `int legend_title_font_size();`

Returns: Font size for the legend title (svg units, default pixels).

Returns: the font size for the legend title (svg units, default pixels).

Returns: the font size for the legend title (svg units, default pixels).

21 `svg_1d_plot & legend_title_font_weight(const std::string & weight);`

Set the font weight for the legend title.

Set the font weight for the legend title. Example: `my_plot.legend_title_font_size(10);`

22 `const std::string & legend_title_font_weight();`

Returns: Font weight for the legend title.

Returns: the font weight for the legend title.

18 `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).

19 `const std::pair< double, double > legend_top_left();`

Returns: SVG coordinate (default pixels) of top left of legend box.

20 `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).

Set position of top left of legend box (svg coordinates, default pixels). Bottom right is controlled by contents, so the user cannot set it.

Set position of top left of legend box (svg coordinates, default pixels). Bottom right is controlled by contents, so the user cannot set it.

21 `const std::pair< double, double > legend_top_left();`

Returns: SVG coordinate (default pixels) of top left of legend box.

Returns: svg coordinate (default pixels) of top left of legend box.

Returns: svg coordinate (default pixels) of top left of legend box.

22 `svg_1d_plot & legend_width(double width);`

Set the width for the legend box.

23 `double legend_width();`

Returns: Width for the legend box.

24 `svg_1d_plot & legend_width(double width);`

Set the width for the legend box.

Set the width for the legend.

Set the width for the legend.

25 `double legend_width();`

Returns: Width for the legend box.

Returns: the width for the legend.

Returns: the width for the legend.

26 `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.

17 `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.

Set license conditions for reproduction, attribution, commercial use, and derivative works, usually "permits", "requires", or "prohibits", and set license_on == true.

Set license conditions for reproduction, attribution, commercial use, and derivative works, usually "permits", "requires", or "prohibits", and set license_on == true.

18 `const std::string license_attribution();`

Returns: SVG document attribution license conditions, usually "permits", "requires", or "prohibits".

19 `const std::string license_attribution();`

Returns: SVG document attribution license conditions, usually "permits", "requires", or "prohibits".

Returns: attribution license conditions, usually "permits", "requires", or "prohibits".

Returns: attribution license conditions, usually "permits", "requires", or "prohibits".

20 `const std::string license_commercialuse();`

Returns: SVG document commercial use license conditions, usually "permits", "requires", or "prohibits".

21 `const std::string license_commercialuse();`

Returns: SVG document commercial use license conditions, usually "permits", "requires", or "prohibits".

Returns: commercial use license conditions, usually "permits", "requires", or "prohibits".

Returns: commercial use license conditions, usually "permits", "requires", or "prohibits".

22 `const std::string license_distribution();`

Returns: SVG document distribution license conditions, usually "permits", "requires", or "prohibits".

23 `const std::string license_distribution();`

Returns: SVG document distribution license conditions, usually "permits", "requires", or "prohibits".

Returns: distribution license conditions, usually "permits", "requires", or "prohibits".

Returns: distribution license conditions, usually "permits", "requires", or "prohibits".

24 `svg_1d_plot & license_on(bool l);`

Set if license conditions should be included in the SVG document.

25 `bool license_on();`

Returns: true if license conditions should be included in the SVG document.

16 `svg_1d_plot & license_on(bool l);`

Set if license conditions should be included in the SVG document.

Set if license conditions should be included in the SVG document.

See Also:

`axis_plot_frame::license`

Set if license conditions should be included in the SVG document.

See Also:

`axis_plot_frame::license`

17 `bool license_on();`

See Also:

`axis_plot_frame::license`

See Also:

`axis_plot_frame::license`

Returns: true if license conditions should be included in the SVG document.

Returns: true if license conditions should be included in the SVG document.

Returns: true if license conditions should be included in the SVG document.

18 `const std::string license_reproduction();`

Returns: SVG document reproduction license conditions, usually "permits", "requires", or "prohibits".

19 `const std::string license_reproduction();`

Returns: SVG document reproduction license conditions, usually "permits", "requires", or "prohibits".

Returns: reproduction license conditions, usually "permits", "requires", or "prohibits".

Returns: reproduction license conditions, usually "permits", "requires", or "prohibits".

20 `svg_1d_plot & limit_color(const svg_color &);`

Set the color for 'at limit' point stroke color.

21 `svg_color limit_color();`

Returns: the color for the 'at limit' point stroke color.

22 `svg_1d_plot & limit_color(const svg_color &);`

Set the color for 'at limit' point stroke color.

Set the color for 'at limit' point stroke color.

Set the color for 'at limit' point stroke color.

13 `svg_color limit_color();`

Returns: the color for the 'at limit' point stroke color.
 Returns: the color for the 'at limit' point stroke color.
 Returns: the color for the 'at limit' point stroke color.

14 `svg_1d_plot & limit_fill_color(const svg_color &);`

Set the color for 'at limit' point fill color.

15 `svg_color limit_fill_color();`

Returns: the color for the 'at limit' point fill color.

16 `svg_1d_plot & limit_fill_color(const svg_color &);`

Set the color for 'at limit' point fill color.

Set the color for 'at limit' point fill color.

Set the color for 'at limit' point fill color.

17 `svg_color limit_fill_color();`

Returns: the color for the 'at limit' point fill color.
 Returns: the color for the 'at limit' point fill color.
 Returns: the color for the 'at limit' point fill color.

18 `svg_1d_plot & one_sd_color(const svg_color &);`

Set the color for the one standard deviation (~67% confidence) ellipse fill.

19 `svg_color one_sd_color();`

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

20 `svg_1d_plot & one_sd_color(const svg_color &);`

Set the color for the one standard deviation (~67% confidence) ellipse fill.

Set the color for the one standard deviation (~67% confidence) ellipse fill.

Set the color for the one standard deviation (~67% confidence) ellipse fill.

21 `svg_color one_sd_color();`

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.
 Returns: Color for the one standard deviation (~67% confidence) ellipse fill.
 Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

```
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).

```
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"); // aspect_ratioole 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` aspect_ratioioch 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.

```
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 <code>std::vector</code>) 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 (aspect_ratioich must be convertible to double).
	U	functor floating-point type (default is <code>double_1d_convert</code>).
Returns:		a reference to data series just added (to make chainable).

18 `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 (aspect_ratioich must be convertible to double).
	U	Functor floating-point type (default is <code>double_1d_convert</code>).
Returns:		a reference to data series just added (to make chainable).

18 `svg_1d_plot & plot_background_color(const svg_color & col);`

Set the fill color of the plot window background.

18 `svg_color plot_background_color();`

Returns: the fill color of the plot window background.

18 `svg_1d_plot & plot_background_color(const svg_color & col);`

Set the fill color of the plot window background.

Set the fill color of the plot window background.

Set the fill color of the plot window background.

18 `svg_color plot_background_color();`

Returns: the fill color of the plot window background.

Returns: the fill color of the plot window background.

Returns: the fill color of the plot window background.

19 `svg_1d_plot & plot_border_color(const svg_color & col);`

Set the color for the plot window background.

20 `svg_color plot_border_color();`

Returns: the color for the plot window background.

21 `svg_1d_plot & plot_border_color(const svg_color & col);`

Set the color for the plot window background.

Set the color for the plot window background. Example: `.plot_border_color(lightgoldenrodyellow)`

Set the color for the plot window background.

22 `svg_color plot_border_color();`

Returns: the color for the plot window background.

Returns: the color for the plot window background.

Returns: the color for the plot window background.

23 `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.

24 `double plot_border_width();`

Returns: the width for the plot window border (svg units, default pixels).

25 `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.

Set the width for the plot window border (svg units, default pixels).

Set the width for the plot window border (svg units, default pixels).

26 `double plot_border_width();`

Returns: the width for the plot window border (svg units, default pixels).

Returns: the width for the plot window border (svg units, default pixels).

Returns: the width for the plot window border (svg units, default pixels).

27 `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).

19 `bool plot_window_on();`

Returns: true if a plot window is wanted (or false if the whole image is to be used).

20 `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).

Set true if a plot window is wanted (or false if the whole image is to be used).

Set true if a plot window is wanted (or false if the whole image is to be used).

21 `bool plot_window_on();`

Returns: true if a plot window is wanted (or false if the whole image is to be used).

Returns: true if a plot window is wanted (or false if the whole image is to be used).

Returns: true if a plot window is wanted (or false if the whole image is to be used).

22 `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.

23 `std::pair< double, double > plot_window_x();`

Returns: both the left and right (X axis) of the plot window.

24 `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.

Set the minimum and maximum (cartesian data units) for the plot window X axis. This is normally calculated from other plot values.

Set the minimum and maximum (cartesian data units) for the plot window X axis. This is normally calculated from other plot values.

25 `std::pair< double, double > plot_window_x();`

Returns: both the left and right (X axis) of the plot window.

Returns: both the left and right (X axis) of the plot window.

Returns: both the left and right (X axis) of the plot window.

26 `double plot_window_x_left();`

Returns: left of the plot window.

27 `double plot_window_x_left();`

Returns: left of the plot window.

Returns: left of the plot window.

Returns: left of the plot window.

28 `double plot_window_x_right();`

Returns: right of the plot window.

29 `double plot_window_x_right();`

Returns: right of the plot window.

Returns: right of the plot window.

Returns: right of the plot window.

20 `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.

21 `std::pair< double, double > plot_window_y();`

Returns: both the top and bottom (Y axis) of the plot window.

22 `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.

Set the minimum and maximum (cartesian data units) for the plot window Y axis. This is normally calculated from other plot values.

Set the minimum and maximum (cartesian data units) for the plot window Y axis. This is normally calculated from other plot values.

23 `std::pair< double, double > plot_window_y();`

Returns: both the top and bottom (Y axis) of the plot window.

Returns: both the top and bottom (Y axis) of the plot window.

Returns: both the top and bottom (Y axis) of the plot window.

24 `double plot_window_y_bottom();`

Returns: top of the plot window.

25 `double plot_window_y_bottom();`

Returns: top of the plot window.

Returns: Top of the plot window.

Returns: Top of the plot window.

26 `double plot_window_y_top();`

Returns: top of the plot window.

27 `double plot_window_y_top();`

Returns: top of the plot window.
 Returns: top of the plot window.
 Returns: top of the plot window.

28 `void set_ids();`

Document ids for use in identifying group elements, for example: <g id = "PLOT_TITLE".../g>

29 `svg_1d_plot & size(unsigned int x, unsigned int y);`

Set SVG image size (SVG units, default pixels).

Set SVG image size (SVG units, default pixels).

20 `std::pair< double, double > size();`

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

21 `svg_1d_plot & size(int x, int y);`

Set SVG image size (SVG units, default pixels).

Set SVG image size (SVG units, default pixels).

22 `std::pair< double, double > size();`

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

23 `svg_1d_plot & three_sd_color(const svg_color &);`

Set the color for three standard deviation (~99% confidence) ellipse fill.

24 `svg_color three_sd_color();`

Returns: Color for three standard deviation (~99% confidence) ellipse fill.

25 `svg_1d_plot & three_sd_color(const svg_color &);`

Set the color for three standard deviation (~99% confidence) ellipse fill.

Set the color for three standard deviation (~99% confidence) ellipse fill.

Set the color for three standard deviation (~99% confidence) ellipse fill.

26 `svg_color three_sd_color();`

Returns: Color for three standard deviation (~99% confidence) ellipse fill.

Returns: Color for three standard deviation (~99% confidence) ellipse fill.

Returns: Color for three standard deviation (~99% confidence) ellipse fill.

22 `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, for example:

```
my_plot.title("Plot of &#xA9; function (&#x00B0;C)");
```

Unicode symbols are at <http://unicode.org/charts/symbols.html>.

23 `const std::string title();`

Returns: Title for plot (whose string may include Unicode for greek letter and symbols).

24 `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, for example:

```
my_plot.title("Plot of &#xA9; function (&#x00B0;C)");
```

Unicode symbols are at <http://unicode.org/charts/symbols.html>.

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>.



Note

Only set plot title once.

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>.



Note

Only set plot title once.

25 `const std::string title();`

Example: `std::cout << "title \" " << my_plot.title() << " \"<< std::endl;` outputs: title "Plot showing several data point marker shapes & sizes."

Example: `std::cout << "title \" " << my_plot.title() << "" " << std::endl;` outputs: title "Plot showing several data point marker shapes & sizes."

Returns: `std::string` text for the title, (whose string may include Unicode for greek letter and symbols).

Returns: Title for plot (whose string may include Unicode for greek letter and symbols).

Returns: Title for plot,

Returns: Title for plot,

21 `svg_1d_plot & title_color(const svg_color & col);`

Set the color of any title of the plot.

22 `svg_color title_color();`

Returns: the color of any title of the plot.

23 `svg_1d_plot & title_color(const svg_color & col);`

Set the color of any title of the plot.

Set the color of any title of the plot.

Set the color of any title of the plot.

24 `svg_color title_color();`

Returns: the color of any title of the plot.

Returns: the color of any title of the plot.

Returns: the color of any title of the plot.

25 `svg_1d_plot & title_font_alignment(align_style alignment);`

Set the alignment for the title.

26 `align_style title_font_alignment();`

Returns: the alignment for the title.

27 `svg_1d_plot & title_font_alignment(align_style alignment);`

Set the alignment for the title.

Set the alignment for the title.

Set the alignment for the title.

28 `align_style title_font_alignment();`

Returns: the alignment for the title.

Returns: the alignment for the title.

Returns: the alignment for the title.

29 `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).

20 `const std::string & title_font_decoration();`

Returns: Font decoration for the title (default normal, or underline, overline or strike-thru).

21 `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).

Set the font decoration for the title (default normal, or underline, overline or strike-thru).

Set the font decoration for the title (default normal, or underline, overline or strike-thru).

22 `const std::string & title_font_decoration();`

Returns: Font decoration for the title (default normal, or underline, overline or strike-thru).

Returns: the font decoration for the title (default normal, or underline, overline or strike-thru).

Returns: the font decoration for the title (default normal, or underline, overline or strike-thru).

23 `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");`)

24 `const std::string & title_font_family();`

Returns: Font family for the title.

25 `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");`)

Set the font family for the title (for example: `.title_font_family("Lucida Sans Unicode");`)

Set the font family for the title (for example: `.title_font_family("Lucida Sans Unicode");`)

26 `const std::string & title_font_family();`

Returns: Font family for the title.

Returns: the font family for the title

Returns: the font family for the title

27 `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...)

28 `int title_font_rotation();`

Returns: the rotation for the title font (degrees).

29 `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...)

Set the rotation for the title font (degrees, 0 to 360).

Set the rotation for the title font (degrees, 0 to 360).

20 `int title_font_rotation();`

Returns: the rotation for the title font (degrees).

Returns: the rotation for the title font (degrees).

Returns: the rotation for the title font (degrees).

21 `svg_1d_plot & title_font_size(unsigned int i);`

Sets the font size for the title (SVG units, default pixels).

Sets the font size for the title (svg units, default pixels).

22 `unsigned int title_font_size();`

Returns: Font size for the title (SVG units, default pixels).

23 `svg_1d_plot & title_font_size(int i);`

Sets the font size for the title (SVG units, default pixels).

Sets the font size for the title (svg units, default pixels).

24 `int title_font_size();`

return Font size for the title (SVG units, default pixels). Example: /code std::cout << my_plot.title_font_size() ...

Returns: the font size for the title (svg units, default pixels).

Returns: the 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_stretch(const std::string & stretch);`

Set the font stretch for the title (default normal), wider or narrow.

Set the font stretch for the title (default normal), wider or narrow.

Set the font stretch for the title (default normal), wider or narrow.

28 `const std::string & title_font_stretch();`

Returns: Font stretch for the title.

Returns: the font stretch for the title.

Returns: the font stretch for the title.

29 `svg_1d_plot & title_font_style(const std::string & style);`

Set the font style for the title (default normal).

30 `const std::string & title_font_style();`

Returns: Font style for the title (default normal).

31 `svg_1d_plot & title_font_style(const std::string & style);`

Set the font style for the title (default normal).

Set the font style for the title (default normal).

Set the font style for the title (default normal).

32 `const std::string & title_font_style();`

Returns: Font style for the title (default normal).

Returns: the font style for the title (default normal).

Returns: the font style for the title (default normal).

33 `svg_1d_plot & title_font_weight(const std::string & weight);`

Set the font weight for the title (default normal).

34 `const std::string & title_font_weight();`

Returns: Font weight for the title.

35 `svg_1d_plot & title_font_weight(const std::string & weight);`

Set the font weight for the title (default normal).

Set the font weight for the title (default normal).

Set the font weight for the title (default normal).

36 `const std::string & title_font_weight();`

Returns: Font weight for the title.

Returns: the font weight for the title.

Returns: the font weight for the title.

37 `svg_1d_plot & title_on(bool cmd);`

If set true, show a title for the plot. Note: is set true by setting a title.

38 `bool title_on();`

Returns: true if will show a title for the plot.

29 `svg_1d_plot & title_on(bool cmd);`

If set true, show a title for the plot. Note: is set true by setting a title.

If set true, show a title for the plot.

If set true, show a title for the plot.

20 `bool title_on();`

If true, will show a title for the plot.

If true, will show a title for the plot.

Returns: true if will show a title for the plot.

21 `text_style & title_style();`

Returns: All style info for the title, font, famil, size ... (SVG units, default pixels).

Returns: Font size for the title (svg units, default pixels). Example: std::cout << "title style " << my_2d_plot.title_style() << std::endl; Outputs: title style text_style(10, "Lucida Sans Unicode", "", "normal", "", "", 900)

22 `svg_1d_plot & title_text_length(double);`

Sets the text_length for the title (SVG units, default pixels).

Sets the estimated text length for the title (svg units, default pixels).

23 `double title_text_length();`

return text_length for the title (SVG units, default pixels).

Returns: the estimated text length for the title (svg units, default pixels).

24 `svg_1d_plot & two_sd_color(const svg_color &);`

Set the color for two standard deviation (~95% confidence) ellipse fill.

25 `svg_color two_sd_color();`

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

26 `svg_1d_plot & two_sd_color(const svg_color &);`

Set the color for two standard deviation (~95% confidence) ellipse fill.

Set the color for two standard deviation (~95% confidence) ellipse fill.

Set the color for two standard deviation (~95% confidence) ellipse fill.

27 `svg_color two_sd_color();`

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

28 `void update_image();`

Calls functions to add all plot information to the image, including plot window, axes, ticks, labels, grids, legend, and finally all the data series.

29 `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, `aspect_ratioich` 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.

30 `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.

31 `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>".

32 `svg_color x_addlimits_color();`

Returns: the color of X confidence limits of value, for example, the color of "<1.23, 1.45>".

33 `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>".

Set the color of X confidence limits value, for example, the color of "<1.23 , 1.34>".

Set the color of X confidence limits value, for example, the color of "<1.23 , 1.34>".

24 `svg_color x_addlimits_color();`

Get the color of X confidence limits of value, for example, the color of "<1.23 , 1.34>"".

Get the color of X confidence limits of value, for example, the color of "<1.23 , 1.34>"".

Returns: the color of X confidence limits of value, for example, the color of "<1.23, 1.45>)".

25 `svg_1d_plot & x_addlimits_on(bool b);`

Set if to append confidence limits to data point X values near data points markers.

26 `bool x_addlimits_on();`

Returns: true if to append confidence limits estimate to data point X values near data points markers.

27 `svg_1d_plot & x_addlimits_on(bool b);`

Set if to append confidence limits to data point X values near data points markers.

Set if to append confidence limits to data point X values near data points markers. (May not be implemented yet).

Set if to append confidence limits to data point X values near data points markers. (May not be implemented yet).

28 `bool x_addlimits_on();`

Returns: true if to append confidence limits estimate to data point X values near data points markers.

Returns: if to append confidence limits to data point X values near data points markers. (May not be implemented yet).

Returns: if to append confidence limits to data point X values near data points markers. (May not be implemented yet).

29 `double x_auto_max_value();`

Returns: X-axis maximum value computed by autoscale.

30 `double x_auto_max_value();`

Returns: X-axis maximum value computed by autoscale.

Returns: the X-axis maximum value computed by autoscale.

Returns: the X-axis maximum value computed by autoscale.

31 `double x_auto_min_value();`

Returns: X-axis minimum value computed by autoscale.

32 `double x_auto_min_value();`

Returns: X-axis minimum value computed by autoscale.

Returns: the X-axis minimum value computed by autoscale.

Returns: the X-axis minimum value computed by autoscale.

33 `double x_auto_tick_interval();`

Returns: the X-axis major tick interval computed by autoscale.

24 `double x_auto_tick_interval();`

Returns: the X-axis major tick interval computed by autoscale.
 Returns: the X-axis major tick interval computed by autoscale.
 Returns: the X-axis major tick interval computed by autoscale.

25 `int x_auto_ticks();`

Returns: the X-axis number of major ticks computed by autoscale.

26 `int x_auto_ticks();`

Returns: the X-axis number of major ticks computed by autoscale.
 Returns: the X-axis number of major ticks computed by autoscale.
 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.

28 `svg_1d_plot & x_autoscale(bool b);`

Set true if to use autoscale values for X-axis.

29 `svg_1d_plot & x_autoscale(std::pair< double, double > p);`

autoscale X axis using a pair of doubles.

30 `svg_1d_plot & x_autoscale(const T & container);`

<

31 `svg_1d_plot & x_autoscale(const T & begin, const T & end);`

<

32 `bool x_autoscale();`

Returns: true if to use autoscale value for X-axis.
 Returns: true if to use autoscale value for X-axis.
 Returns: true if to use autoscale value for X-axis.

33 `svg_1d_plot & x_autoscale(bool b);`

Set true if to use autoscale values for X-axis.

Set true if to use autoscaled values for X-axis.

Set true if to use autoscaled values for X-axis.

Returns: Reference to caller to make chainable.

Returns: Reference to caller to make chainable.

34 `svg_1d_plot & x_autoscale(std::pair< double, double > p);`

autoscale X axis using a pair of doubles.

Set to use X min & max pair of double values to autoscale X-axis.

Set to use X min & max pair of double values to autoscale X-axis.

35 `svg_1d_plot & x_autoscale(const T & container);`

<

Data series (all values) to use to calculate autoscaled X-axis values.

By default, use these calculated values.

Data series (all values) to use to calculate autoscaled X-axis values.

By default, use these calculated values.

36 `svg_1d_plot & x_autoscale(const T & begin, const T & end);`

<

Data series (range accessed using iterators) to use to calculate autoscaled X-axis values.

By default, use these calculated values.

Data series (range accessed using iterators) to use to calculate autoscaled X-axis values.

By default, use these calculated values.

37 `svg_1d_plot & x_axis_color(const svg_color & col);`

Set the color of the X-axis line.

38 `svg_color x_axis_color();`

Returns: the color of the X-axis line.

39 `svg_1d_plot & x_axis_color(const svg_color & col);`

Set the color of the X-axis line.

Set the color of the X-axis line.

Set the color of the X-axis line.

40 `svg_color x_axis_color();`

Returns: the color of the X-axis line.

Returns: the color of the X-axis line.

Returns: the color of the X-axis line.

31 `svg_1d_plot & x_axis_label_color(const svg_color & col);`

Set X axis label color, for example, red.

32 `svg_color x_axis_label_color();`

Returns: X axis label color. X-axis ticks values label style.

33 `svg_1d_plot & x_axis_label_color(const svg_color & col);`

Set X axis label color, for example, red.

Set X axis label color.

Set X axis label color.

34 `svg_color x_axis_label_color();`

Returns: X axis label color. X-axis ticks values label style.

Returns: X axis label color.

Returns: X axis label color.

35 `svg_1d_plot & x_axis_on(bool is);`

If set true, draw a horizontal X-axis line.

36 `bool x_axis_on();`

Returns: true if will draw a horizontal X-axis line.

37 `svg_1d_plot & x_axis_on(bool is);`

If set true, draw a horizontal X-axis line.

If set true, draw a horizontal X-axis line.

If set true, draw a horizontal X-axis line.

38 `bool x_axis_on();`

If true, draw a horizontal X-axis line.

If true, draw a horizontal X-axis line.

Returns: true if will draw a horizontal X-axis line.

39 `const std::string x_axis_position();`

Returns: the position (or intersection with Y-axis) of the X-axis.

40 `const std::string x_axis_position();`

Returns: the position (or intersection with Y-axis) of the X-axis.

Returns: the position (or intersection with Y-axis) of the X-axis.

Returns: the position (or intersection with Y-axis) of the X-axis.

31 `svg_1d_plot & x_axis_vertical(double fraction);`

Set vertical position of X-axis for 1D as fraction of plot window.

32 `bool x_axis_vertical();`

Returns: vertical position of X-axis for 1D as fraction of plot window.

33 `svg_1d_plot & x_axis_vertical(double fraction);`

Set vertical position of X-axis for 1D as fraction of plot window.

Set vertical position of X-axis for 1D as fraction of plot window.

Set vertical position of X-axis for 1D as fraction of plot window.

34 `bool x_axis_vertical();`

Returns: vertical position of X-axis for 1D as fraction of plot window.

Returns: vertical position of X-axis for 1D as fraction of plot window.

Returns: vertical position of X-axis for 1D as fraction of plot window.

35 `svg_1d_plot & x_axis_width(double width);`

Set the width of X-axis lines.

36 `double x_axis_width();`

Returns: the width of X-axis lines.

37 `svg_1d_plot & x_axis_width(double width);`

Set the width of X-axis lines.

Set the width of X-axis lines.

Set the width of X-axis lines.

38 `double x_axis_width();`

Returns: the width of X-axis lines.

Returns: the width of X-axis lines.

Returns: the width of X-axis lines.

39 `svg_1d_plot & x_datetime_color(const svg_color & col);`

Set the color of X date time , for example, the color of text in "".

40 `svg_color x_datetime_color();`

Returns: the color of X date time, for example, the color of text in "".

31 `svg_1d_plot & x_datetime_color(const svg_color & col);`

Set the color of X date time , for example, the color of text in "".

Set the color of X point datetime, for example, the color of text in "2004-Jan-1 05:21:33.20".

Set the color of X point datetime, for example, the color of text in "2004-Jan-1 05:21:33.20".

32 `svg_color x_datetime_color();`

Get the color of X point date time, for example, the color of text in "2004-Jan-1 05:21:33.20".

Get the color of X point date time, for example, the color of text in "2004-Jan-1 05:21:33.20".

Returns: the color of X date time, for example, the color of text in "".

33 `svg_1d_plot & x_datetime_on(bool b);`

Set true if to append date time to data point X values near data points markers.

34 `bool x_datetime_on();`

Returns: true if to append an date time to data point X values near data points markers.

35 `svg_1d_plot & x_datetime_on(bool b);`

Set true if to append date time to data point X values near data points markers.

Set true if to append a datetime to data point X values near data points markers. (May not be implemented yet).

Set true if to append a datetime to data point X values near data points markers. (May not be implemented yet).

36 `bool x_datetime_on();`

Returns: true if to append an date time to data point X values near data points markers.

Returns: true if to append a ID or name to data point X values near data points markers. (May not be implemented yet).

Returns: true if to append a ID or name to data point X values near data points markers. (May not be implemented yet).

37 `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.

38 `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.

Set prefix, separator and suffix for x_style

**Note**

if you want a space, you must use a Unicode space "\ ", for example, ",\ " rather than just ",".

Set prefix, separator and suffix for x_style

**Note**

if you want a space, you must use a Unicode space "\ ", for example, ",\ " rather than just ",".

39 `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)".

40 `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)".

41 `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)".

Set the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

Set the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

42 `svg_color x_df_color();`

Get the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

Get the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

Returns: the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

43 `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.

44 `bool x_df_on();`

Returns: true if to append a degrees of freedom estimate to data point X values near data points markers.

45 `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.

Set true if to append a degrees of freedom estimate to data point X values near data points markers.

Set true if to append a degrees of freedom estimate to data point X values near data points markers.

36 `bool x_df_on();`

Returns: true if to append a degrees of freedom estimate to data point X values near data points markers.
 Returns: true if to append a degrees of freedom estimate to data point X values near data points markers. (May not be implemented yet).
 Returns: true if to append a degrees of freedom estimate to data point X values near data points markers. (May not be implemented yet).

37 `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".

38 `svg_color x_id_color();`

Returns: the color of X X id or name, for example, the color of text in "my_id".

39 `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".

Set the color of X ID or name, for example, the color of text in "My_id".

Set the color of X ID or name, for example, the color of text in "My_id".

40 `svg_color x_id_color();`

Get the color of X ID or name, for example, the color of text in "My_id".

Get the color of X ID or name, for example, the color of text in "My_id".

Returns: the color of X X id or name, for example, the color of text in "my_id".

41 `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.

42 `bool x_id_on();`

Returns: true if to append an ID or name to data point X values near data points markers.

43 `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.

Set true if to append a id or name to data point X values near data points markers. (May not be implemented yet).

Set true if to append a id or name to data point X values near data points markers. (May not be implemented yet).

44 `bool x_id_on();`

Returns: true if to append an ID or name to data point X values near data points markers.

Returns: true if to append an ID or name to data point X values near data points markers. (May not be implemented yet).

Returns: true if to append an ID or name to data point X values near data points markers. (May not be implemented yet).

35 `svg_1d_plot & x_label(const std::string & str);`

Set the text to label the X-axis (and set `x_label_on(true)`).

36 `std::string x_label();`

Returns: the text to label the X-axis.

37 `svg_1d_plot & x_label(const std::string & str);`

Set the text to label the X-axis (and set `x_label_on(true)`).

Set the text to label the X-axis (and set `x_label_on(true)`).

Set the text to label the X-axis (and set `x_label_on(true)`).

38 `std::string x_label();`

Returns: the text to label the X-axis.

Returns: the text to label the X-axis.

Returns: the text to label the X-axis.

39 `svg_1d_plot & x_label_color(const svg_color & col);`

Returns: the color of the Y-axis line.

40 `svg_color x_label_color();`

Returns: the color of X-axis label (including any units).

41 `svg_1d_plot & x_label_color(const svg_color & col);`

Set the color of X-axis label (including any units).

Set the color of X-axis label (including any units).

Returns: the color of the Y-axis line.

42 `svg_color x_label_color();`

Returns: the color of X-axis label (including any units).

Returns: the color of X-axis label (including any units).

Returns: the color of X-axis label (including any units).

43 `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"

34 `const std::string & x_label_font_family();`

Returns: X tick value label font family.

35 `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"
```

Set X tick value label font family.

Set X tick value label font family.

36 `const std::string & x_label_font_family();`

Returns: X tick value label font family.

Returns: X tick value label font family.

Returns: X tick value label font family.

37 `svg_1d_plot & x_label_font_size(unsigned int i);`

Set X axis label font size (svg units, default pixels).

Set X axis label font size (svg units, default pixels).

38 `unsigned int x_label_font_size();`

Returns: X axis label font size (svg units, default pixels).

39 `svg_1d_plot & x_label_font_size(int i);`

Set X axis label font size (svg units, default pixels).

Set X axis label font size (svg units, default pixels).

40 `int x_label_font_size();`

Returns: X axis label font size (svg units, default pixels).

Returns: X axis label font size (svg units, default pixels).

Returns: X axis label font size (svg units, default pixels).

41 `svg_1d_plot & x_label_on(bool cmd);`

Returns: true if X major ticks should mark downwards.

42 `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).

33 `svg_1d_plot & x_label_on(bool cmd);`

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).

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).

Returns: true if X major ticks should mark downwards.

34 `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).

Returns: true if want to show X-axis label text.

Returns: true if want to show X-axis label text.

35 `svg_1d_plot & x_label_units(const std::string & str);`

Set the text to add units to the X-axis label.

36 `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.

37 `svg_1d_plot & x_label_units(const std::string & str);`

Set the text to add units to the X-axis label.

Set the text to add units to the X-axis label.

Set the text to add units to the X-axis label.

38 `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.

Returns: the text to add units to the X-axis label. The label will only be shown if `x_label_on()` == true.

Returns: the text to add units to the X-axis label. The label will only be shown if `x_label_on()` == true.

39 `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.

40 `bool x_label_units_on();`

Set true if want X axis label to include units (as well as label like "length").

31 `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.

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.

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.

32 `bool x_label_units_on();`

Set true if want X axis label to include units (as well as label like "length").

Set true if want X axis label to include units (as well as label like "length").

Set true if want X axis label to include units (as well as label like "length").

33 `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).

34 `double x_label_width();`

Returns: the width (boldness) of X-axis label (including any units).

35 `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).

Set the width (boldness) of X-axis label (including any units). (not recommended until browsers implement better).

Set the width (boldness) of X-axis label (including any units). (not recommended until browsers implement better).

36 `double x_label_width();`

Returns: the width (boldness) of X-axis label (including any units).

Returns: The width (boldness) of X-axis label (including any units).

Returns: The width (boldness) of X-axis label (including any units).

37 `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.

38 `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.

Set true if want to strip redundant zeros, signs and exponents. **Example:** reducing "1.2e+000" to "1.2". This markedly reduces visual clutter, and is the default.

Set true if want to strip redundant zeros, signs and exponents. **Example:** reducing "1.2e+000" to "1.2". This markedly reduces visual clutter, and is the default.

39 `svg_1d_plot & x_major_grid_color(const svg_color & col);`

Set the color of X-axis major grid lines.

30 `svg_color x_major_grid_color();`

Set the color of X-axis major grid lines.

31 `svg_1d_plot & x_major_grid_color(const svg_color & col);`

Set the color of X-axis major grid lines.

Set the color of X-axis major grid lines.

Set the color of X-axis major grid lines.

32 `svg_color x_major_grid_color();`

Set the color of X-axis major grid lines.

Returns: the color of X-axis major grid lines.

Returns: the color of X-axis major grid lines.

33 `svg_1d_plot & x_major_grid_on(bool is);`

If set true, will include a major X-axis grid.

34 `bool x_major_grid_on();`

Returns: true if will include a major X-axis grid.

35 `svg_1d_plot & x_major_grid_on(bool is);`

If set true, will include a major X-axis grid.

If set true, will include a major X-axis grid.

If set true, will include a major X-axis grid.

36 `bool x_major_grid_on();`

If true, will include a major X-axis grid.

If true, will include a major X-axis grid.

Returns: true if will include a major X-axis grid.

37 `svg_1d_plot & x_major_grid_width(double w);`

Set the width of X-axis major grid lines.

38 `double x_major_grid_width();`

Returns: the color of X-axis major grid lines.

39 `svg_1d_plot & x_major_grid_width(double w);`

Set the width of X-axis major grid lines.

Set the width of X-axis major grid lines.

Set the width of X-axis major grid lines.

40 `double x_major_grid_width();`

Returns: the color of X-axis major grid lines.

Returns: the color of X-axis major grid lines.

Returns: the color of X-axis major grid lines.

41 `svg_1d_plot & x_major_interval(double inter);`

Set the interval between X-axis major ticks.

42 `double x_major_interval();`

Returns: the interval between X-axis major ticks.

43 `svg_1d_plot & x_major_interval(double inter);`

Set the interval between X-axis major ticks.

Set the interval between X-axis major ticks.

Set the interval between X-axis major ticks.

44 `double x_major_interval();`

Returns: the interval between X-axis major ticks.

Returns: the interval between X-axis major ticks.

Returns: the interval between X-axis major ticks.

45 `svg_1d_plot & x_major_label_rotation(rotate_style rot);`

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

`rotate_style`

46 `rotate_style x_major_label_rotation();`

See Also:

rotate_style

Returns: rotation for X ticks major value labels.

47 `svg_1d_plot & x_major_label_rotation(rotate_style rot);`

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

rotate_style

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

rotate_style

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

rotate_style

48 `rotate_style x_major_label_rotation();`**See Also:**

rotate_style

See Also:

rotate_style

See Also:

rotate_style

Returns: rotation for X ticks major value labels.

Returns: rotation for X ticks major value labels.

Returns: rotation for X ticks major value labels.

49 `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.

50 `int x_major_labels_side();`

Returns: the side for X ticks major value labels.

51 `svg_1d_plot & x_major_labels_side(int side);`

Position of labels for X major ticks on horizontal X axis line.

Position of labels for major ticks on horizontal X-axis line.

- place > 0 labels to left of Y-axis line (`left_side`) (default),
- place = 0 (false) no major tick labels on Y-axis.
- place > 0 labels to right of Y-axis line (`right_side`).

See Also:

`enum boost::svg::place` for named

Position of labels for major ticks on horizontal X-axis line.

- place > 0 labels to left of Y-axis line (`left_side`) (default),
- place = 0 (false) no major tick labels on Y-axis.
- place > 0 labels to right of Y-axis line (`right_side`).

See Also:

`enum boost::svg::place` for named

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.

42 `int x_major_labels_side();`

Returns: the side for X ticks major value labels.

Returns: The side for X ticks major value labels (see `enum boost::svg::side`).

Returns: The side for X ticks major value labels (see `enum boost::svg::side`).

43 `svg_1d_plot & x_major_tick(double d);`

Set interval (Cartesian units) between major ticks.

44 `double x_major_tick();`

Returns: interval (Cartesian units) between major ticks.

45 `svg_1d_plot & x_major_tick(double d);`

Set interval (Cartesian units) between major ticks.

Set interval (Cartesian units) between major ticks.

Set interval (Cartesian units) between major ticks.

46 `double x_major_tick();`

Returns: interval (Cartesian units) between major ticks.

Returns: interval (Cartesian units) between major ticks.

Returns: interval (Cartesian units) between major ticks.

47 `svg_1d_plot & x_major_tick_color(const svg_color & col);`

Set the color of X-axis major ticks.

48 `svg_color x_major_tick_color();`

Returns: the color of X-axis major ticks.

49 `svg_1d_plot & x_major_tick_color(const svg_color & col);`

Set the color of X-axis major ticks.

Set the color of X-axis major ticks.

Set the color of X-axis major ticks.

40 `svg_color x_major_tick_color();`

Returns: the color of X-axis major ticks.

Returns: the color of X-axis major ticks.

Returns: the color of X-axis major ticks.

41 `svg_1d_plot & x_major_tick_length(double length);`

Set length of X major ticks (SVG units, default pixels).

42 `double x_major_tick_length();`

Set length of X major ticks (SVG units, default pixels).

43 `svg_1d_plot & x_major_tick_length(double length);`

Set length of X major ticks (SVG units, default pixels).

Set length of X major ticks.

Set length of X major ticks.

44 `double x_major_tick_length();`

Set length of X major ticks (SVG units, default pixels).

Returns: length of X major ticks.

Returns: length of X major ticks.

45 `svg_1d_plot & x_major_tick_width(double width);`

Set width of X major ticks (SVG units, default pixels).

46 `double x_major_tick_width();`

Set width of X major ticks (SVG units, default pixels).

47 `svg_1d_plot & x_major_tick_width(double width);`

Set width of X major ticks (SVG units, default pixels).

Set width of X major ticks.

Set width of X major ticks.

48 `double x_major_tick_width();`

Set width of X major ticks (SVG units, default pixels).

Returns: width of X major ticks.

Returns: width of X major ticks.

49 `svg_1d_plot & x_max(double x);`

Set the maximum value on the X-axis.

40 `double x_max();`

autoscale set & get parameters, Note: all these *MUST* precede x_autoscale(data) call.

Returns: the maximum value on the X-axis.

41 `svg_1d_plot & x_max(double x);`

Set the maximum value on the X-axis.

Set the maximum value on the X-axis.

Set the maximum value on the X-axis.

42 `double x_max();`

autoscale set & get parameters, Note: all these *MUST* precede x_autoscale(data) call.

Returns: the maximum value on the X-axis.

Returns: the maximum value on the X-axis. (Can also get both minimum and maximum as a std::pair).

Returns: the maximum value on the X-axis. (Can also get both minimum and maximum as a std::pair).

43 `svg_1d_plot & x_min(double min_x);`

Set the minimum value on the X-axis.

44 `double x_min();`

Returns: the minimum value on the X-axis.

45 `svg_1d_plot & x_min(double min_x);`

Set the minimum value on the X-axis.

Set the minimum value on the X-axis.

Set the minimum value on the X-axis.

46 `double x_min();`

Returns: the minimum value on the X-axis.

Returns: the minimum value on the X-axis. (Can also get both minimum and maximum as a std::pair).

Returns: the minimum value on the X-axis. (Can also get both minimum and maximum as a std::pair).

47 `svg_1d_plot & x_min_ticks(int min_ticks);`

Set X-axis autoscale to include at least minimum number of ticks (default = 6).

48 `int x_min_ticks();`

Returns: X-axis autoscale minimum number of ticks.

49 `svg_1d_plot & x_min_ticks(int min_ticks);`

Set X-axis autoscale to include at least minimum number of ticks (default = 6).

Set X-axis autoscale to include at least minimum number of ticks (default = 6). Must precede x_autoscale(data) call.

Set X-axis autoscale to include at least minimum number of ticks (default = 6). Must precede x_autoscale(data) call.

40 `int x_min_ticks();`

Returns: X-axis autoscale minimum number of ticks.

Returns: X-axis autoscale minimum number of ticks.

Returns: X-axis autoscale minimum number of ticks.

41 `svg_1d_plot & x_minor_grid_color(const svg_color & col);`

Set the color of X-axis minor grid lines.

42 `svg_color x_minor_grid_color();`

Returns: the color of X-axis minor grid lines.

43 `svg_1d_plot & x_minor_grid_color(const svg_color & col);`

Set the color of X-axis minor grid lines.

Set the color of X-axis minor grid lines.

Set the color of X-axis minor grid lines.

44 `svg_color x_minor_grid_color();`

Returns: the color of X-axis minor grid lines.

Returns: the color of X-axis minor grid lines.

Returns: the color of X-axis minor grid lines.

45 `svg_1d_plot & x_minor_grid_on(bool is);`

If set true, will include a minor X-axis grid.

46 `bool x_minor_grid_on();`

Returns: true if will include a major X-axis grid.

47 `svg_1d_plot & x_minor_grid_on(bool is);`

If set true, will include a minor X-axis grid.

If set true, will include a minor X-axis grid.

If set true, will include a minor X-axis grid.

48 `bool x_minor_grid_on();`

If true, will include a minor X-axis grid.

If true, will include a minor X-axis grid.

Returns: true if will include a major X-axis grid.

49 `svg_1d_plot & x_minor_grid_width(double w);`

Set the width of X-axis minor grid lines.

50 `double x_minor_grid_width();`

Returns: the width of X-axis minor grid lines.

51 `svg_1d_plot & x_minor_grid_width(double w);`

Set the width of X-axis minor grid lines.

Set the width of X-axis minor grid lines.

Set the width of X-axis minor grid lines.

52 `double x_minor_grid_width();`

Returns: the width of X-axis minor grid lines.

Returns: the width of X-axis minor grid lines.

Returns: the width of X-axis minor grid lines.

53 `double x_minor_interval();`

Returns: interval between X minor ticks.

54 `double x_minor_interval();`

Returns: interval between X minor ticks.

Returns: interval between X minor ticks.

Returns: interval between X minor ticks.

55 `svg_1d_plot & x_minor_interval(double interval);`

Set interval between X-axis minor ticks.

46 `svg_1d_plot & x_minor_interval(double interval);`

Set interval between X-axis minor ticks.

Set interval between X-axis minor ticks.

Set interval between X-axis minor ticks.

47 `svg_1d_plot & x_minor_tick_color(const svg_color & col);`

Set the color of X-axis minor ticks.

48 `svg_color x_minor_tick_color();`

Returns: the color of X-axis minor ticks.

49 `svg_1d_plot & x_minor_tick_color(const svg_color & col);`

Set the color of X-axis minor ticks.

Set the color of X-axis minor ticks.

Set the color of X-axis minor ticks.

50 `svg_color x_minor_tick_color();`

Returns: the color of X-axis minor ticks.

Returns: the color of X-axis minor ticks.

Returns: the color of X-axis minor ticks.

51 `svg_1d_plot & x_minor_tick_length(double length);`

Set length of X minor ticks (SVG units, default pixels).

52 `double x_minor_tick_length();`

Returns: length of X minor ticks (SVG units, default pixels).

53 `svg_1d_plot & x_minor_tick_length(double length);`

Set length of X minor ticks (SVG units, default pixels).

Set length of X minor ticks.

Set length of X minor ticks.

54 `double x_minor_tick_length();`

Returns: length of X minor ticks (SVG units, default pixels).

Returns: length of X minor ticks.

Returns: length of X minor ticks.

45 `svg_1d_plot & x_minor_tick_width(double width);`

Set width of X minor ticks (SVG units, default pixels).

46 `double x_minor_tick_width();`

Returns: width of X minor ticks (SVG units, default pixels).

47 `svg_1d_plot & x_minor_tick_width(double width);`

Set width of X minor ticks (SVG units, default pixels).

Set width of X minor ticks.

Set width of X minor ticks.

48 `double x_minor_tick_width();`

Returns: width of X minor ticks (SVG units, default pixels).

Returns: width of X minor ticks.

Returns: width of X minor ticks.

49 `svg_1d_plot & x_num_minor_ticks(unsigned int num);`

Set number of X-axis minor ticks between major ticks.

Set number of X-axis minor ticks between major ticks.

50 `unsigned int x_num_minor_ticks();`

Returns: number of X-axis minor ticks between major ticks.

51 `svg_1d_plot & x_num_minor_ticks(int num);`

Set number of X-axis minor ticks between major ticks.

Set number of X-axis minor ticks between major ticks.

52 `int x_num_minor_ticks();`

Returns: number of X-axis minor ticks between major ticks.

Returns: number of X-axis minor ticks between major ticks. Note: NOT float or double!

Returns: number of X-axis minor ticks between major ticks. Note: NOT float or double!

53 `svg_1d_plot & x_order_color(const svg_color & col);`

Set the color of X order #, for example, the color of #42.

54 `svg_color x_order_color();`

Returns: the color of X order #, for example, the color of #42.

45 `svg_1d_plot & x_order_color(const svg_color & col);`

Set the color of X order #, for example, the color of #42.

Set the color of X point order in sequence, for example, #3.

Set the color of X point order in sequence, for example, #3.

46 `svg_color x_order_color();`

Get the color of X point order in sequence, for example, #3.

Get the color of X point order in sequence, for example, #3.

Returns: the color of X order #, for example, the color of #42.

47 `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.

48 `bool x_order_on();`

Returns: true if to append an order # to data point X values near data points markers.

49 `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.

Set true if to append an order # to data point X values near data points markers.

Set true if to append an order # to data point X values near data points markers.

40 `bool x_order_on();`

Returns: true if to append an order # to data point X values near data points markers.

Returns: true if to append an order # to data point X values near data points markers.

Returns: true if to append an order # to data point X values near data points markers.

41 `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)".

42 `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)".

43 `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)".

Set the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

Set the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

44 `svg_color x_plusminus_color();`

Get the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

Get the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

Returns: the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

45 `svg_1d_plot & x_plusminus_on(bool b);`

Set if to append std_dev estimate to data point X values near data points markers.

46 `bool x_plusminus_on();`

Returns: true if to append std_dev estimate to data point X values near data points markers.

47 `svg_1d_plot & x_plusminus_on(bool b);`

Set if to append std_dev estimate to data point X values near data points markers.

Set if to append std_dev estimate to data point X values near data points markers. (May not be implemented yet).

Set if to append std_dev estimate to data point X values near data points markers. (May not be implemented yet).

48 `bool x_plusminus_on();`

Returns: true if to append std_dev estimate to data point X values near data points markers.

Returns: if to append std_dev estimate to data point X values near data points markers. (May not be implemented yet).

Returns: if to append std_dev estimate to data point X values near data points markers. (May not be implemented yet).

49 `const std::string x_prefix();`

Returns: the prefix.

50 `const std::string x_prefix();`

Get the prefix (only used if separator != "")

Get the prefix (only used if separator != "")

Returns: the prefix.

51 `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.

52 `std::pair< double, double > x_range();`

Returns: the range of values on the X-axis.

53 `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.

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 doubles, and the range must not be too small.

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 doubles, and the range must not be too small.

41 `std::pair< double, double > x_range();`

Returns: the range of values on the X-axis.

Returns: the range of values on the X-axis. (Need to use boost::svg::detail::operator<< to display this).

Returns: the range of values on the X-axis. (Need to use boost::svg::detail::operator<< to display this).

45 `const std::string x_separator();`

Returns: the separator, perhaps including Unicode.

46 `const std::string x_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;", for example, "\u00A0;" rather than " ".

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;", for example, "\u00A0;" rather than " ".

Returns: the separator, perhaps including Unicode.

47 `svg_1d_plot & x_size(unsigned int i);`

Set SVG image X-axis size (SVG units, default pixels).

Set SVG image X-axis size (SVG units, default pixels).

48 `unsigned int x_size();`

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

49 `svg_1d_plot & x_size(int i);`

Set SVG image X-axis size (SVG units, default pixels).

Set SVG image X-axis size (SVG units, default pixels).

50 `int x_size();`

Get SVG image X-axis size as horizontal width (SVG units, default pixels). Get SVG image X-axis size as horizontal width (SVG units, default pixels).

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

51 `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).

52 `int x_steps();`

Returns: autoscale to set ticks in steps.

53 `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).

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).

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).

54 `int x_steps();`

Returns: autoscale to set ticks in steps.

Returns: autoscale to set ticks in steps.
 Returns: autoscale to set ticks in steps.

55 `const std::string x_suffix();`

Returns: the suffix (only used if separator != "")

56 `const std::string x_suffix();`

Get the suffix (only used if separator != "")

Get the suffix (only used if separator != "")

Returns: the suffix (only used if separator != "")

57 `svg_1d_plot & x_ticks_down_on(bool cmd);`

Set true if Y major ticks should mark upwards.

58 `bool x_ticks_down_on();`

Returns: true if Y major ticks should mark upwards.

59 `svg_1d_plot & x_ticks_down_on(bool cmd);`

Set true if Y major ticks should mark upwards.

Set true if X major ticks should mark downwards.

Set true if X major ticks should mark downwards.

60 `bool x_ticks_down_on();`

Returns: true if Y major ticks should mark upwards.

Returns: true if X major ticks should mark downwards.

Returns: true if X major ticks should mark downwards.

61 `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.

62 `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.

63 `svg_1d_plot & x_ticks_on_window_or_axis(int side);`

Set position of X ticks on window or axis.

Set X ticks on window or axis

- cmd -1 bottom of plot window,

- cmd 0 on X axis.
- cmd +1 top of plot window.
Set X ticks on window or axis
- cmd -1 bottom of plot window,
- cmd 0 on X axis.
- cmd +1 top of plot window.
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.

54 `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.
- Returns: if X axis ticks wanted on the window (rather than on axis).
 -1 bottom of plot window, 0 on X axis, +1 top of plot window.
- Returns: if X axis ticks wanted on the window (rather than on axis).
 -1 bottom of plot window, 0 on X axis, +1 top of plot window.

55 `svg_1d_plot & x_ticks_up_on(bool cmd);`

Set true if X major ticks should mark upwards.

56 `bool x_ticks_up_on();`

Returns: true if X major ticks should mark upwards.

57 `svg_1d_plot & x_ticks_up_on(bool cmd);`

Set true if X major ticks should mark upwards.

Set true if X major ticks should mark upwards.

Set true if X major ticks should mark upwards.

58 `bool x_ticks_up_on();`

Returns: true if X major ticks should mark upwards.

Returns: true if X major ticks should mark upwards.

Returns: true if X major ticks should mark upwards.

59 `svg_1d_plot & x_ticks_values_color(const svg_color & col);`

Set X axis tick value label color.

60 `svg_color x_ticks_values_color();`

Returns: X-axis ticks value label color.

61 `svg_1d_plot & x_ticks_values_color(const svg_color & col);`

Set X axis tick value label color.

Set X axis tick value label color.

Set X axis tick value label color.

52 `svg_color x_ticks_values_color();`

Returns: X-axis ticks value label color.

Returns: X-axis ticks value label color.

Returns: X-axis ticks value label color.

53 `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"
```

54 `const std::string & x_ticks_values_font_family();`

Returns: X ticks value label font family.

55 `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"
```

Set X ticks value label font family.

Set X ticks value label font family.

56 `const std::string & x_ticks_values_font_family();`

Returns: X ticks value label font family.

Returns: X ticks value label font family.

Returns: X ticks value label font family.

57 `svg_1d_plot & x_ticks_values_font_size(unsigned int i);`

Set X ticks value label font size (svg units, default pixels).

Set X ticks value label font size (svg units, default pixels).

58 `unsigned int x_ticks_values_font_size();`

Set X ticks value label font size (svg units, default pixels).

59 `svg_1d_plot & x_ticks_values_font_size(int i);`

Set X ticks value label font size (svg units, default pixels).

Set X ticks value label font size (svg units, default pixels).

50 `int x_ticks_values_font_size();`

Set X ticks value label font size (svg units, default pixels).

Returns: X ticks value label font size (svg units, default pixels).

Returns: X ticks value label font size (svg units, default pixels).

51 `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.

52 `std::ios_base::fmtflags x_ticks_values_ioflags();`

Returns: iostream format flags of data point X values near data points markers.

53 `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.

Set iostream format flags of data point X values near data points markers. Useful to set hexadecimal, fixed and scientific, (std::ios::scientific).

Set iostream format flags of data point X values near data points markers. Useful to set hexadecimal, fixed and scientific, (std::ios::scientific).

54 `std::ios_base::fmtflags x_ticks_values_ioflags();`

Returns: iostream format flags of data point X values near data points markers.

Returns: iostream format flags of data point X values near data points markers. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

Returns: iostream format flags of data point X values near data points markers. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

55 `svg_1d_plot & x_ticks_values_precision(int p);`

Set iostream decimal digits precision of data point X values near data points markers.

56 `int x_ticks_values_precision();`

Returns: iostream decimal digits precision of data point X values near data points markers.

57 `svg_1d_plot & x_ticks_values_precision(int p);`

Set iostream decimal digits precision of data point X values near data points markers.

Set iostream decimal digits precision of data point X values near data points markers.

Set iostream decimal digits precision of data point X values near data points markers.

58 `int x_ticks_values_precision();`

Returns: ostream decimal digits precision of data point X values near data points markers.
 Returns: ostream decimal digits precision of data point X values near data points markers.
 Returns: ostream decimal digits precision of data point X values near data points markers.

59 `svg_1d_plot & x_tight(double tight);`

Set tolerance to autoscale to permit data points slightly outside both end ticks.

60 `double x_tight();`

Returns: tolerance given to autoscale to permit data points slightly outside both end ticks. Get results of autoscaling.

61 `svg_1d_plot & x_tight(double tight);`

Set tolerance to autoscale to permit data points slightly outside both end ticks.

Set tolerance to autoscale to permit data points slightly outside both end ticks. default 0. Must precede x_autoscale(data) call.

Set tolerance to autoscale to permit data points slightly outside both end ticks. default 0. Must precede x_autoscale(data) call.

62 `double x_tight();`

Returns: tolerance given to autoscale to permit data points slightly outside both end ticks. Get results of autoscaling.

Returns: tolerance given to autoscale to permit data points slightly outside both end ticks.

Returns: tolerance given to autoscale to permit data points slightly outside both end ticks.

58 `svg_1d_plot & x_value_font_size(unsigned int i);`

Set X tick value label font size (svg units, default pixels).

Set X tick value label font size (svg units, default pixels).

59 `unsigned int x_value_font_size();`

Returns: X tick value label font size (svg units, default pixels).

55 `svg_1d_plot & x_value_font_size(int i);`

Set X tick value label font size (svg units, default pixels).

Set X tick value label font size (svg units, default pixels).

56 `int x_value_font_size();`

Returns: X tick value label font size (svg units, default pixels).

Returns: X tick value label font size (svg units, default pixels).

Returns: X tick value label font size (svg units, default pixels).

57 `svg_1d_plot & 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)
```

58 `std::ios_base::fmtflags x_value_ioflags();`

Returns: stream std::ios::fmtflags for control of format of X value labels.

59 `svg_1d_plot & 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)
```

Set iostream std::ios::fmtflags for X value label (default decimal == 0X201). Mainly useful for changing to scientific, fixed or hexadecimal format. For example: .x_value_ioflags(std::ios::dec | std::ios::scientific)

Set iostream std::ios::fmtflags for X value label (default decimal == 0X201). Mainly useful for changing to scientific, fixed or hexadecimal format. For example: .x_value_ioflags(std::ios::dec | std::ios::scientific)

50 `std::ios_base::fmtflags x_value_ioflags();`

Returns: stream std::ios::fmtflags for control of format of X value labels.

Returns: stream ioflags for control of format of X value labels.

Returns: stream ioflags for control of format of X value labels.

51 `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.

52 `int x_value_precision();`

Returns: Precision of X-tick label values in decimal digits

53 `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.

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.

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.

54 `int x_value_precision();`

Returns: Precision of X-tick label values in decimal digits

Returns: precision of X tick label values in decimal digits

Returns: precision of X tick label values in decimal digits

55 `svg_1d_plot & x_values_color(const svg_color & col);`

Set the color of data point X values near data points markers.

56 `svg_color x_values_color();`

Returns: the color of data point X values near data points markers.

57 `svg_1d_plot & x_values_color(const svg_color & col);`

Set the color of data point X values near data points markers.

Set the color of data point X values near data points markers.

Set the color of data point X values near data points markers.

58 `svg_color x_values_color();`

Returns: the color of data point X values near data points markers.

Returns: the color of data point X values near data points markers.

Returns: the color of data point X values near data points markers.

59 `svg_1d_plot & x_values_font_family(const std::string & family);`

Set font family of data point X values near data points markers.

60 `const std::string & x_values_font_family();`

Returns: font family of data point X values near data points markers.

61 `svg_1d_plot & x_values_font_family(const std::string & family);`

Set font family of data point X values near data points markers.

Set font family of data point X values near data points markers.

Set font family of data point X values near data points markers.

62 `const std::string & x_values_font_family();`

Set font family of data point X values near data points markers.

Set font family of data point X values near data points markers.

Returns: font family of data point X values near data points markers.

63 `svg_1d_plot & x_values_font_size(unsigned int i);`

Set font size of data point X values near data points markers.

Set font size of data point X values near data points markers.

54 `unsigned int x_values_font_size();`

Returns: font size of data point X values near data points markers.

55 `svg_1d_plot & x_values_font_size(int i);`

Set font size of data point X values near data points markers.

Set font size of data point X values near data points markers.

56 `int x_values_font_size();`

Returns: font size of data point X values near data points markers.

Returns: font size of data point X values near data points markers.

Returns: font size of data point X values near data points markers.

57 `svg_1d_plot & x_values_ioflags(std::ios_base::fmtflags f);`

Set ostream format flags of data point X values near data points markers.

58 `std::ios_base::fmtflags x_values_ioflags();`

Returns: ostream format flags of data point X values near data points markers.

59 `svg_1d_plot & x_values_ioflags(std::ios_base::fmtflags f);`

Set ostream format flags of data point X values near data points markers.

Set ostream format flags of data point X values near data points markers. Useful to set hexadecimal, fixed and scientific, (std::ios::scientific).

Set ostream format flags of data point X values near data points markers. Useful to set hexadecimal, fixed and scientific, (std::ios::scientific).

60 `std::ios_base::fmtflags x_values_ioflags();`

Returns: ostream format flags of data point X values near data points markers.

Returns: ostream format flags of data point X values near data points markers. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

Returns: ostream format flags of data point X values near data points markers. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

61 `svg_1d_plot & x_values_on(bool b);`

Set true to show data point values near data points markers.

62 `bool x_values_on();`

Returns: true if to show data point values near data points markers.

63 `svg_1d_plot & x_values_on(bool b);`

Set true to show data point values near data points markers.

Returns: true if values of X data points are shown (for example: 1.23).
 Returns: true if values of X data points are shown (for example: 1.23).

54 `bool x_values_on();`

If true, show data point values near data points markers.

If true, show data point values near data points markers.
 Returns: true if to show data point values near data points markers.

55 `svg_1d_plot & x_values_precision(int p);`

Set iostream decimal digits precision of data point X values near data points markers.

56 `int x_values_precision();`

Returns: iostream decimal digits precision of data point X values near data points markers.

57 `svg_1d_plot & x_values_precision(int p);`

Set iostream decimal digits precision of data point X values near data points markers.

Set iostream decimal digits precision of data point X values near data points markers.

Set iostream decimal digits precision of data point X values near data points markers.

58 `int x_values_precision();`

Returns: iostream decimal digits precision of data point X values near data points markers.
 Returns: iostream decimal digits precision of data point X values near data points markers.
 Returns: iostream decimal digits precision of data point X values near data points markers.

59 `svg_1d_plot & x_values_rotation(rotate_style rotate);`

Returns: the rotation (rotate_style) of data point X values near data points markers.

60 `int x_values_rotation();`

Set the rotation (rotate_style) of data point X values near data points markers.

61 `svg_1d_plot & x_values_rotation(rotate_style rotate);`

Returns: the rotation (rotate_style) of data point X values near data points markers.
 Returns: the rotation (rotate_style) of data point X values near data points markers. (Degrees: 0 to 360 in 45 steps).
 Returns: the rotation (rotate_style) of data point X values near data points markers. (Degrees: 0 to 360 in 45 steps).

62 `int x_values_rotation();`

Set the rotation (rotate_style) of data point X values near data points markers.

Returns: the rotation of data point X values near data points markers.

Returns: the rotation of data point X values near data points markers.

53 `svg_1d_plot & x_with_zero(bool b);`

Set X-axis autoscale to include zero (default = false).

54 `bool x_with_zero();`

Returns: true if X-axis autoscale to include zero (default = false).

55 `svg_1d_plot & x_with_zero(bool b);`

Set X-axis autoscale to include zero (default = false).

Set X-axis autoscale to include zero (default = false). Must preceed x_autoscale(data) call.

Set X-axis autoscale to include zero (default = false). Must preceed x_autoscale(data) call.

56 `bool x_with_zero();`

Returns: true if X-axis autoscale to include zero (default = false).

Returns: true if X-axis autoscale to include zero (default = false).

Returns: true if X-axis autoscale to include zero (default = false).

57 `svg_1d_plot & y_axis_color(const svg_color & col);`

Set the color of the Y-axis line.

58 `svg_color y_axis_color();`

Returns: the color of the Y-axis line.

59 `svg_1d_plot & y_axis_color(const svg_color & col);`

Set the color of the Y-axis line.

Set the color of the Y-axis line.

Set the color of the Y-axis line.

60 `svg_color y_axis_color();`

Returns: the color of the Y-axis line.

Returns: The color of the Y-axis line.

Returns: The color of the Y-axis line.

61 `svg_1d_plot & y_axis_on(bool is);`

If set true, draw a vertical Y-axis line.

62 `bool y_axis_on();`

Returns: true if will draw a horizontal X-axis line.

58 `svg_1d_plot & y_axis_on(bool is);`

If set true, draw a vertical Y-axis line.

If set true, draw a vertical Y-axis line.

If set true, draw a vertical Y-axis line.

59 `bool y_axis_on();`

If true, draw a vertical Y-axis line.

If true, draw a vertical Y-axis line.

Returns: true if will draw a horizontal X-axis line.

55 `svg_1d_plot & y_label(const std::string & str);`

Set the text for the Y-axis label (and set `y_label_on(true)`).

56 `std::string y_label();`

Returns: the text for the Y-axis label. The label will only be shown if `y_label_on() == true`.

57 `svg_1d_plot & y_label(const std::string & str);`

Set the text for the Y-axis label (and set `y_label_on(true)`).

Set the text for the Y-axis label (and set `y_label_on(true)`).

Set the text for the Y-axis label (and set `y_label_on(true)`).

58 `std::string y_label();`

Returns: the text for the Y-axis label. The label will only be shown if `y_label_on() == true`.

Returns: the text for the Y-axis label. The label will only be shown if `y_label_on() == true`.

Returns: the text for the Y-axis label. The label will only be shown if `y_label_on() == true`.

59 `svg_1d_plot & y_label_color(const svg_color & col);`

Set the color of Y-axis label (including any units).

60 `svg_color y_label_color();`

Returns: the color of Y-axis label (including any units).

61 `svg_1d_plot & y_label_color(const svg_color & col);`

Set the color of Y-axis label (including any units).

Set the color of Y-axis label (including any units).

Set the color of Y-axis label (including any units).

62 `svg_color y_label_color();`

Returns: the color of Y-axis label (including any units).
 Returns: the color of Y-axis label (including any units).
 Returns: the color of Y-axis label (including any units).

63 `svg_1d_plot & y_label_units(const std::string & str);`

Set the text to add units to the Y-axis label.

64 `std::string y_label_units();`

Returns: the text to add units to the X-axis label.

65 `svg_1d_plot & y_label_units(const std::string & str);`

Set the text to add units to the Y-axis label.

Set the text to add units to the Y-axis label.

Set the text to add units to the Y-axis label.

66 `std::string y_label_units();`

Returns: the text to add units to the X-axis label.
 Returns: the text to add units to the X-axis label.
 Returns: the text to add units to the X-axis label.

67 `bool y_labels_strip_e0s();`

Returns: if to strip redundant zeros, signs and exponents, for example, reducing "1.2e+000" to "1.2"

68 `bool y_labels_strip_e0s();`

Returns: if to strip redundant zeros, signs and exponents, for example, reducing "1.2e+000" to "1.2"
 Returns: true if set to strip redundant zeros, signs and exponents. **Example:** reducing "1.2e+000" to "1.2".
 Returns: true if set to strip redundant zeros, signs and exponents. **Example:** reducing "1.2e+000" to "1.2".

69 `double y_minor_interval();`

Returns: interval between Y minor ticks.

70 `double y_minor_interval();`

Returns: interval between Y minor ticks.
 Returns: interval between Y minor ticks.
 Returns: interval between Y minor ticks.

71 `unsigned int y_size();`

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

62 `svg_1d_plot & y_size(unsigned int i);`

Set SVG image Y-axis size (SVG units, default pixels).

Set SVG image Y-axis size (SVG units, default pixels).

63 `int y_size();`

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

64 `svg_1d_plot & y_size(int i);`

Set SVG image Y-axis size (SVG units, default pixels).

Set SVG image Y-axis size (SVG units, default pixels).

svg_1d_plot protected member functions

1. `void adjust_limits(double & x, double & y);`

If value of a data point reaches limit of max, min, infinity, use the appropriate plot min or max value instead.

If value of a data point reaches limit of max, min, infinity, use the appropriate plot min or max value instead.

2. `void adjust_limits(double & x, double & y);`

Clear all layers of the plot.

When writing to multiple documents, the contents of the plot may change significantly between. Rather than figuring out what has and has not changed, just erase the contents of the legend, title... in the document and start over.

3. `void clear_all();`

Clear all layers of the plot.

When writing to multiple documents, the contents of the plot may change significantly between. Rather than figuring out what has and has not changed, just erase the contents of the legend, title... in the document and start over.

4. `void clear_all();`

Clear the whole image background layer of the SVG plot.

5. `void clear_background();`

Clear the whole image background layer of the SVG plot.

< Clear the whole image background layer of the SVG plot.

< Clear the whole image background layer of the SVG plot.

7. `void clear_grids();`

Clear the grids layer of the SVG plot.

8. `void clear_grids();`

Clear the grids layer of the SVG plot.

< Clear the grids layer of the SVG plot.

< Clear the grids layer of the SVG plot.

9. `void clear_legend();`

Clear the legend layer of the SVG plot.

10. `void clear_legend();`

Clear the legend layer of the SVG plot.

< Clear the legend layer of the SVG plot.

< Clear the legend layer of the SVG plot.

11. `void clear_plot_background();`

Clear the plot area background layer of the SVG plot.

12. `void clear_plot_background();`

Clear the plot area background layer of the SVG plot.

< Clear the plot area background layer of the SVG plot.

< Clear the plot area background layer of the SVG plot.

13. `void clear_points();`

Clear the data points layer of the SVG plot.

14. `void clear_points();`

Clear the data points layer of the SVG plot.

< Clear the data points layer of the SVG plot.

< Clear the data points layer of the SVG plot.

15. `void clear_title();`

Clear the plot title layer of the SVG plot.

16. `void clear_title();`

Clear the Y axis layer of the SVG plot.

Clear the plot title layer of the SVG plot.

< Clear the plot title layer of the SVG plot.

< Clear the plot title layer of the SVG plot.

17. `void clear_x_axis();`

Clear the X axis layer of the SVG plot.

18. `void clear_x_axis();`

Clear the X axis layer of the SVG plot.

< Clear the X axis layer of the SVG plot.

< Clear the X axis layer of the SVG plot.

19. `void clear_y_axis();`

Clear the Y axis layer of the SVG plot.

20. `void clear_y_axis();`

< Clear the Y axis layer of the SVG plot.

< Clear the Y axis layer of the SVG plot.

21. `void draw_legend();`

22. `void draw_legend();`

Draw the legend border and background, (using the size and position computed by function `size_legend_box`), and legend text title (if any and if required), and any data point marker lines, and any shapes for data point markers, and any data series descriptor text(s).

Draw the legend border, text header (if any) and data point marker lines and/or shapes.

23. `void draw_plot_point(double x, double y, g_element & g_ptr,
const plot_point_style & sty);`

24. `void draw_plot_point(double x, double y, g_element & g_ptr,
plot_point_style & sty, unc< false > ux,
unc< false > uy);`

25. `void draw_plot_point(double x, double y, g_element & g_ptr,
const plot_point_style & sty);`

26. `void draw_plot_point(double x, double y, g_element & g_ptr,
plot_point_style & sty, unc< false > ux,
unc< false > uy);`

Draw a plot data point marker shape or symbol whose size and stroke and fill colors are specified in plot_point_style sty, possibly including uncertainty ellipses showing multiples of standard deviation.

Draw a plot data point marker shape or symbol whose size and stroke and fill colors are specified in plot_point_style sty, possibly including uncertainty ellipses showing multiples of standard deviation.

27. `void draw_plot_point_value(double x, double y, g_element & g_ptr,
value_style & val_style,
plot_point_style & point_style, Meas uvalue);`

28. `void draw_plot_point_value(double x, double y, g_element & g_ptr,
value_style & val_style,
plot_point_style & point_style, Meas uvalue);`

`void draw_plot_point_value(double x, double y, g_element& g_ptr, value_style& val_style, plot_point_style& point_style, unc<false> uvalue)` Write one data point (X or Y) value as a string, for example "1.23e-2", near the data point marker. Unnecessary e, +, & leading exponent zeros may optionally be stripped, and the position and rotation controlled. std_dev estimate, typically standard deviation (approximately half conventional 95% confidence "plus or minus") may be optionally be appended. Degrees of freedom estimate (number of replicates) may optionally be appended. ID or name of point, order in sequence, and datetime may also be added. For example: "3.45 +-0.1(10)"

The precision and format (scientific, fixed), and color and font type and size can be controlled too.

Unicode space text_plusminus glyph.

`void draw_plot_point_value(double x, double y, g_element& g_ptr, value_style& val_style, plot_point_style& point_style, unc<false> uvalue)` Write one data point (X or Y) value as a string, for example "1.23e-2", near the data point marker. Unnecessary e, +, & leading exponent zeros may optionally be stripped (highly recommended), and the position and rotation controlled. std_dev estimate, typically standard deviation (approximately half conventional 95% confidence "plus or minus") may be optionally be appended. Degrees of freedom estimate (number of replicates) may optionally be appended. ID or name of point, order in sequence, and datetime may also be added. For example: "3.45 +-0.1(10)"

The precision and format (scientific, fixed), and color and font type and size can be controlled too.

Unicode space and text_plusminus glyph.

29. `void draw_plot_point_values(double x, double y, g_element & x_g_ptr,
g_element & y_g_ptr, const value_style & x_sty,
const value_style & y_sty, Meas uncx, Meas uncny);`

30. `void draw_plot_point_values(double x, double y, g_element & x_g_ptr,
g_element & y_g_ptr, const value_style & x_sty,
const value_style & y_sty, Meas uncx, Meas uncny,
unc< false > uncny);`

31. `void draw_plot_point_values(double x, double y, g_element & x_g_ptr,
g_element & y_g_ptr, const value_style & x_sty,
const value_style & y_sty, Meas uncx, Meas uncny);`

32 `void draw_plot_point_values(double x, double y, g_element & x_g_ptr,
 g_element & y_g_ptr, const value_style & x_sty,
 const value_style & y_sty, Meas uncx,
 unc< false > uncy);`

Write the **pair** of data points X and Y values as a string.

The x parameter also carries the measurement information for the pair, and so is a **Meas**, not just an **unc<false>** as is the Y parameter. If a separator starting with newline, then both on the same line, for example "1.23, 3.45", or "[5.6, 7.8] X value_style is used to provide the prefix and separator, and Y value_style to provide the suffix. For example,

`x_style prefix("[X=", and separator ",\u00A0;Y= ", " and Y value_style = "]")` will produce a value label like "[X=-1.23, Y=4.56]"

Note



You need to use a Unicode space for get space for all browsers. For a long a string you may need to make the total image size bigger, and to orient the value labels with care. `draw_plot_point_values` is only when both X and Y pairs are wanted.

Also strip unnecessary e, + and leading exponent zeros, if required.

Unicode space text_plusminus glyph.

Write the **pair** of data points X and Y values as a string.

The x parameter also carries the measurement information for the pair, and so is a **Meas**, not just an **unc<false>** as is the Y parameter. If a separator starting with newline, then both on the same line, for example "1.23, 3.45", or "[5.6, 7.8] X value_style is used to provide the prefix and separator, and Y value_style to provide the suffix. For example,

`x_style prefix("[X=", and separator ",\u00A0;Y= ", " and Y value_style = "]")` will produce a value label like "[X=-1.23, Y=4.56]"

Note



You need to use a Unicode space for get space for all browsers. For a long a string you may need to make the total image size bigger, and to orient the value labels with care. `draw_plot_point_values` is only when both X and Y pairs are wanted.

Also strip unnecessary e, + and leading exponent zeros, if required.

Unicode space text_plusminus glyph.

33. `void draw_title();`

34. `void draw_title();`

Draw title (for the whole plot). Update `title_info_` with position. Assumes `align = center_align` Using `center_align` will ensure that title will center correctly because the render engine does the centering. (Even if the original string is made much longer because it contains Unicode, Greek, math symbols etc, taking about 8 characters per symbol. For example, the Unicode symbol for square root is "\u221A," but takes only about one character width).

Draw title (for the whole plot). Update `title_info_` with position. Assumes `align = center_align` Using `center_align` will ensure that title will center correctly because the render engine does the centering. (Even if the original string is made much longer because

it contains Unicode, Greek, math symbols etc, taking about 8 characters per symbol. For example, the Unicode symbol for square root is "√" but takes only about one character width).

35. `void draw_x_axis();`

36. `void draw_x_axis();`

Draw horizontal X-axis line & plot window line to hold, and ticks and grids.

Draw horizontal X-axis line & plot window line to hold, and ticks and grids.

37. `void draw_x_axis_label();`

38. `void draw_x_axis_label();`

Draw the X-axis label text (for example, length), and append any optional units (for example, km).

Draw the X-axis label text (for example, length), and append any optional units (for example, km).

39. `void draw_x_major_tick(double i, path_element & tick_path,
path_element & grid_path);`

40. `void draw_x_major_tick(double i, path_element & tick_path,
path_element & grid_path);`

Draw major ticks - and grid too if wanted. If major_value_labels_side then value shown beside the major tick.

Draw major ticks - and grid too if wanted. If major_value_labels_side then value shown beside the major tick.

41. `void draw_x_minor_tick(double j, path_element & tick_path,
path_element & grid_path);`

42. `void draw_x_minor_tick(double j, path_element & tick_path,
path_element & grid_path);`

< Draw X-axis minor ticks, and optional grid. (Value is NOT (yet) shown beside the minor tick).

< Draw X-axis minor ticks, and optional grid. (Value is NOT (yet) shown beside the minor tick).

43. `void place_legend_box();`

44. `void place_legend_box();`

Place legend box (if required). Default legend position is outside top right, level with plot window.

Place legend box (if required).

45. `void size_legend_box();`

46. `void size_legend_box();`

Calculate how big the legend box needs to be to hold the legend title and the data point markers (symbols or shapes), and any line marks showing lines used joining points, and any data series descriptor text(s).

Calculate how big the legend box needs to be to hold the title and any point markers (symbols or lines) and any series descriptor text.

47. `void transform_point(double & x, double & y);`

48. `void transform_point(double & x, double & y);`

< Scale & shift both X & Y to graph Cartesian coordinates.

< Scale & shift both X & Y to graph Cartesian coordinates.

49. `void transform_x(double & x);`

50. `void transform_x(double & x);`

< Scale and shift X value only.

< Scale and shift X value only.

51. `void transform_y(double & y);`

52. `void transform_y(double & y);`

< Scale and shift Y value only.

< Scale and shift Y value only.

svg_1d_plot public public data members

1. `int x_axis_position_;`

See Also:

`boost::svg::x_axis_intersect`.

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 &);

    // public data members
    plot_point_style limit_point_style_; // Default is cone pointing down.
    plot_line_style line_style_; // No line style for 1-D, only for 2-D.
    plot_point_style point_style_; // circle, square...
    std::vector< Meas > series_; // Normal 'OK to plot' data values.
    std::vector< double > series_limits_; // 'limit' values: too big, too small or NaN.
    std::string title_; // title of data series (to show on legend).
};

};
```

Description

Scan each data point sorting them into the appropriate `std::vector`s, 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:

<code>begin</code>	iterator to 1st element in container to show.
<code>end</code>	iterator to last element in container to show.
<code>title</code>	Title of series of data values.

Template Parameters:

<code>C</code>	An iterator into STL container: array, <code>std::vector<double></code> , <code>std::vector<unc></code> , <code>std::vector<Meas></code> , <code>std::set</code> , <code>std::map</code> ...
----------------	--

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). Example: `.shape(square)`, `.shape(circlet)`

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).

Function strip_e0s

boost::svg::strip_e0s — Add/uncomment in jamfile.v2.

Synopsis

```
// In header: <boost/svg_plot/svg_1d_plot.hpp>

const std::string strip_e0s(std::string s);
```

Description

Returns: length of trimmed string (perhaps unchanged).
 Returns: length of trimmed string (perhaps unchanged).

Header <[boost/svg_plot/svg_2d_plot.hpp](#)>

```
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);
    bool autoscale();
    svg_2d_plot & autoscale(bool);
    svg_2d_plot & autoscale_check_limits(bool);
    bool autoscale_check_limits();
    svg_2d_plot & autoscale_check_limits(bool);
    bool autoscale_check_limits();
    svg_2d_plot & autoscale_plusminus(double);
    double autoscale_plusminus();
    svg_2d_plot & autoscale_plusminus(double);
    double autoscale_plusminus();
    svg_2d_plot & axes_on(bool);
    bool axes_on();
    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_color(const svg_color &);
    svg_color background_border_color();
    svg_2d_plot & background_border_width(double);
    double background_border_width();
    svg_2d_plot & background_border_width(double);
    double background_border_width();
    svg_color background_color();
    svg_2d_plot & background_color(const svg_color &);
    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 & boost_license_on(bool);
    bool boost_license_on();
    void calculate_plot_window();
    svg_2d_plot & confidence(double);
    double confidence();
    svg_2d_plot & confidence(double);
    double confidence();
    svg_2d_plot & coord_precision(int);
    int coord_precision();
    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_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 & copyright_holder(const std::string);
    const std::string copyright_holder();
    svg_2d_plot & data_lines_width(double);
    double data_lines_width();
```

```

svg_2d_plot & data_lines_width(double);
double data_lines_width();
svg_2d_plot & derived();
const svg_2d_plot & derived() const;
svg_2d_plot & derived();
const svg_2d_plot & derived() const;
svg_2d_plot & description(const std::string);
const std::string & description();
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 & document_title(const std::string);
std::string document_title();
void draw_bars();
void draw_bezier_lines(const svg_2d_plot_series &);
void draw_histogram();
svg_2d_plot &
draw_line(double, double, double, const svg_color & = black);
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_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_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 &
draw_plot_line(double, double, double, double, const svg_color & = black);
void draw_plot_lines();
void draw_plot_points();
void draw_straight_lines(const svg_2d_plot_series &);
void draw_y_axis();
void draw_y_axis_label();
void draw_y_major_tick(double, path_element &, path_element &);
void draw_y_minor_tick(double, path_element &, path_element &);
svg_2d_plot & image_border_margin(double);
double image_border_margin();
svg_2d_plot & image_border_margin(double);
double image_border_margin();
svg_2d_plot & image_border_width(double);
double image_border_width();
svg_2d_plot & image_border_width(double);
double image_border_width();
unsigned int image_x_size();
svg_2d_plot & image_x_size(unsigned int);
int image_x_size();
svg_2d_plot & image_x_size(int);
unsigned int image_y_size();
svg_2d_plot & image_y_size(unsigned int);
int image_y_size();
svg_2d_plot & image_y_size(int);
bool is_in_window(double, double);

```

```

svg_2d_plot & legend_background_color(const svg_color &);
svg_color legend_background_color();
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();
svg_2d_plot & legend_border_color(const svg_color &);
svg_color legend_border_color();
const std::pair< double, double > legend_bottom_right();
const std::pair< double, double > legend_bottom_right();
bool legend_box_fill_on();
bool legend_box_fill_on();
svg_2d_plot & legend_color(const svg_color &);
svg_color legend_color();
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_family(const std::string &);
const std::string & legend_font_family();
svg_2d_plot & legend_font_size(int);
int legend_font_size();
svg_2d_plot & legend_font_weight(const std::string &);
const std::string & legend_font_weight();
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_lines(bool);
bool legend_lines();
svg_2d_plot & legend_on(bool);
bool legend_on();
svg_2d_plot & legend_on(bool);
bool legend_on();
bool legend_outside();
bool legend_outside();
svg_2d_plot & legend_place(legend_places);
legend_places legend_place();
svg_2d_plot & legend_place(legend_places);
legend_places legend_place();
svg_2d_plot & legend_text_font_size(int);
int legend_text_font_size();
svg_2d_plot & legend_text_font_weight(const std::string &);
const std::string & legend_text_font_weight();
svg_2d_plot & legend_title(const std::string);
const std::string legend_title();
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_title_font_size(int);
int legend_title_font_size();
svg_2d_plot & legend_title_font_weight(const std::string &);
const std::string & legend_title_font_weight();
svg_2d_plot & legend_top_left(double, double);
const std::pair< double, double > legend_top_left();
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 & 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");
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_attribution();
const std::string license_commercialuse();
const std::string license_commercialuse();
const std::string license_distribution();
const std::string license_distribution();
svg_2d_plot & license_on(bool);
bool license_on();
svg_2d_plot & license_on(bool);
bool license_on();
const std::string license_reproduction();
const std::string license_reproduction();
svg_2d_plot & limit_color(const svg_color &);
svg_color limit_color();
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 & 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();
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_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_color(const svg_color &);
svg_color plot_border_color();
svg_2d_plot & plot_border_width(double);
double plot_border_width();
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_on(bool);
bool plot_window_on();
svg_2d_plot & plot_window_x(double, double);
std::pair< double, double > plot_window_x();
svg_2d_plot & plot_window_x(double, double);
std::pair< double, double > plot_window_x();

```

```

double plot_window_x_left();
double plot_window_x_left();
double plot_window_x_right();
double plot_window_x_right();
svg_2d_plot & plot_window_y(double, double);
std::pair< double, double > plot_window_y();
svg_2d_plot & plot_window_y(double, double);
std::pair< double, double > plot_window_y();
double plot_window_y_bottom();
double plot_window_y_bottom();
double plot_window_y_top();
double plot_window_y_top();
void set_ids();
svg_2d_plot & size(unsigned int, unsigned int);
std::pair< double, double > size();
svg_2d_plot & size(int, int);
std::pair< double, double > size();
svg_2d_plot & three_sd_color(const svg_color &);
svg_color three_sd_color();
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(const std::string);
const std::string title();
svg_2d_plot & title_color(const svg_color &);
svg_color title_color();
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_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_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_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_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_size(int);
int title_font_size();
svg_2d_plot & title_font_stretch(const std::string &);
const std::string & title_font_stretch();
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_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_font_weight(const std::string &);
const std::string & title_font_weight();
svg_2d_plot & title_on(bool);
bool title_on();

```

```

svg_2d_plot & title_on(bool);
bool title_on();
text_style & title_style();
svg_2d_plot & title_text_length(double);
double title_text_length();
void transform_pair(std::pair< double, double > &);
svg_2d_plot & two_sd_color(const svg_color &);
svg_color two_sd_color();
svg_2d_plot & two_sd_color(const svg_color &);
svg_color two_sd_color();
void update_image();
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_color(const svg_color &);
svg_color x_addlimits_color();
svg_2d_plot & x_addlimits_on(bool);
bool x_addlimits_on();
svg_2d_plot & x_addlimits_on(bool);
bool x_addlimits_on();
double x_auto_max_value();
double x_auto_max_value();
double x_auto_min_value();
double x_auto_min_value();
double x_auto_tick_interval();
double x_auto_tick_interval();
int x_auto_ticks();
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 &);
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_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_label_color(const svg_color &);
svg_color x_axis_label_color();
svg_2d_plot & x_axis_on(bool);
bool x_axis_on();
svg_2d_plot & x_axis_on(bool);
bool x_axis_on();
const std::string x_axis_position();
const std::string x_axis_position();
svg_2d_plot & x_axis_vertical(double);
bool x_axis_vertical();
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_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_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_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_color(const svg_color &);

svg_color x_df_color();
svg_2d_plot & x_df_on(bool);
bool x_df_on();
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_color(const svg_color &);

svg_color x_id_color();
svg_2d_plot & x_id_on(bool);
bool x_id_on();
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(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_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_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_font_size(int);
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(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_units_on(bool);
bool x_label_units_on();
svg_2d_plot & x_label_width(double);
double x_label_width();
svg_2d_plot & x_label_width(double);
double x_label_width();
svg_2d_plot & x_labels_strip_e0s(bool);
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_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_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_grid_width(double);
double x_major_grid_width();
svg_2d_plot & x_major_interval(double);
double x_major_interval();
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_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(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_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_length(double);
double x_major_tick_length();
svg_2d_plot & x_major_tick_width(double);
double x_major_tick_width();
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_max(double);
double x_max();
svg_2d_plot & x_min(double);
double x_min();
svg_2d_plot & x_min(double);
double x_min();
svg_2d_plot & x_min_ticks(int);
int x_min_ticks();
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_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_on(bool);
bool x_minor_grid_on();
svg_2d_plot & x_minor_grid_width(double);
double x_minor_grid_width();
svg_2d_plot & x_minor_grid_width(double);
double x_minor_grid_width();
double x_minor_interval();
double x_minor_interval();
svg_2d_plot & x_minor_interval(double);

```

```

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_color(const svg_color &);

svg_color x_minor_tick_color();
svg_2d_plot & x_minor_tick_length(double);
double x_minor_tick_length();
double x_minor_tick_length();
svg_2d_plot & x_minor_tick_width(double);
double x_minor_tick_width();
svg_2d_plot & x_minor_tick_width(double);
double x_minor_tick_width();
double x_minor_tick_width();
svg_2d_plot & x_num_minor_ticks(unsigned int);
unsigned int x_num_minor_ticks();
svg_2d_plot & x_num_minor_ticks(int);
int x_num_minor_ticks();
svg_2d_plot & x_order_color(const svg_color &);

svg_color x_order_color();
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_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_color(const svg_color &);

svg_color x_plusminus_color();
svg_2d_plot & x_plusminus_on(bool);
bool x_plusminus_on();
svg_2d_plot & x_plusminus_on(bool);
bool x_plusminus_on();
const std::string x_prefix();
const std::string x_prefix();
svg_2d_plot & x_range(double, double);
std::pair< double, double > x_range();
svg_2d_plot & x_range(double, double);
std::pair< double, double > x_range();
const std::string x_separator();
const std::string x_separator();
svg_2d_plot & x_size(unsigned int);
unsigned int x_size();
svg_2d_plot & x_size(int);
int x_size();
svg_2d_plot & x_steps(int);
int x_steps();
svg_2d_plot & x_steps(int);
int x_steps();
const std::string x_suffix();
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_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_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_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_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_font_size(int);
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_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_ticks_values_precision(int);
int x_ticks_values_precision();
svg_2d_plot & x_tight(double);
double x_tight();
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_font_size(int);
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_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_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_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_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_font_size(int);
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_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_on(bool);
bool x_values_on();
svg_2d_plot & x_values_precision(int);
int x_values_precision();
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_values_rotation(rotate_style);
int x_values_rotation();

```

```

svg_2d_plot & x_with_zero(bool);
bool x_with_zero();
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 &);
const svg_color y_axis_label_color();
svg_2d_plot & y_axis_on(bool);
bool y_axis_on();
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 &);
const 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(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 &);
const svg_color y_label_color();
svg_2d_plot & y_label_color(const svg_color &);
const 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(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();
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);
int y_size();
svg_2d_plot & y_size(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 &);
const 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();

// protected member functions
void adjust_limits(double &, double &);

void adjust_limits(double &, double &);

void clear_all();
void clear_all();
void clear_background();
void clear_background();
void clear_grids();
void clear_grids();
void clear_legend();
void clear_legend();
void clear_plot_background();
void clear_plot_background();
void clear_points();
void clear_points();
void clear_title();
void clear_title();
void clear_x_axis();
void clear_x_axis();
void clear_y_axis();
void clear_y_axis();
void draw_legend();
void draw_legend();
void draw_plot_point(double, double, g_element &, const plot_point_style &);

void draw_plot_point(double, double, g_element &, plot_point_style &,
                     unc< false >, unc< false >);

void draw_plot_point(double, double, g_element &, const plot_point_style &);

void draw_plot_point(double, double, g_element &, plot_point_style &,
                     unc< false >, unc< false >);

void draw_plot_point_value(double, double, g_element &, value_style &,
                           plot_point_style &, Meas);

void draw_plot_point_value(double, double, g_element &, value_style &,
                           plot_point_style &, Meas);

void draw_plot_point_values(double, double, g_element &, g_element &,
                           const value_style &, const value_style &, Meas,
                           Meas);

void draw_plot_point_values(double, double, g_element &, g_element &,
                           Meas);

```

```

        const value_style &, const value_style &, Meas,
        unc< false >);
void draw_plot_point_values(double, double, g_element &, g_element &,
                           const value_style &, const value_style &, Meas,
                           Meas);
void draw_plot_point_values(double, double, g_element &, g_element &,
                           const value_style &, const value_style &, Meas,
                           unc< false >);
void draw_title();
void draw_title();
void draw_x_axis();
void draw_x_axis();
void draw_x_axis_label();
void draw_x_axis_label();
void draw_x_major_tick(double, path_element &, path_element &);
void draw_x_major_tick(double, path_element &, path_element &);
void draw_x_minor_tick(double, path_element &, path_element &);
void draw_x_minor_tick(double, path_element &, path_element &);
void place_legend_box();
void place_legend_box();
void size_legend_box();
void size_legend_box();
void transform_point(double &, double &);
void transform_point(double &, double &);
void transform_x(double &);
void transform_x(double &);
void transform_y(double &);
void transform_y(double &);

// public data members
text_style a_style_; // Defaults for text_style (contains font size & type etc).
double alpha_; // = 0.05; // oss.iword(confidenceIndex) / 1.e6; // Pick up alpha.
bool autoscale_check_limits_; // true if to check autoscale values for infinity, NaN, max, min.
double autoscale_plusminus_; // For uncertain values, allow for text_plusminus ellipses showing 67%, 95% □
and 99% confidence limits.

```

For example, if a max value is 1.2 +or- 0.02, then 1.4 will be used for autoscaling the maximum.

Similarly, if a min value is 1.2 +or- 0.02, then 1.0 will be used for autoscaling the minimum.

```

double biggest_point_font_size_; // Biggest point marker symbol - determines vertical spacing.
double epsilon_;
double horizontal_line_spacing_;
double horizontal_marker_spacing_;
double horizontal_spacing_;
svg image_; // Stored so as to avoid rewriting style information constantly.
box_style image_border_; // rectangular border box style of all image width, color...
bool is_a_data_series_line_; // true if any data series have point markers to show in legend (default false).
bool is_a_data_series_text_; // true is any series should show text describing the data series (default false). For example: my_plot.plot(my_data_0, "my_data_0_text");
bool is_a_point_marker_;
bool is_legend_title_; // true if legend_title_.text() != "" (for example: .legend_title("My Legend")); (default false).
bool isNoisyDigit_;
double legend_bottom_; // SVG Coordinates of bottom of legend box.
box_style legend_box_; // rectangular box style of legend width, color...
double legend_height_; // Height of legend box (pixels).
double legend_left_; // Left of legend box.
bool legend_lines_; // true if wish to add a colored line for each data series in legend box.
size_t legend_longest_; // longest (both header & data) string in legend box.
bool legend_on_; // true if to provide a legend box (default false unless a legend header title is set so that legend_title_.text() != "").
legend_places legend_place_; // Place for any legend box.
double legend_right_; // SVG Coordinates of right of legend box,.
text_element legend_text_; // Legend box series data descriptor text(if any).
double legend_text_font_size_; // Font size of legend text.
text_style legend_text_style_; // Style for legend text.
text_element legend_title_; // Legend box header or title (if any).
double legend_title_font_size_; // Font size of legend header/title.
text_style legend_title_style_; // Style for legend title.
double legend_top_; // Top of legend box.
double legend_widest_line_; // Width of longest of legend header/title and widest data series pointer+line+text.
double legend_width_; // Width of legend box (pixels).
const double margin; // Plot window margin to allow to rounding etc when checking if a point is inside window with is_in_window function.
std::vector< text_element > notes_; // Store of text for annotation. (Not used yet?)
bool outside_legend_on_; // true if legend box should be outside the plot window (default true).
double plot_bottom_; // SVG Y coordinate of bottom side of plot window.
double plot_left_; // SVG X coordinate (pixels) of left side of plot window.
double plot_right_; // SVG X coordinate of right side of plot window.
double plot_top_; // SVG Y coordinate of top side of plot window.
box_style plot_window_border_; // rectangular border box style of plot window width, color...
std::string plot_window_clip_;
bool plot_window_on_; // true if to use a separate plot window (not the whole image).
text_style point_symbols_style_; // Style used for symbol marking a data point.
std::vector< svg_2d_plot_series > serieses_; // Store of several series of data points for transformation.
double text_margin_; // Marginal space factor around text items like title. text_margin_*font_size = vertical distance in svg units, text_margin_*font_size = horizontal distance in svg units.
double text_plusminus_;
text_element title_info_; // Plot title text etc.
bool title_on_; // true if to display a title for the whole plot (default true).
text_style title_style_; // Style for plot title.
int uncSigDigits_;
text_style value_style_; // Style used for data point value label.
double vertical_line_spacing_;
double vertical_marker_spacing_;
double vertical_spacing_;
double x_auto_max_value_; // Values calculated by scale_axis (used only if x_autoscale == true).
double x_auto_min_value_; // Values calculated by scale_axis (used only if x_autoscale == true).
double x_auto_tick_interval_; // X tick major interval.
int x_auto_ticks_; // Number of X ticks.

```

```

bool x_autoscale_; // true if to use any X-axis autoscale values.
axis_line_style x_axis_; // Style of X-axis.
text_style x_axis_label_style_; // Style for tick labels on X-axis.
int x_axis_position_; // Intersection with Y-axis, or not.
bool x_include_zero_; // true if autoscaled, to include zero.
text_element x_label_info_; // X-axis label text, for example: "length".
int x_min_ticks_; // If autoscaled, set a minimum number of X ticks.
bool x_plusminus_on_; // http://en.wikipedia.org/wiki/Plus-minus_sign Unicode &#0xB1; HTML &plusmn;
double x_scale_; // scale factor used by transform() to go from Cartesian to SVG coordinates.
double x_shift_; // shift factor used by transform() to go from Cartesian to SVG coordinates.
int x_steps_; // true if autoscale to set any prescaling to multiple of decimal 1, 2, 5, 10 etc.
ticks_labels_style x_ticks_; // Style of X-axis tick marks and labels.
bool x_ticks_on_; // true if X-axis to have ticks.
double x_tight_; // Tolerance used by autoscale to avoid extra ticks.
text_element x_units_info_; // X-axis units, for example: "mm".
text_element x_value_label_info_; // X-axis tick value text, for example: "1.2" or "1.2e+001".
text_style x_value_label_style_; // Style for data point value labels on X-axis.
bool x_values_on_; // true if values of X data are shown (as 1.23).
value_style x_values_style_; // Data point X value marking, font, size etc.
bool xy_values_on_; // true if values of X & Y pairs are shown (as 1.23, 3.43).
double y_auto_max_value_; // Values calculated by scale_axis (used only if y_autoscale == true).
double y_auto_min_value_; // Values calculated by scale_axis (used only if y_autoscale == true).
double y_auto_tick_interval_; // tick major interval (calculated by Y autoscale).
int y_auto_ticks_; // Number of ticks (calculated by Y autoscale).
bool y_autoscale_; // true if to use any y_axis autoscale values.
axis_line_style y_axis_; // Style of Y-axis.
text_style y_axis_label_style_; // Style for tick labels on Y-axis.
int y_axis_position_; // Intersection with X-axis, or not.
bool y_include_zero_; // true if autoscale to include zero.
text_element y_label_info_; // Y-axis label text, for example: "volume".
int y_min_ticks_; // If autoscaled, set a minimum number of Y ticks.
double y_scale_; // scale factor used by transform() to go from Cartesian to SVG coordinates.
double y_shift_; // shift factor used by transform() to go from Cartesian to SVG coordinates.
int y_steps_; // If autoscaled, set any prescaling to decimal 1, 2, 5, 10 etc.
ticks_labels_style y_ticks_; // Style of Y-axis tick marks and labels.
bool y_ticks_on_; // true if Y-axis to have ticks.
double y_tight_; // Tolerance used by autoscale to avoid extra ticks.
text_element y_units_info_; // Y-axis units, for example: "min". (2-D only).
std::ios_base::fmtflags y_value_ioflags_; // std::iosflags used for Y value labels (default ios::dec).
text_element y_value_label_info_; // Y-axis tick value text, for example: "1.2" or "1.2e+001".
rotate_style y_value_label_rotation_; // Direction point Y value labels written (default horizontal).
text_style y_value_label_style_; // Style for data point value labels on Y-axis.
int y_value_precision_; // std::ios precision used for Y value labels (default 3).
bool y_values_on_; // true if values of Y data are shown (as 3.45).
value_style y_values_style_; // Data point Y value marking, font, size etc.
};


```

Description

See Also:

[svg_2d_plot_series](#) that allows data values to be added.

See Also:

[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-D 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. `bool autoscale();`

Returns: true if to use autoscale values autoscaling for X-axis.

Returns: true if to use autoscale values for X-axis. `autoscale()` is same as `x_autoscale`.

Returns: true if to use autoscale values for X-axis. `autoscale()` is same as `x_autoscale`.

4. `svg_2d_plot & autoscale(bool b);`

Set true if to use autoscale values for X-axis.

Set whether to use X autoscaled values. Same as `x_autoscale`, and used by boxplot too.

Set whether to use X autoscaled values. Same as `x_autoscale`, and used by boxplot too.

5. `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'.

6. `bool autoscale_check_limits();`

Returns: true if to check that values used for autoscaling are within limits.

7. `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'.

Set to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed.

Set to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed.

8. `bool autoscale_check_limits();`

Returns: true if to check that values used for autoscaling are within limits.

Returns: to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed.

Returns: to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed.

9. `svg_2d_plot & autoscale_plusminus(double);`

Set how many std_dev or standard deviations to allow for ellipses when autoscaling.

10. `double autoscale_plusminus();`

Returns: How many std_dev or standard deviations allowed for ellipses when autoscaling.

11. `svg_2d_plot & autoscale_plusminus(double);`

Set how many std_dev or standard deviations to allow for ellipses when autoscaling.

Set how many std_dev or standard deviation to allow for ellipse when autoscaling. Default is 3 for 99% confidence.

Set how many std_dev or standard deviation to allow for ellipse when autoscaling. Default is 3 for 99% confidence.

12. `double autoscale_plusminus();`

Returns: How many std_dev or standard deviations allowed for ellipses when autoscaling.

Returns: how many std_dev or standard deviation to allow for ellipse when autoscaling. Default is 3 for 99% confidence.

Returns: how many std_dev or standard deviation to allow for ellipse when autoscaling. Default is 3 for 99% confidence.

13. `svg_2d_plot & axes_on(bool is);`

Set true if to draw **both** x and y axes (note plural axes).

14. `bool axes_on();`

Returns: true if to draw **both** x and y axis on.

15. `svg_2d_plot & axes_on(bool is);`

Set true if to draw **both** x and y axes (note plural axes).

If set true, draw **both** x and y axes (note plural axes).

If set true, draw **both** x and y axes (note plural axes).

16. `bool axes_on();`

Returns: true if to draw **both** x and y axis on.

Returns: true if **both** x and y axis on.

Returns: true if **both** x and y axis on.

17. `svg_2d_plot & background_border_color(const svg_color & col);`

Set plot background border color.

18. `svg_color background_border_color();`

Returns: plot background border color.

19. `svg_2d_plot & background_border_color(const svg_color & col);`

Set plot background border color.

Set plot background border color.

`background_border_color`, for example: `svg_2d_plot my_plot(my_data, "My Data").background_border_color(red).background_color(azure);`

Set plot background border color.

`background_border_color`, for example: `svg_2d_plot my_plot(my_data, "My Data").background_border_color(red).background_color(azure);`

20. `svg_color background_border_color();`

Returns: plot background border color.

Returns: plot background border color.

Returns: plot background border color.

21. `svg_2d_plot & background_border_width(double w);`

Set plot background border width.

22. `double background_border_width();`

Returns: Plot background border width.

23. `svg_2d_plot & background_border_width(double w);`

Set plot background border width.

Set plot background border width.

Set plot background border width.

24. `double background_border_width();`

- Returns: Plot background border width.
 Returns: plot background border width.
 Returns: plot background border width.
25. `svg_color background_color();`
- Returns: Plot background color.
26. `svg_2d_plot & background_color(const svg_color & col);`
- Set plot background color.
27. `svg_color background_color();`
- Returns: Plot background color.
 Returns: plot background color.
 Returns: plot background color.
28. `svg_2d_plot & background_color(const svg_color & col);`
- Set plot background color.
- Set plot background color.
- Set plot background color.
29. `svg_2d_plot & boost_license_on(bool l);`
- Set true if the Boost license conditions should be included in the SVG document.
30. `bool boost_license_on();`
- Returns: true if the Boost license conditions should be included in the SVG document.
31. `svg_2d_plot & boost_license_on(bool l);`
- Set true if the Boost license conditions should be included in the SVG document.
- Set if the Boost license conditions should be included in the SVG document.
- Set if the Boost license conditions should be included in the SVG document.
32. `bool boost_license_on();`
- Returns: true if the Boost license conditions should be included in the SVG document.
 Returns: if the Boost license conditions should be included in the SVG document. To set see `axis_plot_frame::boost_license_on(bool)`.
 Returns: if the Boost license conditions should be included in the SVG document. To set see `axis_plot_frame::boost_license_on(bool)`.
33. `void calculate_plot_window();`

The plot window is used to set a clip path: this ensures that data points and lines (and anything else) outside this window are NOT drawn.

All calculation use svg units, pixels by default.

34. `svg_2d_plot & confidence(double);`

Set confidence alpha for display of confidence intervals (default 0.05 for 95%).

35. `double confidence();`

Returns: Confidence alpha for display of confidence intervals (default 0.05 for 95%).

36. `svg_2d_plot & confidence(double);`

Set confidence alpha for display of confidence intervals (default 0.05 for 95%).

Set alpha for displaying confidence intervals. Default is 0.05 for 95% confidence.

Set alpha for displaying confidence intervals. Default is 0.05 for 95% confidence.

37. `double confidence();`

Returns: Confidence alpha for display of confidence intervals (default 0.05 for 95%).

Returns: alpha for displaying confidence intervals. Default is 0.05.

Returns: alpha for displaying confidence intervals. Default is 0.05.

38. `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.

39. `int coord_precision();`

Returns: precision of SVG coordinates in decimal digits.

40. `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.

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.

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.

41. `int coord_precision();`

Returns: precision of SVG coordinates in decimal digits.

Returns: precision of SVG coordinates in decimal digits.
Returns: precision of SVG coordinates in decimal digits.

42 `svg_2d_plot & copyright_date(const std::string d);`

Writes copyright date to the SVG document. and as metadata:<meta name="date" content="2007" />

43 `const std::string copyright_date();`

Returns: SVG document copyright_date.

44 `svg_2d_plot & copyright_date(const std::string d);`

Writes copyright date to the SVG document. and as metadata:<meta name="date" content="2007" />

Writes copyright date to the document. and as metadata <meta name="date" content="2007" />

Writes copyright date to the document. and as metadata <meta name="date" content="2007" />

45 `const std::string copyright_date();`

Returns: SVG document copyright_date.

Returns: copyright_date.

Returns: copyright_date.

46 `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" />

47 `const std::string copyright_holder();`

Returns: SVG document copyright holder.

48 `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" />

Writes copyright_holder to the document (as<!-- SVG Plot Copyright Paul A. Bristow 2007 -->) and as metadata: <meta name="copyright" content="Paul A. Bristow" /meta>

Writes copyright_holder to the document (for header as<!-- SVG Plot Copyright Paul A. Bristow 2007 -->) and as metadata: <meta name="copyright" content="Paul A. Bristow" /meta>

49 `const std::string copyright_holder();`

Returns: SVG document copyright holder.

Returns: copyright holder.

Returns: copyright holder.

50 `svg_2d_plot & data_lines_width(double width);`

Set the width of lines joining data points.

51. `double data_lines_width();`

Returns: the width of lines joining data points.

52. `svg_2d_plot & data_lines_width(double width);`

Set the width of lines joining data points.

Set the width of lines joining data points.

Set the width of lines joining data points.

53. `double data_lines_width();`

Returns: the width of lines joining data points.

Returns: the width of lines joining data points.

Returns: the width of lines joining data points.

54. `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. Sadly this has the most unwelcome effect of terminally confusing the Visual Studio Intellisense, and sometimes the debugger as well :-(

55. `const svg_2d_plot & derived() const;`

const version of derived()

56. `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. Sadly this has the most unwelcome effect of terminally confusing the Visual Studio Intellisense, and sometimes the debugger as well :-(

57. `const svg_2d_plot & derived() const;`

const version of derived()

58. `svg_2d_plot & description(const std::string d);`

Writes description to the document for header as.

<desc> My Description </desc>.

59. `const std::string & description();`

Returns: Description of the document for header as<desc> My description </desc>.

60. `svg_2d_plot & description(const std::string d);`

Writes description to the document for header as.

<desc> My Description </desc>.

Writes description to the SVG document for header, for example: <desc> My Data </desc>

Writes description to the document for header, for example: <desc> My Data </desc>

61. `const std::string & description();`

Returns: Description of the document for header as<desc> My description </desc>.

Returns: Description of the document for title as<desc> ... </desc>

Returns: description of the document for header as<desc> ... </desc>

62. `svg_2d_plot & document_title(const std::string d);`

Set document title to the document for header as.

<title> My Title </title>.

63. `std::string document_title();`

Returns: Document title to the document for header as<title> My Title </title>.

64. `svg_2d_plot & document_title(const std::string d);`

Set document title to the document for header as.

<title> My Title </title>.

Write document title to the SVG document for title as<title> My Title </title>

Write document title to the SVG document for header as<title> My Title </title>

65. `std::string document_title();`

Returns: Document title to the document for header as<title> My Title </title>.

Returns: Get document title to the document for title as<title> My Title </title>

Returns: Get document title to the document for header as<title> My Title </title>

66. `void draw_bars();`

Draw normal bar chart for 'good' non-limit points.

67. `void draw_bezier_lines(const svg_2d_plot_series & series);`

Add Bezier curve line between data points. Warning: At present it is assumed that all data points lie within the plot window. If this is not true, then strange and unpredictable curves will be produced!

68. `void draw_histogram();`

Draw a histogram with variable width but contiguous bins. Histograms differ from bar charts in the the *area* denotes the value, whereas the bar *height* denotes the value for a bar chart. bin widths are provided from the X-axis data series values. The 1st data X-value provides the start of the 1st bin, the 2nd data X-value provides the end of the 1st bin, and the 1st Y-value the area of the

1st bin, and the start of the second bin, and so on, until the width of last bin is calculated from the last data point in series, that must have a zero area. ? NaN Bins can be the same (most common) or different widths. Intervals must not overlap and bins must be adjacent. <http://en.wikipedia.org/wiki/Histogram>

Attempts to allow a row or horizontal were abandoned because of complications with the use of map which orders the x values providing the bins. Using the y values for the bins implies changing the Y axes labeling and scaling too.

69.

```
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.

70.

```
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.

Annotate plot with a line from SVG Coordinates X1, Y1 to X2, Y2.

Default color black.

Annotate plot with a line from SVG Coordinates X1, Y1 to X2, Y2.

Default color black.

71.

```
svg_2d_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.

72.

```
svg_2d_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.

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.

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.

73. `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.

74. `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.

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.

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.

75. `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.

76. `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.

Annotate plot with a line from user's Cartesian Coordinates X1, Y1 to X2, Y2.

For example, -10, -10, +10, +10, Default color 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.

77. `void draw_plot_lines();`

Draw line through data series, Bezier curved or straight, or none.

78. `void draw_plot_points();`

Draw normal 'good' non-limit points, and then any 'at limits' points.

Draw the abnormal 'at_limit' points (if any).

79. `void draw_straight_lines(const svg_2d_plot_series & series);`

Add line between series of data points (straight rather than a Bezier curve). Area fill with color if specified.

80. `void draw_y_axis();`

Draw the Y-axis line, grids and ticks with labels.

81. `void draw_y_axis_label();`

Draw a vertical Y-axis label, and optional y units.

82. `void draw_y_major_tick(double value, path_element & tick_path, path_element & grid_path);`

Draw a Y-axis major tick, tick value labels & grids.

83. `void draw_y_minor_tick(double value, path_element & tick_path, path_element & grid_path);`

Draw a Y-axis minor tick and optional grid. (minor ticks do not have value labels).

84. `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.

85. `double image_border_margin();`

Returns: the margin around the plot window border (svg units, default pixels).

86. `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.

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.

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.

87. `double image_border_margin();`

Returns: the margin around the plot window border (svg units, default pixels).

Returns: the margin around the plot window border (svg units, default pixels).

Returns: the margin around the plot window border (svg units, default pixels).

88. `svg_2d_plot & image_border_width(double w);`

Set the svg image border width (svg units, default pixels).

89. `double image_border_width();`

Returns: the svg image border width (svg units, default pixels).

90. `svg_2d_plot & image_border_width(double w);`

Set the svg image border width (svg units, default pixels).

Set the svg image border width (svg units, default pixels).

Set the svg image border width (svg units, default pixels).

91. `double image_border_width();`

Returns: the svg image border width (svg units, default pixels).

Returns: the svg image border width (svg units, default pixels).

Returns: the svg image border width (svg units, default pixels).

92. `unsigned int image_x_size();`

Obselete - deprecated use x_size()

93. `svg_2d_plot & image_x_size(unsigned int i);`

Obselete - deprecated - use x_size().

Set SVG image X-axis size (SVG units, default pixels).

94. `int image_x_size();`

Obselete - deprecated use x_size()

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

95. `svg_2d_plot & image_x_size(int i);`

Obselete - deprecated - use x_size().

Set SVG image X-axis size (SVG units, default pixels).

96. `unsigned int image_y_size();`

Obselete - deprecated - use y_size()

97. `svg_2d_plot & image_y_size(unsigned int i);`

Obselete - deprecated - use `y_size()`

Set SVG image Y-axis size (SVG units, default pixels).

98. `int image_y_size();`

Obselete - deprecated - use `y_size()`

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

99. `svg_2d_plot & image_y_size(int i);`

Obselete - deprecated - use `y_size()`

Set SVG image Y-axis size (SVG units, default pixels).

100. `bool is_in_window(double x, double y);`

Check if a point is within the plot window (or not too far outside).

101. `svg_2d_plot & legend_background_color(const svg_color & col);`

Set the background fill color of the legend box.

102. `svg_color legend_background_color();`

Returns: the background fill color of the legend box.

103. `svg_2d_plot & legend_background_color(const svg_color & col);`

Set the background fill color of the legend box.

Set the background fill color of the legend box.//

Set the background fill color of the legend box.

104. `svg_color legend_background_color();`

Returns: the background fill color of the legend box.

Returns: the background fill color of the legend box.

Returns: the background fill color of the legend box.

105. `svg_2d_plot & legend_border_color(const svg_color & col);`

Set the border stroke color of the legend box.

106. `svg_color legend_border_color();`

Returns: the border stroke color of the legend box.

107. `svg_2d_plot & legend_border_color(const svg_color & col);`

Set the border stroke color of the legend box.

Set the border stroke color of the legend box.

Set the border stroke color of the legend box.

18 `svg_color legend_border_color();`

Returns: the border stroke color of the legend box.

Returns: the border stroke color of the legend box.

Returns: the border stroke color of the legend box.

19 `const std::pair< double, double > legend_bottom_right();`

Returns: SVG coordinate (default pixels) of bottom right of legend box.

10 `const std::pair< double, double > legend_bottom_right();`

Returns: SVG coordinate (default pixels) of bottom right of legend box.

Returns: svg coordinate (default pixels) of Bottom right of legend box.

Returns: svg coordinate (default pixels) of Bottom right of legend box.

11 `bool legend_box_fill_on();`

Returns: true if legend box has a background fill color.

12 `bool legend_box_fill_on();`

Returns: true if legend box has a background fill color.

Returns: true if legend box has a background fill color.

Returns: true if legend box has a background fill color.

13 `svg_2d_plot & legend_color(const svg_color & col);`

Set the color of the title of the legend.

14 `svg_color legend_color();`

Returns: the color of the title of the legend.

15 `svg_2d_plot & legend_color(const svg_color & col);`

Set the color of the title of the legend.

Set the color of the title of the legend.

Set the color of the title of the legend.

16 `svg_color legend_color();`

Returns: the color of the title of the legend.

Returns: the color of the title of the legend.

Returns: the color of the title of the legend.

17 `svg_2d_plot & legend_font_family(const std::string & family);`

Set the font family for the legend title.

18 `const std::string & legend_font_family();`

Returns: the font family for the legend title.

19 `svg_2d_plot & legend_font_family(const std::string & family);`

Set the font family for the legend text.

Set the font family for the legend title.

Set the font family for the legend title.

20 `const std::string & legend_font_family();`

Returns: the font family for the legend title.

Returns: the font family for the legend title.

Returns: the font family for the legend title.

21 `svg_2d_plot & legend_font_size(int size);`

Set Font size for **both** the legend title and legend text (svg units, default pixels).

Returns: Font size for the legend title and legend text.

22 `int legend_font_size();`



Note

If function .legend_font_size(size) has been used to set both title and text font sizes, then these will be the same (unless altered by a call of .legend_text_font_size(size)).

Returns: Font size for the legend title (svg units, default pixels).

Returns: Font size for the legend title (svg units, default pixels).

23 `svg_2d_plot & legend_font_weight(const std::string & weight);`

Set the font weight for the legend title.

24 `const std::string & legend_font_weight();`

Returns: Font weight for the legend title.

25 `svg_2d_plot & legend_font_weight(const std::string & weight);`

Set the font weight for the legend title.

Set the font weight for the legend title.

Set the font weight for the legend title.

16 `const std::string & legend_font_weight();`

Returns: Font weight for the legend text.
 Returns: the font weight for the legend title.
 Returns: the font weight for the legend title.

17 `svg_2d_plot & legend_header_font_size(int size);`

Set legend header font size (svg units, default pixels).

Set legend header font size (svg units, default pixels).

18 `int legend_header_font_size();`

Returns: legend header font size (svg units, default pixels).
 Returns: legend header font size (svg units, default pixels).

19 `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.

20 `bool legend_lines();`

Returns: true if legend should include samples of the lines joining data points.

21 `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.

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.

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.

22 `bool legend_lines();`

Returns: true if legend should include samples of the lines joining data points.
 Returns: true if legend should include samples of the lines joining data points.
 Returns: true if legend should include samples of the lines joining data points.

23 `svg_2d_plot & legend_on(bool cmd);`

Set true if a legend is wanted.

14 `bool legend_on();`

Returns: true if a legend is wanted.

15 `svg_2d_plot & legend_on(bool cmd);`

Set true if a legend is wanted.

Set true if a legend is wanted.

Set true if a legend is wanted.

16 `bool legend_on();`

Returns: true if a legend is wanted.

Returns: true if a legend is wanted.

Returns: true if a legend is wanted.

17 `bool legend_outside();`

Returns: if the legend should be outside the plot area.

18 `bool legend_outside();`

Returns: if the legend should be outside the plot area.

Returns: if the legend should be outside the plot area.

Returns: if the legend should be outside the plot area.

19 `svg_2d_plot & legend_place(legend_places l);`

Set the position of the legend..

See Also:

`boost::svg::legend_places`

20 `legend_places legend_place();`

See Also:

`boost::svg::legend_places`

Returns: the position of the legend,

21 `svg_2d_plot & legend_place(legend_places l);`

Set the position of the legend..

See Also:

`boost::svg::legend_places`

Set the position of the legend,

See Also:

`boost::svg::legend_places`

Set the position of the legend,

See Also:

`boost::svg::legend_places`

142 `legend_places legend_place();`

See Also:

`boost::svg::legend_places`

See Also:

`boost::svg::legend_places`

See Also:

`boost::svg::legend_places`

Returns: the position of the legend,

Returns: the position of the legend,

Returns: the position of the legend,

143 `svg_2d_plot & legend_text_font_size(int size);`

Set the font size for the legend title (svg units, default pixels).

Returns: Font size for the legend text.

144 `int legend_text_font_size();`

Returns: Font size for the legend text (svg units, default pixels).

Returns: the font size for the legend title (svg units, default pixels).

145 `svg_2d_plot & legend_text_font_weight(const std::string & weight);`

Set the font weight for the legend text. Example: `my_plot.legend_text_font_size(20);`

146 `const std::string & legend_text_font_weight();`

Returns: the font weight for the legend text.

147 `svg_2d_plot & legend_title(const std::string title);`

Set the title for the legend.

148 `const std::string legend_title();`

Returns: Title for the legend.

149 `svg_2d_plot & legend_title(const std::string title);`

Set the title for the legend.

Set the title for the legend.

Set the title for the legend.

10 `const std::string legend_title();`

Returns: Title for the legend.

Returns: the title for the legend.

Returns: the title for the legend.

11 `svg_2d_plot & legend_title_font_size(unsigned int size);`

Set the font size for the legend title (svg units, default pixels).

Returns: Font family for the legend title.

12 `unsigned int legend_title_font_size();`

Returns: Font size for the legend title (svg units, default pixels).

13 `svg_2d_plot & legend_title_font_size(int size);`

Set the font size for the legend title (svg units, default pixels).

Returns: Font size for the legend title.

14 `int legend_title_font_size();`

Returns: Font size for the legend title (svg units, default pixels).

Returns: the font size for the legend title (svg units, default pixels).

Returns: the font size for the legend title (svg units, default pixels).

15 `svg_2d_plot & legend_title_font_weight(const std::string & weight);`

Set the font weight for the legend title.

Set the font weight for the legend title. Example: `my_plot.legend_title_font_size(10);`

16 `const std::string & legend_title_font_weight();`

Returns: Font weight for the legend title.

Returns: the font weight for the legend title.

17 `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).

18 `const std::pair< double, double > legend_top_left();`

Returns: SVG coordinate (default pixels) of top left of legend box.

19 `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).

Set position of top left of legend box (svg coordinates, default pixels). Bottom right is controlled by contents, so the user cannot set it.

Set position of top left of legend box (svg coordinates, default pixels). Bottom right is controlled by contents, so the user cannot set it.

10 `const std::pair< double, double > legend_top_left();`

Returns: SVG coordinate (default pixels) of top left of legend box.

Returns: svg coordinate (default pixels) of top left of legend box.

Returns: svg coordinate (default pixels) of top left of legend box.

11 `svg_2d_plot & legend_width(double width);`

Set the width for the legend box.

12 `double legend_width();`

Returns: Width for the legend box.

13 `svg_2d_plot & legend_width(double width);`

Set the width for the legend box.

Set the width for the legend.

Set the width for the legend.

14 `double legend_width();`

Returns: Width for the legend box.

Returns: the width for the legend.

Returns: the width for the legend.

15 `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.

16 `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.

Set license conditions for reproduction, attribution, commercial use, and derivative works, usually "permits", "requires", or "prohibits", and set license_on == true.

Set license conditions for reproduction, attribution, commercial use, and derivative works, usually "permits", "requires", or "prohibits", and set license_on == true.

17 `const std::string license_attribution();`

Returns: SVG document attribution license conditions, usually "permits", "requires", or "prohibits".

18 `const std::string license_attribution();`

Returns: SVG document attribution license conditions, usually "permits", "requires", or "prohibits".

Returns: attribution license conditions, usually "permits", "requires", or "prohibits".

Returns: attribution license conditions, usually "permits", "requires", or "prohibits".

19 `const std::string license_commercialuse();`

Returns: SVG document commercial use license conditions, usually "permits", "requires", or "prohibits".

20 `const std::string license_commercialuse();`

Returns: SVG document commercial use license conditions, usually "permits", "requires", or "prohibits".

Returns: commercial use license conditions, usually "permits", "requires", or "prohibits".

Returns: commercial use license conditions, usually "permits", "requires", or "prohibits".

21 `const std::string license_distribution();`

Returns: SVG document distribution license conditions, usually "permits", "requires", or "prohibits".

22 `const std::string license_distribution();`

Returns: SVG document distribution license conditions, usually "permits", "requires", or "prohibits".

Returns: distribution license conditions, usually "permits", "requires", or "prohibits".

Returns: distribution license conditions, usually "permits", "requires", or "prohibits".

23 `svg_2d_plot & license_on(bool l);`

Set if license conditions should be included in the SVG document.

24 `bool license_on();`

Returns: true if license conditions should be included in the SVG document.

25 `svg_2d_plot & license_on(bool l);`

Set if license conditions should be included in the SVG document.

Set if license conditions should be included in the SVG document.

See Also:

`axis_plot_frame::license`

Set if license conditions should be included in the SVG document.

See Also:

axis_plot_frame::license

16 `bool license_on();`

See Also:

axis_plot_frame::license

See Also:

axis_plot_frame::license

Returns: true if license conditions should be included in the SVG document.

Returns: true if license conditions should be included in the SVG document.

Returns: true if license conditions should be included in the SVG document.

17 `const std::string license_reproduction();`

Returns: SVG document reproduction license conditions, usually "permits", "requires", or "prohibits".

18 `const std::string license_reproduction();`

Returns: SVG document reproduction license conditions, usually "permits", "requires", or "prohibits".

Returns: reproduction license conditions, usually "permits", "requires", or "prohibits".

Returns: reproduction license conditions, usually "permits", "requires", or "prohibits".

19 `svg_2d_plot & limit_color(const svg_color &);`

Set the color for 'at limit' point stroke color.

20 `svg_color limit_color();`

Returns: the color for the 'at limit' point stroke color.

21 `svg_2d_plot & limit_color(const svg_color &);`

Set the color for 'at limit' point stroke color.

Set the color for 'at limit' point stroke color.

Set the color for 'at limit' point stroke color.

22 `svg_color limit_color();`

Returns: the color for the 'at limit' point stroke color.

Returns: the color for the 'at limit' point stroke color.

Returns: the color for the 'at limit' point stroke color.

23 `svg_2d_plot & limit_fill_color(const svg_color &);`

Set the color for 'at limit' point fill color.

24 `svg_color limit_fill_color();`

Returns: the color for the 'at limit' point fill color.

15 `svg_2d_plot & limit_fill_color(const svg_color &);`

Set the color for 'at limit' point fill color.

Set the color for 'at limit' point fill color.

Set the color for 'at limit' point fill color.

16 `svg_color limit_fill_color();`

Returns: the color for the 'at limit' point fill color.

Returns: the color for the 'at limit' point fill color.

Returns: the color for the 'at limit' point fill color.

17 `svg_2d_plot & one_sd_color(const svg_color &);`

Set the color for the one standard deviation (~67% confidence) ellipse fill.

18 `svg_color one_sd_color();`

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

19 `svg_2d_plot & one_sd_color(const svg_color &);`

Set the color for the one standard deviation (~67% confidence) ellipse fill.

Set the color for the one standard deviation (~67% confidence) ellipse fill.

Set the color for the one standard deviation (~67% confidence) ellipse fill.

20 `svg_color one_sd_color();`

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

21 `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.

12 `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.

13 `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.

14 `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.

15 `svg_2d_plot & plot_background_color(const svg_color & col);`

Set the fill color of the plot window background.

16 `svg_color plot_background_color();`

Returns: the fill color of the plot window background.

17 `svg_2d_plot & plot_background_color(const svg_color & col);`

Set the fill color of the plot window background.

Set the fill color of the plot window background.

Set the fill color of the plot window background.

18 `svg_color plot_background_color();`

Returns: the fill color of the plot window background.
 Returns: the fill color of the plot window background.
 Returns: the fill color of the plot window background.

19 `svg_2d_plot & plot_border_color(const svg_color & col);`

Set the color for the plot window background.

20 `svg_color plot_border_color();`

Returns: the color for the plot window background.

21 `svg_2d_plot & plot_border_color(const svg_color & col);`

Set the color for the plot window background.

Set the color for the plot window background. Example: `.plot_border_color(lightgoldenrodyellow)`

Set the color for the plot window background.

22 `svg_color plot_border_color();`

Returns: the color for the plot window background.
 Returns: the color for the plot window background.
 Returns: the color for the plot window background.

23 `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.

24 `double plot_border_width();`

Returns: the width for the plot window border (svg units, default pixels).

25 `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.

Set the width for the plot window border (svg units, default pixels).

Set the width for the plot window border (svg units, default pixels).

26 `double plot_border_width();`

Returns: the width for the plot window border (svg units, default pixels).
 Returns: the width for the plot window border (svg units, default pixels).
 Returns: the width for the plot window border (svg units, default pixels).

27 `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).

28 `bool plot_window_on();`

Returns: true if a plot window is wanted (or false if the whole image is to be used).

29 `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).

Set true if a plot window is wanted (or false if the whole image is to be used).

Set true if a plot window is wanted (or false if the whole image is to be used).

30 `bool plot_window_on();`

Returns: true if a plot window is wanted (or false if the whole image is to be used).

Returns: true if a plot window is wanted (or false if the whole image is to be used).

Returns: true if a plot window is wanted (or false if the whole image is to be used).

31 `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.

32 `std::pair< double, double > plot_window_x();`

Returns: both the left and right (X axis) of the plot window.

33 `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.

Set the minimum and maximum (cartesian data units) for the plot window X axis. This is normally calculated from other plot values.

Set the minimum and maximum (cartesian data units) for the plot window X axis. This is normally calculated from other plot values.

34 `std::pair< double, double > plot_window_x();`

Returns: both the left and right (X axis) of the plot window.

Returns: both the left and right (X axis) of the plot window.

Returns: both the left and right (X axis) of the plot window.

35 `double plot_window_x_left();`

Returns: left of the plot window.

26 `double plot_window_x_left();`

Returns: left of the plot window.
 Returns: left of the plot window.
 Returns: left of the plot window.

27 `double plot_window_x_right();`

Returns: right of the plot window.

28 `double plot_window_x_right();`

Returns: right of the plot window.
 Returns: right of the plot window.
 Returns: right of the plot window.

29 `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.

30 `std::pair< double, double > plot_window_y();`

Returns: both the top and bottom (Y axis) of the plot window.

31 `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.

Set the minimum and maximum (cartesian data units) for the plot window Y axis. This is normally calculated from other plot values.

Set the minimum and maximum (cartesian data units) for the plot window Y axis. This is normally calculated from other plot values.

32 `std::pair< double, double > plot_window_y();`

Returns: both the top and bottom (Y axis) of the plot window.
 Returns: both the top and bottom (Y axis) of the plot window.
 Returns: both the top and bottom (Y axis) of the plot window.

33 `double plot_window_y_bottom();`

Returns: top of the plot window.

34 `double plot_window_y_bottom();`

Returns: top of the plot window.
 Returns: Top of the plot window.
 Returns: Top of the plot window.

25 `double plot_window_y_top();`

Returns: top of the plot window.

26 `double plot_window_y_top();`

Returns: top of the plot window.

Returns: top of the plot window.

Returns: top of the plot window.

27 `void set_ids();`

document ids for use in <g id = "PLOT_TITLE".../>

28 `svg_2d_plot & size(unsigned int x, unsigned int y);`

Set SVG image size (SVG units, default pixels).

Set SVG image size (SVG units, default pixels).

29 `std::pair< double, double > size();`

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

30 `svg_2d_plot & size(int x, int y);`

Set SVG image size (SVG units, default pixels).

Set SVG image size (SVG units, default pixels).

31 `std::pair< double, double > size();`

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

32 `svg_2d_plot & three_sd_color(const svg_color &);`

Set the color for three standard deviation (~99% confidence) ellipse fill.

33 `svg_color three_sd_color();`

Returns: Color for three standard deviation (~99% confidence) ellipse fill.

34 `svg_2d_plot & three_sd_color(const svg_color &);`

Set the color for three standard deviation (~99% confidence) ellipse fill.

Set the color for three standard deviation (~99% confidence) ellipse fill.

Set the color for three standard deviation (~99% confidence) ellipse fill.

25 `svg_color three_sd_color();`

Returns: Color for three standard deviation (~99% confidence) ellipse fill.
 Returns: Color for three standard deviation (~99% confidence) ellipse fill.
 Returns: Color for three standard deviation (~99% confidence) ellipse fill.

26 `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, for example:

```
my_plot.title("Plot of &#xA9; function (&#x00B0;C)");
```

Unicode symbols are at <http://unicode.org/charts/symbols.html>.

27 `const std::string title();`

Returns: Title for plot (whose string may include Unicode for greek letter and symbols).

28 `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, for example:

```
my_plot.title("Plot of &#xA9; function (&#x00B0;C)");
```

Unicode symbols are at <http://unicode.org/charts/symbols.html>.

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>.

Note



Only set plot title once.

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>.

Note



Only set plot title once.

29 `const std::string title();`

Example: `std::cout << "title \" " << my_plot.title() << " \"<< std::endl;` outputs: title "Plot showing several data point marker shapes & sizes."

Example: `std::cout << "title \" " << my_plot.title() << " \"<< std::endl;` outputs: title "Plot showing several data point marker shapes & sizes."

Returns: `std::string` text for the title, (whose string may include Unicode for greek letter and symbols).

Returns: Title for plot (whose string may include Unicode for greek letter and symbols).

Returns: Title for plot,

Returns: Title for plot,

20 `svg_2d_plot & title_color(const svg_color & col);`

Set the color of any title of the plot.

21 `svg_color title_color();`

Returns: the color of any title of the plot.

22 `svg_2d_plot & title_color(const svg_color & col);`

Set the color of any title of the plot.

Set the color of any title of the plot.

Set the color of any title of the plot.

23 `svg_color title_color();`

Returns: the color of any title of the plot.

Returns: the color of any title of the plot.

Returns: the color of any title of the plot.

24 `svg_2d_plot & title_font_alignment(align_style alignment);`

Set the alignment for the title.

25 `align_style title_font_alignment();`

Returns: the alignment for the title.

26 `svg_2d_plot & title_font_alignment(align_style alignment);`

Set the alignment for the title.

Set the alignment for the title.

Set the alignment for the title.

27 `align_style title_font_alignment();`

Returns: the alignment for the title.

Returns: the alignment for the title.
 Returns: the alignment for the title.

28 `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).

29 `const std::string & title_font_decoration();`

Returns: Font decoration for the title (default normal, or underline, overline or strike-thru).

20 `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).

Set the font decoration for the title (default normal, or underline, overline or strike-thru).

Set the font decoration for the title (default normal, or underline, overline or strike-thru).

21 `const std::string & title_font_decoration();`

Returns: Font decoration for the title (default normal, or underline, overline or strike-thru).
 Returns: the font decoration for the title (default normal, or underline, overline or strike-thru).
 Returns: the font decoration for the title (default normal, or underline, overline or strike-thru).

22 `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");.

23 `const std::string & title_font_family();`

Returns: Font family for the title.

24 `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");.

Set the font family for the title (for example: .title_font_family("Lucida Sans Unicode");

Set the font family for the title (for example: .title_font_family("Lucida Sans Unicode");

25 `const std::string & title_font_family();`

Returns: Font family for the title.
 Returns: the font family for the title
 Returns: the font family for the title

26 `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...

27 `int title_font_rotation();`

Returns: the rotation for the title font (degrees).

28 `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...).

Set the rotation for the title font (degrees, 0 to 360).

Set the rotation for the title font (degrees, 0 to 360).

29 `int title_font_rotation();`

Returns: the rotation for the title font (degrees).

Returns: the rotation for the title font (degrees).

Returns: the rotation for the title font (degrees).

30 `svg_2d_plot & title_font_size(unsigned int i);`

Sets the font size for the title (SVG units, default pixels).

Sets the font size for the title (svg units, default pixels).

31 `unsigned int title_font_size();`

Returns: Font size for the title (SVG units, default pixels).

32 `svg_2d_plot & title_font_size(int i);`

Sets the font size for the title (SVG units, default pixels).

Sets the font size for the title (svg units, default pixels).

33 `int title_font_size();`

return Font size for the title (SVG units, default pixels). Example: /code std::cout << my_plot.title_font_size() ...

Returns: the font size for the title (svg units, default pixels).

Returns: the font size for the title (svg units, default pixels).

34 `svg_2d_plot & title_font_stretch(const std::string & stretch);`

Set the font stretch for the title (default normal), wider or narrow.

35 `const std::string & title_font_stretch();`

Returns: Font stretch for the title.

36 `svg_2d_plot & title_font_stretch(const std::string & stretch);`

Set the font stretch for the title (default normal), wider or narrow.

Set the font stretch for the title (default normal), wider or narrow.

Set the font stretch for the title (default normal), wider or narrow.

27 `const std::string & title_font_stretch();`

Returns: Font stretch for the title.
 Returns: the font stretch for the title.
 Returns: the font stretch for the title.

28 `svg_2d_plot & title_font_style(const std::string & style);`

Set the font style for the title (default normal).

29 `const std::string & title_font_style();`

Returns: Font style for the title (default normal).

30 `svg_2d_plot & title_font_style(const std::string & style);`

Set the font style for the title (default normal).

Set the font style for the title (default normal).

Set the font style for the title (default normal).

31 `const std::string & title_font_style();`

Returns: Font style for the title (default normal).
 Returns: the font style for the title (default normal).
 Returns: the font style for the title (default normal).

32 `svg_2d_plot & title_font_weight(const std::string & weight);`

Set the font weight for the title (default normal).

33 `const std::string & title_font_weight();`

Returns: Font weight for the title.

34 `svg_2d_plot & title_font_weight(const std::string & weight);`

Set the font weight for the title (default normal).

Set the font weight for the title (default normal).

Set the font weight for the title (default normal).

35 `const std::string & title_font_weight();`

Returns: Font weight for the title.
 Returns: the font weight for the title.
 Returns: the font weight for the title.

36 `svg_2d_plot & title_on(bool cmd);`

If set true, show a title for the plot. Note: is set true by setting a title.

27 `bool title_on();`

Returns: true if will show a title for the plot.

28 `svg_2d_plot & title_on(bool cmd);`

If set true, show a title for the plot. Note: is set true by setting a title.

If set true, show a title for the plot.

If set true, show a title for the plot.

29 `bool title_on();`

If true, will show a title for the plot.

If true, will show a title for the plot.

Returns: true if will show a title for the plot.

30 `text_style & title_style();`

Returns: All style info for the title, font, famil, size ... (SVG units, default pixels).

Returns: Font size for the title (svg units, default pixels). Example: std::cout << "title style " << my_2d_plot.title_style() << std::endl; Outputs: title style text_style(10, "Lucida Sans Unicode", "", "normal", "", "", 900)

31 `svg_2d_plot & title_text_length(double);`

Sets the text_length for the title (SVG units, default pixels).

Sets the estimated text length for the title (svg units, default pixels).

32 `double title_text_length();`

return text_length for the title (SVG units, default pixels).

Returns: the estimated text length for the title (svg units, default pixels).

33 `void transform_pair(std::pair< double, double > & pt);`

Transform both x and y from Cartesian to SVG coordinates. SVG image is 0, 0 at top left, Cartesian at bottom left.

34 `svg_2d_plot & two_sd_color(const svg_color &);`

Set the color for two standard deviation (~95% confidence) ellipse fill.

35 `svg_color two_sd_color();`

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

36 `svg_2d_plot & two_sd_color(const svg_color &);`

Set the color for two standard deviation (~95% confidence) ellipse fill.

Set the color for two standard deviation (~95% confidence) ellipse fill.

Set the color for two standard deviation (~95% confidence) ellipse fill.

27 `svg_color two_sd_color();`

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

28 `void update_image();`

Draw the whole SVG image.

29 `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.

30 `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.

31 `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>".

32 `svg_color x_addlimits_color();`

Returns: the color of X confidence limits of value, for example, the color of "<1.23, 1.45>".

33 `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>".

Set the color of X confidence limits value, for example, the color of "<1.23 , 1.34>".

Set the color of X confidence limits value, for example, the color of "<1.23 , 1.34>".

34 `svg_color x_addlimits_color();`

Get the color of X confidence limits of value, for example, the color of "<1.23 , 1.34>"".

Get the color of X confidence limits of value, for example, the color of "<1.23 , 1.34>"".

Returns: the color of X confidence limits of value, for example, the color of "<1.23, 1.45>".

35 `svg_2d_plot & x_addlimits_on(bool b);`

Set if to append confidence limits to data point X values near data points markers.

26 `bool x_addlimits_on();`

Returns: true if to append confidence limits estimate to data point X values near data points markers.

27 `svg_2d_plot & x_addlimits_on(bool b);`

Set if to append confidence limits to data point X values near data points markers.

Set if to append confidence limits to data point X values near data points markers. (May not be implemented yet).

Set if to append confidence limits to data point X values near data points markers. (May not be implemented yet).

28 `bool x_addlimits_on();`

Returns: true if to append confidence limits estimate to data point X values near data points markers.

Returns: if to append confidence limits to data point X values near data points markers. (May not be implemented yet).

Returns: if to append confidence limits to data point X values near data points markers. (May not be implemented yet).

29 `double x_auto_max_value();`

Returns: X-axis maximum value computed by autoscale.

30 `double x_auto_max_value();`

Returns: X-axis maximum value computed by autoscale.

Returns: the X-axis maximum value computed by autoscale.

Returns: the X-axis maximum value computed by autoscale.

31 `double x_auto_min_value();`

Returns: X-axis minimum value computed by autoscale.

32 `double x_auto_min_value();`

Returns: X-axis minimum value computed by autoscale.

Returns: the X-axis minimum value computed by autoscale.

Returns: the X-axis minimum value computed by autoscale.

33 `double x_auto_tick_interval();`

Returns: the X-axis major tick interval computed by autoscale.

34 `double x_auto_tick_interval();`

Returns: the X-axis major tick interval computed by autoscale.

Returns: the X-axis major tick interval computed by autoscale.

Returns: the X-axis major tick interval computed by autoscale.

35 `int x_auto_ticks();`

Returns: the X-axis number of major ticks computed by autoscale.

36 `int x_auto_ticks();`

Returns: the X-axis number of major ticks computed by autoscale.
 Returns: the X-axis number of major ticks computed by autoscale.
 Returns: the X-axis number of major ticks computed by autoscale.

37 `bool x_autoscale();`

Returns: true if to use autoscale value for X-axis.

38 `svg_2d_plot & x_autoscale(bool b);`

Set true if to use autoscale values for X-axis.

39 `svg_2d_plot & x_autoscale(std::pair< double, double > p);`

autoscale X axis using a pair of doubles.

30 `svg_2d_plot & x_autoscale(const T & container);`

<

31 `svg_2d_plot & x_autoscale(const T & begin, const T & end);`

<

32 `bool x_autoscale();`

Returns: true if to use autoscale value for X-axis.
 Returns: true if to use autoscale value for X-axis.
 Returns: true if to use autoscale value for X-axis.

33 `svg_2d_plot & x_autoscale(bool b);`

Set true if to use autoscale values for X-axis.

Set true if to use autoscaled values for X-axis.

Set true if to use autoscaled values for X-axis.

Returns: Reference to caller to make chainable.
 Returns: Reference to caller to make chainable.

34 `svg_2d_plot & x_autoscale(std::pair< double, double > p);`

autoscale X axis using a pair of doubles.

Set to use X min & max pair of double values to autoscale X-axis.

Set to use X min & max pair of double values to autoscale X-axis.

35 `svg_2d_plot & x_autoscale(const T & container);`

<

Data series (all values) to use to calculate autoscaled X-axis values.

By default, use these calculated values.

Data series (all values) to use to calculate autoscaled X-axis values.

By default, use these calculated values.

36 `svg_2d_plot & x_autoscale(const T & begin, const T & end);`

<

Data series (range accessed using iterators) to use to calculate autoscaled X-axis values.

By default, use these calculated values.

Data series (range accessed using iterators) to use to calculate autoscaled X-axis values.

By default, use these calculated values.

37 `axis_line_style & x_axis();`

Returns: true if horizontal X-axis line to be drawn.

38 `svg_2d_plot & x_axis_color(const svg_color & col);`

Set the color of the X-axis line.

39 `svg_color x_axis_color();`

Returns: the color of the X-axis line.

40 `svg_2d_plot & x_axis_color(const svg_color & col);`

Set the color of the X-axis line.

Set the color of the X-axis line.

Set the color of the X-axis line.

41 `svg_color x_axis_color();`

Returns: the color of the X-axis line.

Returns: the color of the X-axis line.

Returns: the color of the X-axis line.

42 `svg_2d_plot & x_axis_label_color(const svg_color & col);`

Set X axis label color, for example, red.

33 `svg_color x_axis_label_color();`

Returns: X axis label color. X-axis ticks values label style.

34 `svg_2d_plot & x_axis_label_color(const svg_color & col);`

Set X axis label color, for example, red.

Set X axis label color.

Set X axis label color.

35 `svg_color x_axis_label_color();`

Returns: X axis label color. X-axis ticks values label style.

Returns: X axis label color.

Returns: X axis label color.

36 `svg_2d_plot & x_axis_on(bool is);`

If set true, draw a horizontal X-axis line.

37 `bool x_axis_on();`

Returns: true if will draw a horizontal X-axis line.

38 `svg_2d_plot & x_axis_on(bool is);`

If set true, draw a horizontal X-axis line.

If set true, draw a horizontal X-axis line.

If set true, draw a horizontal X-axis line.

39 `bool x_axis_on();`

If true, draw a horizontal X-axis line.

If true, draw a horizontal X-axis line.

Returns: true if will draw a horizontal X-axis line.

40 `const std::string x_axis_position();`

Returns: the position (or intersection with Y-axis) of the X-axis.

41 `const std::string x_axis_position();`

Returns: the position (or intersection with Y-axis) of the X-axis.

Returns: the position (or intersection with Y-axis) of the X-axis.

Returns: the position (or intersection with Y-axis) of the X-axis.

42 `svg_2d_plot & x_axis_vertical(double fraction);`

Set vertical position of X-axis for 1D as fraction of plot window.

33 `bool x_axis_vertical();`

Returns: vertical position of X-axis for 1D as fraction of plot window.

34 `svg_2d_plot & x_axis_vertical(double fraction);`

Set vertical position of X-axis for 1D as fraction of plot window.

Set vertical position of X-axis for 1D as fraction of plot window.

Set vertical position of X-axis for 1D as fraction of plot window.

35 `bool x_axis_vertical();`

Returns: vertical position of X-axis for 1D as fraction of plot window.

Returns: vertical position of X-axis for 1D as fraction of plot window.

Returns: vertical position of X-axis for 1D as fraction of plot window.

36 `svg_2d_plot & x_axis_width(double width);`

Set the width of X-axis lines.

37 `double x_axis_width();`

Returns: the width of X-axis lines.

38 `svg_2d_plot & x_axis_width(double width);`

Set the width of X-axis lines.

Set the width of X-axis lines.

Set the width of X-axis lines.

39 `double x_axis_width();`

Returns: the width of X-axis lines.

Returns: the width of X-axis lines.

Returns: the width of X-axis lines.

40 `svg_2d_plot & x_datetime_color(const svg_color & col);`

Set the color of X date time , for example, the color of text in "".

41 `svg_color x_datetime_color();`

Returns: the color of X date time, for example, the color of text in "".

42 `svg_2d_plot & x_datetime_color(const svg_color & col);`

Set the color of X date time , for example, the color of text in "".

Set the color of X point datetime, for example, the color of text in "2004-Jan-1 05:21:33.20".

Set the color of X point datetime, for example, the color of text in "2004-Jan-1 05:21:33.20".

38 `svg_color x_datetime_color();`

Get the color of X point date time, for example, the color of text in "2004-Jan-1 05:21:33.20".

Get the color of X point date time, for example, the color of text in "2004-Jan-1 05:21:33.20".

Returns: the color of X date time, for example, the color of text in "".

34 `svg_2d_plot & x_datetime_on(bool b);`

Set true if to append date time to data point X values near data points markers.

35 `bool x_datetime_on();`

Returns: true if to append an date time to data point X values near data points markers.

36 `svg_2d_plot & x_datetime_on(bool b);`

Set true if to append date time to data point X values near data points markers.

Set true if to append a datetime to data point X values near data points markers. (May not be implemented yet).

Set true if to append a datetime to data point X values near data points markers. (May not be implemented yet).

37 `bool x_datetime_on();`

Returns: true if to append an date time to data point X values near data points markers.

Returns: true if to append a ID or name to data point X values near data points markers. (May not be implemented yet).

Returns: true if to append a ID or name to data point X values near data points markers. (May not be implemented yet).

38 `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.

39 `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.

Set prefix, separator and suffix for x_style

**Note**

if you want a space, you must use a Unicode space "\ ", for example, ",\ " rather than just ", ".

Set prefix, separator and suffix for x_style

**Note**

if you want a space, you must use a Unicode space "\ ", for example, ",\ " rather than just ", ".

30 `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)".

31 `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)".

32 `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)".

Set the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

Set the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

33 `svg_color x_df_color();`

Get the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

Get the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

Returns: the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

34 `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.

35 `bool x_df_on();`

Returns: true if to append a degrees of freedom estimate to data point X values near data points markers.

36 `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.

Set true if to append a degrees of freedom estimate to data point X values near data points markers.

Set true if to append a degrees of freedom estimate to data point X values near data points markers.

37 `bool x_df_on();`

Returns: true if to append a degrees of freedom estimate to data point X values near data points markers.
 Returns: true if to append a degrees of freedom estimate to data point X values near data points markers. (May not be implemented yet).
 Returns: true if to append a degrees of freedom estimate to data point X values near data points markers. (May not be implemented yet).

38 `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".

39 `svg_color x_id_color();`

Returns: the color of X X id or name, for example, the color of text in "my_id".

40 `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".

Set the color of X ID or name, for example, the color of text in "My_id".

Set the color of X ID or name, for example, the color of text in "My_id".

41 `svg_color x_id_color();`

Get the color of X ID or name, for example, the color of text in "My_id".

Get the color of X ID or name, for example, the color of text in "My_id".

Returns: the color of X X id or name, for example, the color of text in "my_id".

42 `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.

43 `bool x_id_on();`

Returns: true if to append an ID or name to data point X values near data points markers.

44 `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.

Set true if to append a id or name to data point X values near data points markers. (May not be implemented yet).

Set true if to append a id or name to data point X values near data points markers. (May not be implemented yet).

45 `bool x_id_on();`

Returns: true if to append an ID or name to data point X values near data points markers.

Returns: true if to append an ID or name to data point X values near data points markers. (May not be implemented yet).

Returns: true if to append an ID or name to data point X values near data points markers. (May not be implemented yet).

36 `svg_2d_plot & x_label(const std::string & str);`

Set the text to label the X-axis (and set x_label_on(true)).

37 `std::string x_label();`

Returns: the text to label the X-axis.

38 `svg_2d_plot & x_label(const std::string & str);`

Set the text to label the X-axis (and set x_label_on(true)).

Set the text to label the X-axis (and set x_label_on(true)).

Set the text to label the X-axis (and set x_label_on(true)).

39 `std::string x_label();`

Returns: the text to label the X-axis.

Returns: the text to label the X-axis.

Returns: the text to label the X-axis.

40 `svg_2d_plot & x_label_color(const svg_color & col);`

Returns: the color of the Y-axis line.

41 `svg_color x_label_color();`

Returns: the color of X-axis label (including any units).

42 `svg_2d_plot & x_label_color(const svg_color & col);`

Set the color of X-axis label (including any units).

Set the color of X-axis label (including any units).

Returns: the color of the Y-axis line.

43 `svg_color x_label_color();`

Returns: the color of X-axis label (including any units).

Returns: the color of X-axis label (including any units).

Returns: the color of X-axis label (including any units).

44 `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"`

35 `const std::string & x_label_font_family();`

Returns: X tick value label font family.

36 `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"
```

Set X tick value label font family.

Set X tick value label font family.

37 `const std::string & x_label_font_family();`

Returns: X tick value label font family.

Returns: X tick value label font family.

Returns: X tick value label font family.

38 `svg_2d_plot & x_label_font_size(unsigned int i);`

Set X axis label font size (svg units, default pixels).

Set X axis label font size (svg units, default pixels).

39 `unsigned int x_label_font_size();`

Returns: X axis label font size (svg units, default pixels).

40 `svg_2d_plot & x_label_font_size(int i);`

Set X axis label font size (svg units, default pixels).

Set X axis label font size (svg units, default pixels).

41 `int x_label_font_size();`

Returns: X axis label font size (svg units, default pixels).

Returns: X axis label font size (svg units, default pixels).

Returns: X axis label font size (svg units, default pixels).

42 `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.

33 `bool x_label_on();`

Returns: true if to include an X-axis text label.

34 `svg_2d_plot & x_label_units(const std::string & str);`

Set the text to add units to the X-axis label.

35 `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`.

36 `svg_2d_plot & x_label_units(const std::string & str);`

Set the text to add units to the X-axis label.

Set the text to add units to the X-axis label.

Set the text to add units to the X-axis label.

37 `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`.

Returns: the text to add units to the X-axis label. The label will only be shown if `x_label_on() == true`.

Returns: the text to add units to the X-axis label. The label will only be shown if `x_label_on() == true`.

38 `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.

39 `bool x_label_units_on();`

Set true if want X axis label to include units (as well as label like "length").

40 `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.

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.

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.

31 `bool x_label_units_on();`

Set true if want X axis label to include units (as well as label like "length").

Set true if want X axis label to include units (as well as label like "length").

Set true if want X axis label to include units (as well as label like "length").

32 `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).

33 `double x_label_width();`

Returns: the width (boldness) of X-axis label (including any units).

34 `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).

Set the width (boldness) of X-axis label (including any units). (not recommended until browsers implement better).

Set the width (boldness) of X-axis label (including any units). (not recommended until browsers implement better).

35 `double x_label_width();`

Returns: the width (boldness) of X-axis label (including any units).

Returns: The width (boldness) of X-axis label (including any units).

Returns: The width (boldness) of X-axis label (including any units).

36 `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.

37 `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.

Set true if want to strip redundant zeros, signs and exponents. **Example:** reducing "1.2e+000" to "1.2". This markedly reduces visual clutter, and is the default.

Set true if want to strip redundant zeros, signs and exponents. **Example:** reducing "1.2e+000" to "1.2". This markedly reduces visual clutter, and is the default.

38 `svg_2d_plot & x_major_grid_color(const svg_color & col);`

Set the color of X-axis major grid lines.

39 `svg_color x_major_grid_color();`

Set the color of X-axis major grid lines.

40 `svg_2d_plot & x_major_grid_color(const svg_color & col);`

Set the color of X-axis major grid lines.

Set the color of X-axis major grid lines.

Set the color of X-axis major grid lines.

41 `svg_color x_major_grid_color();`

Set the color of X-axis major grid lines.

Returns: the color of X-axis major grid lines.

Returns: the color of X-axis major grid lines.

42 `svg_2d_plot & x_major_grid_on(bool is);`

If set true, will include a major X-axis grid.

43 `bool x_major_grid_on();`

Returns: true if will include a major X-axis grid.

44 `svg_2d_plot & x_major_grid_on(bool is);`

If set true, will include a major X-axis grid.

If set true, will include a major X-axis grid.

If set true, will include a major X-axis grid.

45 `bool x_major_grid_on();`

If true, will include a major X-axis grid.

If true, will include a major X-axis grid.

Returns: true if will include a major X-axis grid.

46 `svg_2d_plot & x_major_grid_width(double w);`

Set the width of X-axis major grid lines.

47 `double x_major_grid_width();`

Returns: the color of X-axis major grid lines.

48 `svg_2d_plot & x_major_grid_width(double w);`

Set the width of X-axis major grid lines.

Set the width of X-axis major grid lines.

Set the width of X-axis major grid lines.

49 `double x_major_grid_width();`

Returns: the color of X-axis major grid lines.
 Returns: the color of X-axis major grid lines.
 Returns: the color of X-axis major grid lines.

40 `svg_2d_plot & x_major_interval(double inter);`

Set the interval between X-axis major ticks.

41 `double x_major_interval();`

Returns: the interval between X-axis major ticks.

42 `svg_2d_plot & x_major_interval(double inter);`

Set the interval between X-axis major ticks.

Set the interval between X-axis major ticks.

Set the interval between X-axis major ticks.

43 `double x_major_interval();`

Returns: the interval between X-axis major ticks.
 Returns: the interval between X-axis major ticks.
 Returns: the interval between X-axis major ticks.

44 `svg_2d_plot & x_major_label_rotation(rotate_style rot);`

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

`rotate_style`

45 `rotate_style x_major_label_rotation();`

See Also:

`rotate_style`

Returns: rotation for X ticks major value labels.

46 `svg_2d_plot & x_major_label_rotation(rotate_style rot);`

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

`rotate_style`

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

rotate_style

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

rotate_style

47 `rotate_style x_major_label_rotation();`

See Also:

rotate_style

See Also:

rotate_style

See Also:

rotate_style

Returns: rotation for X ticks major value labels.

Returns: rotation for X ticks major value labels.

Returns: rotation for X ticks major value labels.

48 `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.

49 `int x_major_labels_side();`

Returns: side for label values for major ticks.

50 `svg_2d_plot & x_major_tick(double d);`

Set interval (Cartesian units) between major ticks.

51 `double x_major_tick();`

Returns: interval (Cartesian units) between major ticks.

52 `svg_2d_plot & x_major_tick(double d);`

Set interval (Cartesian units) between major ticks.

Set interval (Cartesian units) between major ticks.

Set interval (Cartesian units) between major ticks.

53 `double x_major_tick();`

Returns: interval (Cartesian units) between major ticks.

Returns: interval (Cartesian units) between major ticks.

Returns: interval (Cartesian units) between major ticks.

44 `svg_2d_plot & x_major_tick_color(const svg_color & col);`

Set the color of X-axis major ticks.

45 `svg_color x_major_tick_color();`

Returns: the color of X-axis major ticks.

46 `svg_2d_plot & x_major_tick_color(const svg_color & col);`

Set the color of X-axis major ticks.

Set the color of X-axis major ticks.

Set the color of X-axis major ticks.

47 `svg_color x_major_tick_color();`

Returns: the color of X-axis major ticks.

Returns: the color of X-axis major ticks.

Returns: the color of X-axis major ticks.

48 `svg_2d_plot & x_major_tick_length(double length);`

Set length of X major ticks (SVG units, default pixels).

49 `double x_major_tick_length();`

Set length of X major ticks (SVG units, default pixels).

50 `svg_2d_plot & x_major_tick_length(double length);`

Set length of X major ticks (SVG units, default pixels).

Set length of X major ticks.

Set length of X major ticks.

51 `double x_major_tick_length();`

Set length of X major ticks (SVG units, default pixels).

Returns: length of X major ticks.

Returns: length of X major ticks.

52 `svg_2d_plot & x_major_tick_width(double width);`

Set width of X major ticks (SVG units, default pixels).

53 `double x_major_tick_width();`

Set width of X major ticks (SVG units, default pixels).

44 `svg_2d_plot & x_major_tick_width(double width);`

Set width of X major ticks (SVG units, default pixels).

Set width of X major ticks.

Set width of X major ticks.

45 `double x_major_tick_width();`

Set width of X major ticks (SVG units, default pixels).

Returns: width of X major ticks.

Returns: width of X major ticks.

46 `svg_2d_plot & x_max(double x);`

Set the maximum value on the X-axis.

47 `double x_max();`

autoscale set & get parameters, Note: all these *MUST* precede x_autoscale(data) call.

Returns: the maximum value on the X-axis.

48 `svg_2d_plot & x_max(double x);`

Set the maximum value on the X-axis.

Set the maximum value on the X-axis.

Set the maximum value on the X-axis.

49 `double x_max();`

autoscale set & get parameters, Note: all these *MUST* precede x_autoscale(data) call.

Returns: the maximum value on the X-axis.

Returns: the maximum value on the X-axis. (Can also get both minimum and maximum as a std::pair).

Returns: the maximum value on the X-axis. (Can also get both minimum and maximum as a std::pair).

50 `svg_2d_plot & x_min(double min_x);`

Set the minimum value on the X-axis.

51 `double x_min();`

Returns: the minimum value on the X-axis.

52 `svg_2d_plot & x_min(double min_x);`

Set the minimum value on the X-axis.

Set the minimum value on the X-axis.

Set the minimum value on the X-axis.

48 `double x_min();`

Returns: the minimum value on the X-axis.

Returns: the minimum value on the X-axis. (Can also get both minimum and maximum as a std::pair).

Returns: the minimum value on the X-axis. (Can also get both minimum and maximum as a std::pair).

44 `svg_2d_plot & x_min_ticks(int min_ticks);`

Set X-axis autoscale to include at least minimum number of ticks (default = 6).

45 `int x_min_ticks();`

Returns: X-axis autoscale minimum number of ticks.

46 `svg_2d_plot & x_min_ticks(int min_ticks);`

Set X-axis autoscale to include at least minimum number of ticks (default = 6).

Set X-axis autoscale to include at least minimum number of ticks (default = 6). Must precede x_autoscale(data) call.

Set X-axis autoscale to include at least minimum number of ticks (default = 6). Must precede x_autoscale(data) call.

47 `int x_min_ticks();`

Returns: X-axis autoscale minimum number of ticks.

Returns: X-axis autoscale minimum number of ticks.

Returns: X-axis autoscale minimum number of ticks.

48 `svg_2d_plot & x_minor_grid_color(const svg_color & col);`

Set the color of X-axis minor grid lines.

49 `svg_color x_minor_grid_color();`

Returns: the color of X-axis minor grid lines.

50 `svg_2d_plot & x_minor_grid_color(const svg_color & col);`

Set the color of X-axis minor grid lines.

Set the color of X-axis minor grid lines.

Set the color of X-axis minor grid lines.

51 `svg_color x_minor_grid_color();`

Returns: the color of X-axis minor grid lines.

Returns: the color of X-axis minor grid lines.

Returns: the color of X-axis minor grid lines.

42 `svg_2d_plot & x_minor_grid_on(bool is);`

If set true, will include a minor X-axis grid.

43 `bool x_minor_grid_on();`

Returns: true if will include a major X-axis grid.

44 `svg_2d_plot & x_minor_grid_on(bool is);`

If set true, will include a minor X-axis grid.

If set true, will include a minor X-axis grid.

If set true, will include a minor X-axis grid.

45 `bool x_minor_grid_on();`

If true, will include a minor X-axis grid.

If true, will include a minor X-axis grid.

Returns: true if will include a major X-axis grid.

46 `svg_2d_plot & x_minor_grid_width(double w);`

Set the width of X-axis minor grid lines.

47 `double x_minor_grid_width();`

Returns: the width of X-axis minor grid lines.

48 `svg_2d_plot & x_minor_grid_width(double w);`

Set the width of X-axis minor grid lines.

Set the width of X-axis minor grid lines.

Set the width of X-axis minor grid lines.

49 `double x_minor_grid_width();`

Returns: the width of X-axis minor grid lines.

Returns: the width of X-axis minor grid lines.

Returns: the width of X-axis minor grid lines.

50 `double x_minor_interval();`

Returns: interval between X minor ticks.

51 `double x_minor_interval();`

Returns: interval between X minor ticks.

Returns: interval between X minor ticks.

Returns: interval between X minor ticks.

42 `svg_2d_plot & x_minor_interval(double interval);`

Set interval between X-axis minor ticks.

43 `svg_2d_plot & x_minor_interval(double interval);`

Set interval between X-axis minor ticks.

Set interval between X-axis minor ticks.

Set interval between X-axis minor ticks.

44 `svg_2d_plot & x_minor_tick_color(const svg_color & col);`

Set the color of X-axis minor ticks.

45 `svg_color x_minor_tick_color();`

Returns: the color of X-axis minor ticks.

46 `svg_2d_plot & x_minor_tick_color(const svg_color & col);`

Set the color of X-axis minor ticks.

Set the color of X-axis minor ticks.

Set the color of X-axis minor ticks.

47 `svg_color x_minor_tick_color();`

Returns: the color of X-axis minor ticks.

Returns: the color of X-axis minor ticks.

Returns: the color of X-axis minor ticks.

48 `svg_2d_plot & x_minor_tick_length(double length);`

Set length of X minor ticks (SVG units, default pixels).

49 `double x_minor_tick_length();`

Returns: length of X minor ticks (SVG units, default pixels).

50 `svg_2d_plot & x_minor_tick_length(double length);`

Set length of X minor ticks (SVG units, default pixels).

Set length of X minor ticks.

Set length of X minor ticks.

51 `double x_minor_tick_length();`

Returns: length of X minor ticks (SVG units, default pixels).
 Returns: length of X minor ticks.
 Returns: length of X minor ticks.

42 `svg_2d_plot & x_minor_tick_width(double width);`

Set width of X minor ticks (SVG units, default pixels).

43 `double x_minor_tick_width();`

Returns: width of X minor ticks (SVG units, default pixels).

44 `svg_2d_plot & x_minor_tick_width(double width);`

Set width of X minor ticks (SVG units, default pixels).

Set width of X minor ticks.

Set width of X minor ticks.

45 `double x_minor_tick_width();`

Returns: width of X minor ticks (SVG units, default pixels).
 Returns: width of X minor ticks.
 Returns: width of X minor ticks.

46 `svg_2d_plot & x_num_minor_ticks(unsigned int num);`

Set number of X-axis minor ticks between major ticks.

Set number of X-axis minor ticks between major ticks.

47 `unsigned int x_num_minor_ticks();`

Returns: number of X-axis minor ticks between major ticks.

48 `svg_2d_plot & x_num_minor_ticks(int num);`

Set number of X-axis minor ticks between major ticks.

Set number of X-axis minor ticks between major ticks.

49 `int x_num_minor_ticks();`

Returns: number of X-axis minor ticks between major ticks.
 Returns: number of X-axis minor ticks between major ticks. Note: NOT float or double!
 Returns: number of X-axis minor ticks between major ticks. Note: NOT float or double!

50 `svg_2d_plot & x_order_color(const svg_color & col);`

Set the color of X order #, for example, the color of #42.

41 `svg_color x_order_color();`

Returns: the color of X order #, for example, the color of #42.

42 `svg_2d_plot & x_order_color(const svg_color & col);`

Set the color of X order #, for example, the color of #42.

Set the color of X point order in sequence, for example, #3.

Set the color of X point order in sequence, for example, #3.

43 `svg_color x_order_color();`

Get the color of X point order in sequence, for example, #3.

Get the color of X point order in sequence, for example, #3.

Returns: the color of X order #, for example, the color of #42.

44 `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.

45 `bool x_order_on();`

Returns: true if to append an order # to data point X values near data points markers.

46 `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.

Set true if to append an order # to data point X values near data points markers.

Set true if to append an order # to data point X values near data points markers.

47 `bool x_order_on();`

Returns: true if to append an order # to data point X values near data points markers.

Returns: true if to append an order # to data point X values near data points markers.

Returns: true if to append an order # to data point X values near data points markers.

48 `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)".

49 `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)".

50 `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)".

Set the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

Set the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

41 `svg_color x_plusminus_color();`

Get the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

Get the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

Returns: the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

42 `svg_2d_plot & x_plusminus_on(bool b);`

Set if to append std_dev estimate to data point X values near data points markers.

43 `bool x_plusminus_on();`

Returns: true if to append std_dev estimate to data point X values near data points markers.

44 `svg_2d_plot & x_plusminus_on(bool b);`

Set if to append std_dev estimate to data point X values near data points markers.

Set if to append std_dev estimate to data point X values near data points markers. (May not be implemented yet).

Set if to append std_dev estimate to data point X values near data points markers. (May not be implemented yet).

45 `bool x_plusminus_on();`

Returns: true if to append std_dev estimate to data point X values near data points markers.

Returns: if to append std_dev estimate to data point X values near data points markers. (May not be implemented yet).

Returns: if to append std_dev estimate to data point X values near data points markers. (May not be implemented yet).

46 `const std::string x_prefix();`

Returns: the prefix.

47 `const std::string x_prefix();`

Get the prefix (only used if separator != "")

Get the prefix (only used if separator != "")

Returns: the prefix.

48 `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.

49 `std::pair< double, double > x_range();`

Returns: the range of values on the X-axis.

50 `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.

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 doubles, and the range must not be too small.

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 doubles, and the range must not be too small.

51 `std::pair< double, double > x_range();`

Returns: the range of values on the X-axis.

Returns: the range of values on the X-axis. (Need to use boost::svg::detail::operator<< to display this).

Returns: the range of values on the X-axis. (Need to use boost::svg::detail::operator<< to display this).

52 `const std::string x_separator();`

Returns: the separator, perhaps including Unicode.

53 `const std::string x_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;", for example, "\u00A0;" rather than " ".

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;", for example, "\u00A0;" rather than " ".

Returns: the separator, perhaps including Unicode.

54 `svg_2d_plot & x_size(unsigned int i);`

Set SVG image X-axis size (SVG units, default pixels).

Set SVG image X-axis size (SVG units, default pixels).

55 `unsigned int x_size();`

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

56 `svg_2d_plot & x_size(int i);`

Set SVG image X-axis size (SVG units, default pixels).

Set SVG image X-axis size (SVG units, default pixels).

57 `int x_size();`

Get SVG image X-axis size as horizontal width (SVG units, default pixels). Get SVG image X-axis size as horizontal width (SVG units, default pixels).

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

58 `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).

59 `int x_steps();`

Returns: autoscale to set ticks in steps.

50 `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).

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).

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).

51 `int x_steps();`

Returns: autoscale to set ticks in steps.
 Returns: autoscale to set ticks in steps.
 Returns: autoscale to set ticks in steps.

52 `const std::string x_suffix();`

Returns: the suffix (only used if separator != "")

53 `const std::string x_suffix();`

Get the suffix (only used if separator != "")

Get the suffix (only used if separator != "")
 Returns: the suffix (only used if separator != "")

54 `ticks_labels_style & x_ticks();`

Returns: true if ticks are to be marked on the X-axis.

55 `svg_2d_plot & x_ticks_down_on(bool cmd);`

Set true if Y major ticks should mark upwards.

56 `bool x_ticks_down_on();`

Returns: true if Y major ticks should mark upwards.

57 `svg_2d_plot & x_ticks_down_on(bool cmd);`

Set true if Y major ticks should mark upwards.

Set true if X major ticks should mark downwards.

Set true if X major ticks should mark downwards.

58 `bool x_ticks_down_on();`

Returns: true if Y major ticks should mark upwards.
 Returns: true if X major ticks should mark downwards.
 Returns: true if X major ticks should mark downwards.

59 `svg_2d_plot & x_ticks_on_window_or_axis(int);`

Returns: reference to `svg_2d_plot` to make chainable.

60 `int x_ticks_on_window_or_axis();`

Returns: if ticks on the plot window or on the X-axis.

51 `svg_2d_plot & x_ticks_up_on(bool cmd);`

Set true if X major ticks should mark upwards.

52 `bool x_ticks_up_on();`

Returns: true if X major ticks should mark upwards.

53 `svg_2d_plot & x_ticks_up_on(bool cmd);`

Set true if X major ticks should mark upwards.

Set true if X major ticks should mark upwards.

Set true if X major ticks should mark upwards.

54 `bool x_ticks_up_on();`

Returns: true if X major ticks should mark upwards.

Returns: true if X major ticks should mark upwards.

Returns: true if X major ticks should mark upwards.

55 `svg_2d_plot & x_ticks_values_color(const svg_color & col);`

Set X axis tick value label color.

56 `svg_color x_ticks_values_color();`

Returns: X-axis ticks value label color.

57 `svg_2d_plot & x_ticks_values_color(const svg_color & col);`

Set X axis tick value label color.

Set X axis tick value label color.

Set X axis tick value label color.

58 `svg_color x_ticks_values_color();`

Returns: X-axis ticks value label color.

Returns: X-axis ticks value label color.

Returns: X-axis ticks value label color.

59 `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"
```

50 `const std::string & x_ticks_values_font_family();`

Returns: X ticks value label font family.

51 `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"
```

Set X ticks value label font family.

Set X ticks value label font family.

52 `const std::string & x_ticks_values_font_family();`

Returns: X ticks value label font family.

Returns: X ticks value label font family.

Returns: X ticks value label font family.

53 `svg_2d_plot & x_ticks_values_font_size(unsigned int i);`

Set X ticks value label font size (svg units, default pixels).

Set X ticks value label font size (svg units, default pixels).

54 `unsigned int x_ticks_values_font_size();`

Set X ticks value label font size (svg units, default pixels).

55 `svg_2d_plot & x_ticks_values_font_size(int i);`

Set X ticks value label font size (svg units, default pixels).

Set X ticks value label font size (svg units, default pixels).

56 `int x_ticks_values_font_size();`

Set X ticks value label font size (svg units, default pixels).

Returns: X ticks value label font size (svg units, default pixels).

Returns: X ticks value label font size (svg units, default pixels).

57 `svg_2d_plot & x_ticks_values_ioflags(std::ios_base::fmtflags f);`

Set iostream format flags of data point X values near data points markers.

58 `std::ios_base::fmtflags x_ticks_values_ioflags();`

Returns: iostream format flags of data point X values near data points markers.

59 `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.

Set ostream format flags of data point X values near data points markers. Useful to set hexadecimal, fixed and scientific, (std::ios::scientific).

Set ostream format flags of data point X values near data points markers. Useful to set hexadecimal, fixed and scientific, (std::ios::scientific).

60 `std::ios_base::fmtflags x_ticks_values_ioflags();`

Returns: ostream format flags of data point X values near data points markers.

Returns: ostream format flags of data point X values near data points markers. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

Returns: ostream format flags of data point X values near data points markers. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

61 `svg_2d_plot & x_ticks_values_precision(int p);`

Set ostream decimal digits precision of data point X values near data points markers.

62 `int x_ticks_values_precision();`

Returns: ostream decimal digits precision of data point X values near data points markers.

63 `svg_2d_plot & x_ticks_values_precision(int p);`

Set ostream decimal digits precision of data point X values near data points markers.

Set ostream decimal digits precision of data point X values near data points markers.

Set ostream decimal digits precision of data point X values near data points markers.

64 `int x_ticks_values_precision();`

Returns: ostream decimal digits precision of data point X values near data points markers.

Returns: ostream decimal digits precision of data point X values near data points markers.

Returns: ostream decimal digits precision of data point X values near data points markers.

65 `svg_2d_plot & x_tight(double tight);`

Set tolerance to autoscale to permit data points slightly outside both end ticks.

66 `double x_tight();`

Returns: tolerance given to autoscale to permit data points slightly outside both end ticks. Get results of autoscaling.

67 `svg_2d_plot & x_tight(double tight);`

Set tolerance to autoscale to permit data points slightly outside both end ticks.

Set tolerance to autoscale to permit data points slightly outside both end ticks. default 0. Must precede x_autoscale(data) call.

Set tolerance to autoscale to permit data points slightly outside both end ticks. default 0. Must preceed x_autoscale(data) call.

58 `double x_tight();`

Returns: tolerance given to autoscale to permit data points slightly outside both end ticks. Get results of autoscaling.
 Returns: tolerance given to autoscale to permit data points slightly outside both end ticks.
 Returns: tolerance given to autoscale to permit data points slightly outside both end ticks.

59 `svg_2d_plot & x_value_font_size(unsigned int i);`

Set X tick value label font size (svg units, default pixels).

Set X tick value label font size (svg units, default pixels).

50 `unsigned int x_value_font_size();`

Returns: X tick value label font size (svg units, default pixels).

51 `svg_2d_plot & x_value_font_size(int i);`

Set X tick value label font size (svg units, default pixels).

Set X tick value label font size (svg units, default pixels).

52 `int x_value_font_size();`

Returns: X tick value label font size (svg units, default pixels).
 Returns: X tick value label font size (svg units, default pixels).
 Returns: X tick value label font size (svg units, default pixels).

53 `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)`

54 `std::ios_base::fmtflags x_value_ioflags();`

Returns: stream std::ios::fmtflags for control of format of X value labels.

55 `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)`

Set ostream std::ios::fmtflags for X value label (default decimal == 0X201). Mainly useful for changing to scientific, fixed or hexadecimal format. For example: `.x_value_ioflags(std::ios::dec | std::ios::scientific)`

Set iostream std::ios::fmtflags for X value label (default decimal == 0X201). Mainly useful for changing to scientific, fixed or hexadecimal format. For example: .x_value_ioflags(std::ios::dec | std::ios::scientific)

56 `std::ios_base::fmtflags x_value_ioflags();`

Returns: stream std::ios::fmtflags for control of format of X value labels.
 Returns: stream ioflags for control of format of X value labels.
 Returns: stream ioflags for control of format of X value labels.

57 `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.

58 `int x_value_precision();`

Returns: Precision of X-tick label values in decimal digits

59 `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.

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.

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.

60 `int x_value_precision();`

Returns: Precision of X-tick label values in decimal digits
 Returns: precision of X tick label values in decimal digits
 Returns: precision of X tick label values in decimal digits

61 `svg_2d_plot & x_values_color(const svg_color & col);`

Set the color of data point X values near data points markers.

62 `svg_color x_values_color();`

Returns: the color of data point X values near data points markers.

63 `svg_2d_plot & x_values_color(const svg_color & col);`

Set the color of data point X values near data points markers.

Set the color of data point X values near data points markers.

Set the color of data point X values near data points markers.

64 `svg_color x_values_color();`

Returns: the color of data point X values near data points markers.
Returns: the color of data point X values near data points markers.
Returns: the color of data point X values near data points markers.

55 `svg_2d_plot & x_values_font_family(const std::string & family);`

Set font family of data point X values near data points markers.

56 `const std::string & x_values_font_family();`

Returns: font family of data point X values near data points markers.

57 `svg_2d_plot & x_values_font_family(const std::string & family);`

Set font family of data point X values near data points markers.

Set font family of data point X values near data points markers.

Set font family of data point X values near data points markers.

58 `const std::string & x_values_font_family();`

Set font family of data point X values near data points markers.

Set font family of data point X values near data points markers.

Returns: font family of data point X values near data points markers.

59 `svg_2d_plot & x_values_font_size(unsigned int i);`

Set font size of data point X values near data points markers.

Set font size of data point X values near data points markers.

60 `unsigned int x_values_font_size();`

Returns: font size of data point X values near data points markers.

61 `svg_2d_plot & x_values_font_size(int i);`

Set font size of data point X values near data points markers.

Set font size of data point X values near data points markers.

62 `int x_values_font_size();`

Returns: font size of data point X values near data points markers.
Returns: font size of data point X values near data points markers.
Returns: font size of data point X values near data points markers.

63 `svg_2d_plot & x_values_ioflags(std::ios_base::fmtflags f);`

Set iostream format flags of data point X values near data points markers.

54 `std::ios_base::fmtflags x_values_ioflags();`

Returns: iostream format flags of data point X values near data points markers.

55 `svg_2d_plot & x_values_ioflags(std::ios_base::fmtflags f);`

Set iostream format flags of data point X values near data points markers.

Set iostream format flags of data point X values near data points markers. Useful to set hexadecimal, fixed and scientific, (std::ios::scientific).

Set iostream format flags of data point X values near data points markers. Useful to set hexadecimal, fixed and scientific, (std::ios::scientific).

56 `std::ios_base::fmtflags x_values_ioflags();`

Returns: iostream format flags of data point X values near data points markers.

Returns: iostream format flags of data point X values near data points markers. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

Returns: iostream format flags of data point X values near data points markers. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

57 `svg_2d_plot & x_values_on(bool b);`

Set true to show data point values near data points markers.

58 `bool x_values_on();`

Returns: true if to show data point values near data points markers.

59 `svg_2d_plot & x_values_on(bool b);`

Set true to show data point values near data points markers.

Returns: true if values of X data points are shown (for example: 1.23).

Returns: true if values of X data points are shown (for example: 1.23).

60 `bool x_values_on();`

If true, show data point values near data points markers.

If true, show data point values near data points markers.

Returns: true if to show data point values near data points markers.

61 `svg_2d_plot & x_values_precision(int p);`

Set iostream decimal digits precision of data point X values near data points markers.

62 `int x_values_precision();`

Returns: iostream decimal digits precision of data point X values near data points markers.

§3 `svg_2d_plot & x_values_precision(int p);`

Set iostream decimal digits precision of data point X values near data points markers.

Set iostream decimal digits precision of data point X values near data points markers.

Set iostream decimal digits precision of data point X values near data points markers.

§4 `int x_values_precision();`

Returns: iostream decimal digits precision of data point X values near data points markers.

Returns: iostream decimal digits precision of data point X values near data points markers.

Returns: iostream decimal digits precision of data point X values near data points markers.

§5 `svg_2d_plot & x_values_rotation(rotate_style rotate);`

Returns: the rotation (rotate_style) of data point X values near data points markers.

§6 `int x_values_rotation();`

Set the rotation (rotate_style) of data point X values near data points markers.

§7 `svg_2d_plot & x_values_rotation(rotate_style rotate);`

Returns: the rotation (rotate_style) of data point X values near data points markers.

Returns: the rotation (rotate_style) of data point X values near data points markers. (Degrees: 0 to 360 in 45 steps).

Returns: the rotation (rotate_style) of data point X values near data points markers. (Degrees: 0 to 360 in 45 steps).

§8 `int x_values_rotation();`

Set the rotation (rotate_style) of data point X values near data points markers.

Returns: the rotation of data point X values near data points markers.

Returns: the rotation of data point X values near data points markers.

§9 `svg_2d_plot & x_with_zero(bool b);`

Set X-axis autoscale to include zero (default = false).

§0 `bool x_with_zero();`

Returns: true if X-axis autoscale to include zero (default = false).

§1 `svg_2d_plot & x_with_zero(bool b);`

Set X-axis autoscale to include zero (default = false).

Set X-axis autoscale to include zero (default = false). Must preceed x_autoscale(data) call.

Set X-axis autoscale to include zero (default = false). Must preceed x_autoscale(data) call.

§2 `bool x_with_zero();`

Returns: true if X-axis autoscale to include zero (default = false).
 Returns: true if X-axis autoscale to include zero (default = false).
 Returns: true if X-axis autoscale to include zero (default = false).

53 `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.

54 `bool xy_autoscale();`

Returns: true if to autoscale both X and Y Axes.

55 `bool xy_values_on();`

Returns: true if values of X and Y data points are shown (for example: 1.23).

56 `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.

57 `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.

58 `const svg_color y_addlimits_color();`

Returns: Color of Y confidence interval.

59 `bool y_addlimits_on();`

Returns: true if values of Y data points are to include confidence interval.

60 `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.

61 `bool y_autoscale();`

Returns: true if to autoscale minimum and maximum for Y-axis.

62 `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.

63 `svg_2d_plot & y_autoscale(double minimum, double maximum);`

Set minimum & maximum Y values to use to autoscale Y-axis.

Parameters: `maxim-` 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.

64 `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.

65 `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.

66 `template<typename T> svg_2d_plot & y_autoscale(const T & container);`

Whole data series to use to calculate autoscaled values.

67 `axis_line_style & y_axis();`

Returns: true if vertical Y-axis line to be drawn.

68 `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.

69 `svg_color y_axis_color();`

Returns: Ty_axis stroke color.

70 `svg_2d_plot & y_axis_label_color(const svg_color & col);`

Set y_axis stroke color. (`svg_color(0, 0, 0)`), black is default color.)



Note

Setting the stroke color may produce fuzzy characters :-)

Returns: Reference to `svg_2d_plot` to make chainable.

61 `svg_color y_axis_label_color();`

Returns: Y-axis label stroke color.

62 `svg_2d_plot & y_axis_on(bool is);`

If set true, draw a vertical Y-axis line.

63 `bool y_axis_on();`

Returns: true if will draw a horizontal X-axis line.

64 `svg_2d_plot & y_axis_on(bool is);`

If set true, draw a vertical Y-axis line.

If set true, draw a vertical Y-axis line.

If set true, draw a vertical Y-axis line.

65 `bool y_axis_on();`

If true, draw a vertical Y-axis line.

If true, draw a vertical Y-axis line.

Returns: true if will draw a horizontal X-axis line.

66 `const std::string y_axis_position();`

Returns: Text information about Y-axis position.

67 `svg_2d_plot & y_axis_value_color(const svg_color & col);`

Set color of Y-axis **value** labels. (svg_color(0, 0, 0), black is default color.)

Returns: Reference to `svg_2d_plot` to make chainable.

68 `svg_color y_axis_value_color();`



Note

Only return the stroke color (not the fill_color).

Returns: Color of Y-axis tick value labels. (svg_color(0, 0, 0), black is default color.)

69 `svg_2d_plot & y_axis_width(double width);`

Set width of Y-axis line.

Returns: Reference to `svg_2d_plot` to make chainable.

60 `double y_axis_width();`

Returns: Width of Y-axis line.

61 `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.

62 `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.

63 `const svg_color y_df_color();`

Returns: Color of Y degrees of freedom.

64 `bool y_df_on();`

Returns: true if values of Y data points are to include degrees of freedom estimates.

65 `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.

66 `svg_2d_plot & y_label(const std::string & str);`

Set the text for the Y-axis label (and set `y_label_on(true)`).

67 `std::string y_label();`

Returns: the text for the Y-axis label. The label will only be shown if `y_label_on() == true`.

68 `svg_2d_plot & y_label(const std::string & str);`

Set the text for the Y-axis label (and set `y_label_on(true)`).

Set the text for the Y-axis label (and set `y_label_on(true)`).

Set the text for the Y-axis label (and set `y_label_on(true)`).

69 `std::string y_label();`

Returns: the text for the Y-axis label. The label will only be shown if `y_label_on() == true`.

Returns: the text for the Y-axis label. The label will only be shown if `y_label_on() == true`.

Returns: the text for the Y-axis label. The label will only be shown if `y_label_on() == true`.

70 `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.

71 `std::string y_label_axis();`

Returns: text to label Y-axis.

72 `svg_2d_plot & y_label_color(const svg_color & col);`

Set the color of Y-axis label (including any units).

73 `svg_color y_label_color();`

Returns: the color of Y-axis label (including any units).

74 `svg_2d_plot & y_label_color(const svg_color & col);`

Set the color of Y-axis label (including any units).

Set the color of Y-axis label (including any units).

Set the color of Y-axis label (including any units).

75 `svg_color y_label_color();`

Returns: the color of Y-axis label (including any units).

Returns: the color of Y-axis label (including any units).

Returns: the color of Y-axis label (including any units).

76 `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.

67 `const std::string & y_label_font_family();`

Returns: Font family for label on Y-axis.

68 `svg_2d_plot & y_label_font_size(unsigned int i);`

Set Y-axis label text font size.

69 `unsigned int y_label_font_size();`

Returns: Y-axis label text font size.

70 `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.

71 `bool y_label_on();`

Returns: true if Y-axis is to labelled.

72 `svg_2d_plot & y_label_units(const std::string & str);`

Set the text to add units to the Y-axis label.

73 `std::string y_label_units();`

Returns: the text to add units to the X-axis label.

74 `svg_2d_plot & y_label_units(const std::string & str);`

Set the text to add units to the Y-axis label.

Set the text to add units to the Y-axis label.

Set the text to add units to the Y-axis label.

75 `std::string y_label_units();`

Returns: the text to add units to the X-axis label.

Returns: the text to add units to the X-axis label.

Returns: the text to add units to the X-axis label.

76 `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.

```
67 bool y_label_units_on();
```

Returns: true if to add units text to the Y-axis label.

```
68 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.

```
69 const std::string & y_label_weight();
```

Returns: Y-axis label text font weight (for example: "bold").

```
70 svg_2d_plot & y_label_width(double width);
```

Set width of Y-axis value labels.

Returns: Reference to [svg_2d_plot](#) to make chainable.

```
71 double y_label_width();
```

Returns: Width of Y-axis value labels.

```
72 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.

```
73 bool y_labels_strip_e0s();
```

Returns: true if to strip unnecessary zeros, signs from labels.

```
74 svg_2d_plot & y_major_grid_color(const svg_color & col);
```

Set color of Y major grid lines. (`svg_color(200, 220, 255)`), greyblue is default color.) Example: `my_plot.y_major_grid_color(light-blue)`

Returns: Reference to [svg_2d_plot](#) to make chainable.

```
75 const svg_color y_major_grid_color();
```

Returns: Stroke Color of Y major grid lines. (`svg_color(200, 220, 255)`), greyblue is default color.)

```
76 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.

67 `bool y_major_grid_on();`

Returns: true to include major grid lines.

68 `svg_2d_plot & y_major_grid_width(double width);`

Set width of major grid lines.

Returns: Reference to `svg_2d_plot` to make chainable.

69 `double y_major_grid_width();`

Set width of major grid lines.

70 `double y_major_interval();`

Returns: major interval between ticks on Y-axis.

71 `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.

72 `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 (0).

Returns: Reference to `svg_2d_plot` to make chainable.

73 `int y_major_label_rotation();`

Returns: Rotation of Y-axis major tick labels.

74 `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.

75 `int y_major_labels_side();`

Returns: Position of labels (if any) for major ticks on vertical Y-axis line.

66 `svg_2d_plot & y_major_tick_color(const svg_color & col);`

Set color of Y major tick lines. (svg_color(0, 0, 0)), black is default color.)

Returns: Reference to `svg_2d_plot` to make chainable.

67 `const svg_color y_major_tick_color();`

Returns: Color of Y major tick lines. (svg_color(0, 0, 0)), black is default color.)

68 `double y_major_tick_length();`

Returns: Major tick length on Y-axis.

69 `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.

70 `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.

71 `double y_major_tick_width();`

Returns: Width of major ticks on Y-axis.

72 `double y_max();`

Returns: Maximum for Y-axis.

73 `double y_min();`

Returns: Minimum for Y-axis.

74 `svg_2d_plot & y_minor_grid_color(const svg_color & col);`

Set stroke color of Y minor grid lines. (svg_color(200, 220, 255)), greyblue is default color.)

Returns: Reference to `svg_2d_plot` to make chainable.

75 `const svg_color y_minor_grid_color();`

Returns: Color of Y minor grid lines.(svg_color(200, 220, 255)), greyblue is default color.)

76 `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.

67 `bool y_minor_grid_on();`

Set true to include minor grid lines.

68 `svg_2d_plot & y_minor_grid_width(double width);`

Set width of minor grid lines.

Returns: reference to `svg_2d_plot` to make chainable.

69 `double y_minor_grid_width();`

Returns: Width of minor grid lines.

70 `double y_minor_interval();`

Returns: interval between Y minor ticks.

71 `double y_minor_interval();`

Returns: interval between Y minor ticks.

Returns: interval between Y minor ticks.

Returns: interval between Y minor ticks.

72 `svg_2d_plot & y_minor_tick_color(const svg_color & col);`

Set color of Y minor tick lines. (`svg_color(0, 0, 0)`), black is default color.)

Returns: Reference to `svg_2d_plot` to make chainable.

73 `const svg_color y_minor_tick_color();`

Returns: Color of Y minor tick lines. (`svg_color(0, 0, 0)`), black is default color.)

74 `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.

75 `double y_minor_tick_length();`

Returns: Minor tick length on Y-axis.

76 `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.

67 `double y_minor_tick_width();`

Returns: Width of minor ticks on Y-axis.

68 `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.

69 `unsigned int y_num_minor_ticks();`

Returns: Number of minor ticks on Y-axis.

70 `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.

71 `const svg_color y_plusminus_color();`

Returns: Color of Y uncertainty of value.

72 `bool y_plusminus_on();`

Returns: true if values of Y data points are to include uncertainty estimates.

73 `svg_2d_plot & y_plusminus_on(bool b);`

Set true if values of Y data points are to include uncertainty estimates.

74 `const std::string y_prefix();`

Get the prefix (only used if separator != "")

75 `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.

76 `std::pair< double, double > y_range();`

Returns: The range (max and min) for Y-axis.

77 `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

```
"&#x00A0;"
```

rather than " ".

68

```
unsigned int y_size();
```

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

69

```
svg_2d_plot & y_size(unsigned int i);
```

Set SVG image Y-axis size (SVG units, default pixels).

Set SVG image Y-axis size (SVG units, default pixels).

70

```
int y_size();
```

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

71

```
svg_2d_plot & y_size(int i);
```

Set SVG image Y-axis size (SVG units, default pixels).

Set SVG image Y-axis size (SVG units, default pixels).

72

```
const std::string y_suffix();
```

Get the suffix (only used if separator != "")

73

```
ticks_labels_style & y_ticks();
```

Returns: true if ticks are to be marked on the Y-axis.

74

```
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.

75

```
bool y_ticks_left_on();
```

Returns: true if ticks on the Y-axis are to be on left of axis line.

76

```
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.

77 `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.

78 `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.

79 `bool y_ticks_right_on();`

Returns: true if ticks on the Y-axis are to be on right of axis line.

80 `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.

81 `svg_color y_ticks_values_color();`

Returns: color for Y-axis ticks values.

82 `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.

83 `const std::string & y_ticks_values_font_family();`

Returns: font family for Y-axis ticks values.

84 `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.

85 `unsigned int y_ticks_values_font_size();`

Returns: Font size for Y-axis values.

76 `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).

77 `std::ios_base::fmtflags y_ticks_values_ioflags();`

Returns: std::iostream format flags of ticks Y values. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

78 `svg_2d_plot & y_ticks_values_precision(int p);`

Set std::iostream decimal digits precision of ticks Y values.

79 `int y_ticks_values_precision();`

Returns: std::iostream decimal digits precision of ticks Y values..

80 `svg_2d_plot & y_value_ioflags(std::ios_base::fmtflags flags);`

Set std::ioflags of Y tick label values (default 0x201 == dec).

Example:

```
my_plot.x_value_ioflags(ios::dec | ios::scientific).x_value_precision(2);
```

Returns: Reference to `svg_2d_plot` to make chainable.

81 `int y_value_ioflags();`

Returns: All stream ioflags for control of format of Y tick value labels.

82 `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_ioflags(ios::dec | ios::scientific).x_value_precision(2);
```

Returns: Reference to `svg_2d_plot` to make chainable.

83 `int y_value_precision();`

Returns: Precision of Y tick value labels in decimal digits (default 3).

84 `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.

25 `svg_color y_values_color();`

Returns: Color for Y-axis values.

26 `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.

27 `const std::string & y_values_font_family();`

Returns: Font family for Y-axis values.

28 `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.

29 `unsigned int y_values_font_size();`

Returns: Font size for Y-axis values.

30 `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.

31 `std::ios_base::fmtflags y_values_ioflags();`

Returns: std::iostream format flags for data point values.

32 `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).

33 `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.

34 `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.

25 `int y_values_precision();`

Returns: iostream precision for data points Y values.

26 `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.

27 `int y_values_rotation();`

Returns: rotation for value labels on Y-axis.

svg_2d_plot protected member functions

1. `void adjust_limits(double & x, double & y);`

2. `void adjust_limits(double & x, double & y);`

If value of a data point reaches limit of max, min, infinity, use the appropriate plot min or max value instead.

If value of a data point reaches limit of max, min, infinity, use the appropriate plot min or max value instead.

3. `void clear_all();`

Clear all layers of the plot.

When writing to multiple documents, the contents of the plot may change significantly between. Rather than figuring out what has and has not changed, just erase the contents of the legend, title... in the document and start over.

4. `void clear_all();`

Clear all layers of the plot.

When writing to multiple documents, the contents of the plot may change significantly between. Rather than figuring out what has and has not changed, just erase the contents of the legend, title... in the document and start over.

5. `void clear_background();`

Clear the whole image background layer of the SVG plot.

6. `void clear_background();`

Clear the whole image background layer of the SVG plot.

< Clear the whole image background layer of the SVG plot.

< Clear the whole image background layer of the SVG plot.

7. `void clear_grids();`

Clear the grids layer of the SVG plot.

8. `void clear_grids();`

Clear the grids layer of the SVG plot.

< Clear the grids layer of the SVG plot.

< Clear the grids layer of the SVG plot.

9. `void clear_legend();`

Clear the legend layer of the SVG plot.

10. `void clear_legend();`

Clear the legend layer of the SVG plot.

< Clear the legend layer of the SVG plot.

< Clear the legend layer of the SVG plot.

11. `void clear_plot_background();`

Clear the plot area background layer of the SVG plot.

12. `void clear_plot_background();`

Clear the plot area background layer of the SVG plot.

< Clear the plot area background layer of the SVG plot.

< Clear the plot area background layer of the SVG plot.

13. `void clear_points();`

Clear the data points layer of the SVG plot.

14. `void clear_points();`

Clear the data points layer of the SVG plot.

< Clear the data points layer of the SVG plot.

< Clear the data points layer of the SVG plot.

15. `void clear_title();`

Clear the plot title layer of the SVG plot.

16. `void clear_title();`

Clear the Y axis layer of the SVG plot.

Clear the plot title layer of the SVG plot.

< Clear the plot title layer of the SVG plot.

< Clear the plot title layer of the SVG plot.

17. `void clear_x_axis();`

Clear the X axis layer of the SVG plot.

18. `void clear_x_axis();`

Clear the X axis layer of the SVG plot.

< Clear the X axis layer of the SVG plot.

< Clear the X axis layer of the SVG plot.

19. `void clear_y_axis();`

Clear the Y axis layer of the SVG plot.

20. `void clear_y_axis();`

< Clear the Y axis layer of the SVG plot.

< Clear the Y axis layer of the SVG plot.

21. `void draw_legend();`

22. `void draw_legend();`

Draw the legend border and background, (using the size and position computed by function `size_legend_box`), and legend text title (if any and if required), and any data point marker lines, and any shapes for data point markers, and any data series descriptor text(s).

Draw the legend border, text header (if any) and data point marker lines and/or shapes.

23. `void draw_plot_point(double x, double y, g_element & g_ptr,
const plot_point_style & sty);`

24. `void draw_plot_point(double x, double y, g_element & g_ptr,
plot_point_style & sty, unc< false > ux,
unc< false > uy);`

25. `void draw_plot_point(double x, double y, g_element & g_ptr,
const plot_point_style & sty);`

26. `void draw_plot_point(double x, double y, g_element & g_ptr,
plot_point_style & sty, unc< false > ux,
unc< false > uy);`

Draw a plot data point marker shape or symbol whose size and stroke and fill colors are specified in plot_point_style sty, possibly including uncertainty ellipses showing multiples of standard deviation.

Draw a plot data point marker shape or symbol whose size and stroke and fill colors are specified in plot_point_style sty, possibly including uncertainty ellipses showing multiples of standard deviation.

27. `void draw_plot_point_value(double x, double y, g_element & g_ptr,
value_style & val_style,
plot_point_style & point_style, Meas uvalue);`

28. `void draw_plot_point_value(double x, double y, g_element & g_ptr,
value_style & val_style,
plot_point_style & point_style, Meas uvalue);`

`void draw_plot_point_value(double x, double y, g_element& g_ptr, value_style& val_style, plot_point_style& point_style, unc<false> uvalue)` Write one data point (X or Y) value as a string, for example "1.23e-2", near the data point marker. Unnecessary e, +, & leading exponent zeros may optionally be stripped, and the position and rotation controlled. std_dev estimate, typically standard deviation (approximately half conventional 95% confidence "plus or minus") may be optionally be appended. Degrees of freedom estimate (number of replicates) may optionally be appended. ID or name of point, order in sequence, and datetime may also be added. For example: "3.45 +-0.1(10)"

The precision and format (scientific, fixed), and color and font type and size can be controlled too.

Unicode space text_plusminus glyph.

`void draw_plot_point_value(double x, double y, g_element& g_ptr, value_style& val_style, plot_point_style& point_style, unc<false> uvalue)` Write one data point (X or Y) value as a string, for example "1.23e-2", near the data point marker. Unnecessary e, +, & leading exponent zeros may optionally be stripped (highly recommended), and the position and rotation controlled. std_dev estimate, typically standard deviation (approximately half conventional 95% confidence "plus or minus") may be optionally be appended. Degrees of freedom estimate (number of replicates) may optionally be appended. ID or name of point, order in sequence, and datetime may also be added. For example: "3.45 +-0.1(10)"

The precision and format (scientific, fixed), and color and font type and size can be controlled too.

Unicode space and text_plusminus glyph.

29. `void draw_plot_point_values(double x, double y, g_element & x_g_ptr,
g_element & y_g_ptr, const value_style & x_sty,
const value_style & y_sty, Meas uncx, Meas uncny);`

30. `void draw_plot_point_values(double x, double y, g_element & x_g_ptr,
g_element & y_g_ptr, const value_style & x_sty,
const value_style & y_sty, Meas uncx,
unc< false > uncny);`

31. `void draw_plot_point_values(double x, double y, g_element & x_g_ptr,
g_element & y_g_ptr, const value_style & x_sty,
const value_style & y_sty, Meas uncx, Meas unc);`

32. `void draw_plot_point_values(double x, double y, g_element & x_g_ptr,
g_element & y_g_ptr, const value_style & x_sty,
const value_style & y_sty, Meas uncx,
unc< false > unc);`

Write the **pair** of data points X and Y values as a string.

The x parameter also carries the measurement information for the pair, and so is a **Meas**, not just an **unc<false>** as is the Y parameter. If a separator starting with newline, then both on the same line, for example "1.23, 3.45", or "[5.6, 7.8] X value_style is used to provide the prefix and separator, and Y value_style to provide the suffix. For example,

`x_style prefix("[X=", and separator ",\u00A0;Y= ", " and Y value_style = "]")` will produce a value label like "[X=-1.23, Y=4.56]"



Note

You need to use a Unicode space for get space for all browsers. For a long a string you may need to make the total image size bigger, and to orient the value labels with care. `draw_plot_point_values` is only when both X and Y pairs are wanted.

Also strip unnecessary e, + and leading exponent zeros, if required.

Unicode space text_plusminus glyph.

Write the **pair** of data points X and Y values as a string.

The x parameter also carries the measurement information for the pair, and so is a **Meas**, not just an **unc<false>** as is the Y parameter. If a separator starting with newline, then both on the same line, for example "1.23, 3.45", or "[5.6, 7.8] X value_style is used to provide the prefix and separator, and Y value_style to provide the suffix. For example,

`x_style prefix("[X=", and separator ",\u00A0;Y= ", " and Y value_style = "]")` will produce a value label like "[X=-1.23, Y=4.56]"



Note

You need to use a Unicode space for get space for all browsers. For a long a string you may need to make the total image size bigger, and to orient the value labels with care. `draw_plot_point_values` is only when both X and Y pairs are wanted.

Also strip unnecessary e, + and leading exponent zeros, if required.

Unicode space text_plusminus glyph.

33. `void draw_title();`

34. `void draw_title();`

Draw title (for the whole plot). Update `title_info_` with position. Assumes `align = center_align`. Using `center_align` will ensure that title will center correctly because the render engine does the centering. (Even if the original string is made much longer because

it contains Unicode, Greek, math symbols etc, taking about 8 characters per symbol. For example, the Unicode symbol for square root is "√" but takes only about one character width).

Draw title (for the whole plot). Update title_info_ with position. Assumes align = center_align Using center_align will ensure that title will center correctly because the render engine does the centering. (Even if the original string is made much longer because it contains Unicode, Greek, math symbols etc, taking about 8 characters per symbol. For example, the Unicode symbol for square root is "√" but takes only about one character width).

35. `void draw_x_axis();`

36. `void draw_x_axis();`

Draw horizontal X-axis line & plot window line to hold, and ticks and grids.

Draw horizontal X-axis line & plot window line to hold, and ticks and grids.

37. `void draw_x_axis_label();`

38. `void draw_x_axis_label();`

Draw the X-axis label text (for example, length), and append any optional units (for example, km).

Draw the X-axis label text (for example, length), and append any optional units (for example, km).

39. `void draw_x_major_tick(double i, path_element & tick_path,
path_element & grid_path);`

40. `void draw_x_major_tick(double i, path_element & tick_path,
path_element & grid_path);`

Draw major ticks - and grid too if wanted. If major_value_labels_side then value shown beside the major tick.

Draw major ticks - and grid too if wanted. If major_value_labels_side then value shown beside the major tick.

41. `void draw_x_minor_tick(double j, path_element & tick_path,
path_element & grid_path);`

42. `void draw_x_minor_tick(double j, path_element & tick_path,
path_element & grid_path);`

< Draw X-axis minor ticks, and optional grid. (Value is NOT (yet) shown beside the minor tick).

< Draw X-axis minor ticks, and optional grid. (Value is NOT (yet) shown beside the minor tick).

43. `void place_legend_box();`

44. `void place_legend_box();`

Place legend box (if required). Default legend position is outside top right, level with plot window.

Place legend box (if required).

45. `void size_legend_box();`

46. `void size_legend_box();`

Calculate how big the legend box needs to be to hold the legend title and the data point markers (symbols or shapes), and any line marks showing lines used joining points, and any data series descriptor text(s).

Calculate how big the legend box needs to be to hold the title and any point markers (symbols or lines) and any series descriptor text.

47. `void transform_point(double & x, double & y);`

48. `void transform_point(double & x, double & y);`

< Scale & shift both X & Y to graph Cartesian coordinates.

< Scale & shift both X & Y to graph Cartesian coordinates.

49. `void transform_x(double & x);`

50. `void transform_x(double & x);`

< Scale and shift X value only.

< Scale and shift X value only.

51. `void transform_y(double & y);`

52. `void transform_y(double & y);`

< Scale and shift Y value only.

< Scale and shift Y value only.

svg_2d_plot public public data members

1. `bool is_a_data_series_line_;`

true if any series have lines to show in legend (default false). Example: .line_on(true).

2. `std::string plot_window_clip_;`

= "clip_plot_window" id for clippath <http://www.w3.org/TR/SVG/masking.html#ClipPathElement> 14.1 Introduction clipping paths,

which uses any combination of 'path', 'text' and basic shapes to serve as the outline where everything on the "inside" of the outline is allowed to show through but everything on the outside is masked out. So the plot_window_clip_limits display to a plot_window rectangle.

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 = "");

    // private member functions
    void draw_straight_lines(const svg_2d_plot_series &);

    // 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 & point_font_decoration(const std::string);
    const std::string point_font_decoration();
    svg_2d_plot_series & point_font_family(const std::string);
    const std::string point_font_family();
    svg_2d_plot_series & point_font_stretch(const std::string);
    const std::string point_font_stretch();
    svg_2d_plot_series & point_font_style(const std::string);
    const std::string point_font_style();
    svg_2d_plot_series & point_font_weight(const std::string);
    const std::string point_font_weight();
    plot_point_style point_style();
    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();
```

```
// public data members
bar_style bar_style_; // Style of bar used in histograms.
histogram_style histogram_style_; // Style of histogram.
plot_point_style limit_point_style_; // At limit data point marker.
plot_line_style line_style_; // Style (color, width...) of line joining data points.
plot_point_style point_style_; // Data point marker like circle, square...
std::multimap< Meas, unc< false > > series_; // Normal 'OK to plot' data values.
std::multimap< double, double > series_limits_; // 'limit' values: too big or small, or NaN.
std::string title_; // Title of data series (to show on legend using legend style).
};
```

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:

<code>begin</code>	Starting iterator into container of data series <code>begin()</code> to start at the beginning.
<code>end</code>	Ending iterator into container of data series, <code>end()</code> to finish with the last item.
<code>title</code>	Title for the plot.

Template Parameters:

<code>T</code>	an STL container: for example: <code>multimap</code> .
----------------	--

svg_2d_plot_series private member functions

1. `void draw_straight_lines(const svg_2d_plot_series & series);`

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:

<code>col</code>	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 joing 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 (stroke) color. Example: .line_color(blue)
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 & point_font_decoration(const std::string decoration_);`
- Set Data series point marker font decoration. Example `my_plot.plot(my_data, "my_data").shape("Z").point_font_decoration("underline")`
24. `const std::string point_font_decoration();`
- Returns: Decoration of data point marker(s). Examples: "underline", "strikethru".
25. `svg_2d_plot_series & point_font_family(const std::string family_);`

Set Data series point marker font family. Example `my_plot.plot(my_data, "my_data").shape("Z").point_font_family("times new roman")`

26. `const std::string point_font_family();`

Returns: Font family of data point marker(s) symbol. Examples: "arial", "verdana", "courier" ...

27. `svg_2d_plot_series & point_font_stretch(const std::string stretch_);`

Set Data series point marker font stretch. Example `my_plot.plot(my_data, "my_data").shape("Z").point_font_stretch("narrow")`

28. `const std::string point_font_stretch();`

Returns: stretch of data point marker(s). Examples: "narrow", "wide".

29. `svg_2d_plot_series & point_font_style(const std::string style_);`

Set Data series point marker font style. Example `my_plot.plot(my_data, "my_data").shape("Z").point_font_style("italic")`

30. `const std::string point_font_style();`

Returns: style of data point marker(s) "bold", "italic" ..

31. `svg_2d_plot_series & point_font_weight(const std::string weight_);`

Set Data series point marker font weight. Example `my_plot.plot(my_data, "my_data").shape("Z").point_font_weight("bold")`

32. `const std::string point_font_weight();`

Returns: weight of data point marker(s) "bold".

33. `plot_point_style point_style();`

Returns: Point style for data point marker(s).

34. `svg_2d_plot_series & shape(point_shape shape_);`

Set Data series point marker shape. Example: `.shape(square)`, `.shape(circlet)`

35. `point_shape shape();`

Returns: shape of data point marker(s). Examples "square", "cone", circlet ... (See enum point_shape).

36. `svg_2d_plot_series & size(int size_);`

Set Data series point marker size. Example `.shape(square).size(5)`

37. `int size();`

Returns: Size of data point marker(s).

38. `svg_2d_plot_series` & `stroke_color(const svg_color & col_);`

Set Data series point marker stroke color.

39. `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

See Also:

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);
    bool autoscale();
    svg_boxplot & autoscale(bool);
    svg_boxplot & autoscale_check_limits(bool);
    bool autoscale_check_limits();
    svg_boxplot & autoscale_check_limits(bool);
    bool autoscale_check_limits();
    svg_boxplot & autoscale_plusminus(double);
    double autoscale_plusminus();
    svg_boxplot & autoscale_plusminus(double);
    double autoscale_plusminus();
    svg_boxplot & axes_on(bool);
    bool axes_on();
    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_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 & 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();
    void calculate_plot_window();
    void clear_all();
    svg_boxplot & confidence(double);
    double confidence();
    svg_boxplot & confidence(double);
    double confidence();
    svg_boxplot & coord_precision(int);
    int coord_precision();
    svg_boxplot & coord_precision(int);
    int coord_precision();
    svg_boxplot & copyright_date(const std::string);
    const std::string copyright_date();
    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 & copyright_holder(const std::string);
const std::string copyright_holder();
svg_boxplot & data_lines_width(double);
double data_lines_width();
svg_boxplot & data_lines_width(double);
double data_lines_width();
svg_boxplot & derived();
const svg_boxplot & derived() const;
svg_boxplot & derived();
const svg_boxplot & derived() const;
svg_boxplot & description(const std::string);
const std::string & description();
svg_boxplot & description(const std::string);
const std::string & description();
svg_boxplot & document_title(const std::string);
std::string document_title();
svg_boxplot & document_title(const std::string);
std::string document_title();
void draw_box(double, double, double, double, const svg_style &,
void draw_boxplot(svg_boxplot_series &, double);
svg_boxplot &
draw_line(double, double, double, double, const svg_color & = black);
svg_boxplot &
draw_line(double, double, double, double, const svg_color & = black);
void draw_median(double, double, double, const svg_style &,
    const value_style &);
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_note(double, double, std::string, rotate_style = horizontal,
    align_style = center_align, const svg_color & = black,
    text_style & = no_style);
void draw_outliers(double, const std::vector< double > &,
    const std::vector< double > &, const plot_point_style &,
    const plot_point_style &, const value_style &);
svg_boxplot &
draw_plot_curve(double, double, double, double, double,
    const svg_color & = black);
svg_boxplot &
draw_plot_curve(double, double, double, double, double,
    const svg_color & = black);
svg_boxplot &
draw_plot_line(double, double, double, const svg_color & = black);
svg_boxplot &
draw_plot_line(double, double, double, const svg_color & = black);
void draw_title();
void draw_whiskers(double, double, double, double, const svg_style &,
    const svg_style &, const svg_style &);
void draw_x_axis();
void draw_x_axis_label();
void draw_x_major_tick(double, path_element &, text_element &);
void draw_y_axis();
void draw_y_axis_label();
void draw_y_major_tick(double, path_element &, path_element &);
void draw_y_minor_tick(double, path_element &, path_element &);
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_margin(double);
double image_border_margin();
svg_boxplot & image_border_width(double);
double image_border_width();
svg_boxplot & image_border_width(double);
double image_border_width();
unsigned int image_x_size();
svg_boxplot & image_x_size(unsigned int);
int image_x_size();
svg_boxplot & image_x_size(int);
unsigned int image_y_size();
svg_boxplot & image_y_size(unsigned int);
int image_y_size();
svg_boxplot & image_y_size(int);
svg_boxplot & legend_background_color(const svg_color &);
svg_color legend_background_color();
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();
svg_boxplot & legend_border_color(const svg_color &);
svg_color legend_border_color();
const std::pair< double, double > legend_bottom_right();
const std::pair< double, double > legend_bottom_right();
bool legend_box_fill_on();
bool legend_box_fill_on();
svg_boxplot & legend_color(const svg_color &);
svg_color legend_color();
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_family(const std::string &);
const std::string & legend_font_family();
svg_boxplot & legend_font_size(int);
int legend_font_size();
svg_boxplot & legend_font_weight(const std::string &);
const std::string & legend_font_weight();
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_lines(bool);
bool legend_lines();
svg_boxplot & legend_on(bool);
bool legend_on();
svg_boxplot & legend_on(bool);
bool legend_on();
bool legend_outside();
bool legend_outside();
svg_boxplot & legend_place(legend_places);
legend_places legend_place();

```

```

svg_boxplot & legend_place(legend_places);
legend_places legend_place();
svg_boxplot & legend_text_font_size(int);
int legend_text_font_size();
svg_boxplot & legend_text_font_weight(const std::string &);
const std::string & legend_text_font_weight();
svg_boxplot & legend_title(const std::string);
const std::string legend_title();
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_title_font_size(int);
int legend_title_font_size();
svg_boxplot & legend_title_font_weight(const std::string &);
const std::string & legend_title_font_weight();
svg_boxplot & legend_top_left(double, double);
const std::pair< double, double > legend_top_left();
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 & legend_width(double);
double legend_width();
svg_boxplot &
license(std::string = "permits", std::string = "permits",
       std::string = "requires", std::string = "permits",
       std::string = "permits");
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_attribution();
const std::string license_commercialuse();
const std::string license_commercialuse();
const std::string license_distribution();
const std::string license_distribution();
svg_boxplot & license_on(bool);
bool license_on();
svg_boxplot & license_on(bool);
bool license_on();
const std::string license_reproduction();
const std::string license_reproduction();
svg_boxplot & limit_color(const svg_color &);
svg_color limit_color();
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 & 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 & 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_border_width(double);
double plot_border_width();
svg_boxplot & plot_window_on(bool);
bool plot_window_on();
svg_boxplot & plot_window_on(bool);
bool plot_window_on();
svg_boxplot & plot_window_x(double, double);
std::pair<double, double> plot_window_x();
svg_boxplot & plot_window_x(double, double);
std::pair<double, double> plot_window_x();
double plot_window_x_left();
double plot_window_x_left();
double plot_window_x_right();
double plot_window_x_right();
svg_boxplot & plot_window_y(double, double);
std::pair<double, double> plot_window_y();
svg_boxplot & plot_window_y(double, double);
std::pair<double, double> plot_window_y();
double plot_window_y_bottom();
double plot_window_y_bottom();
double plot_window_y_top();
double plot_window_y_top();
svg_boxplot & quartile_definition(int);
int quartile_definition();
svg_boxplot & size(int, int);
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 & 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_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_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_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_rotation(rotate_style);
int title_font_rotation();
svg_boxplot & title_font_size(unsigned int);
unsigned int title_font_size();
svg_boxplot & title_font_size(int);
int title_font_size();
svg_boxplot & title_font_stretch(const std::string &);
const std::string & title_font_stretch();
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_style(const std::string &);
const std::string & title_font_style();
svg_boxplot & title_font_weight(const std::string &);
const std::string & title_font_weight();
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);
text_style & title_style();
svg_boxplot & title_text_length(double);
double title_text_length();
void transform_x(double &);
void transform_y(double &);
svg_boxplot & two_sd_color(const svg_color &);
svg_color two_sd_color();
svg_boxplot & two_sd_color(const svg_color &);
svg_color two_sd_color();
void update_image();
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_color(const svg_color &);
svg_color x_addlimits_color();
svg_boxplot & x_addlimits_on(bool);
bool x_addlimits_on();
svg_boxplot & x_addlimits_on(bool);
bool x_addlimits_on();
double x_auto_max_value();
double x_auto_max_value();
double x_auto_min_value();
double x_auto_min_value();
double x_auto_tick_interval();
double x_auto_tick_interval();
int x_auto_ticks();
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 &);
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_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_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_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_vertical(double);
bool x_axis_vertical();
svg_boxplot & x_axis_width(double);
double x_axis_width();
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_color(const svg_color &);
svg_color x_datetime_color();
svg_boxplot & x_datetime_on(bool);
bool x_datetime_on();
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_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_color(const svg_color &);
svg_color x_df_color();
svg_boxplot & x_df_on(bool);
bool x_df_on();
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_color(const svg_color &);
svg_color x_id_color();
svg_boxplot & x_id_on(bool);
bool x_id_on();
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_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_font_size(int);
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(const std::string &);
std::string x_label_units();
svg_boxplot & x_label_units_on(bool);
bool x_label_units_on();
svg_boxplot & x_label_units_on(bool);
bool x_label_units_on();
svg_boxplot & x_label_width(double);
double x_label_width();
double x_label_width();
svg_boxplot & x_labels_strip_e0s(bool);
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_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_on(bool);
bool x_major_grid_on();
svg_boxplot & x_major_grid_width(double);
double x_major_grid_width();
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_interval(double);
double x_major_interval();
svg_boxplot & x_major_label_rotation(rotate_style);
rotate_style x_major_label_rotation();
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_labels_side(int);
int x_major_labels_side();
svg_boxplot & x_major_tick(double);
double x_major_tick();
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_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_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_max(double);
double x_max();
svg_boxplot & x_min(double);
double x_min();
svg_boxplot & x_min(double);
double x_min();
svg_boxplot & x_min_ticks(int);
int x_min_ticks();
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_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_on(bool);
bool x_minor_grid_on();
svg_boxplot & x_minor_grid_width(double);
double x_minor_grid_width();
svg_boxplot & x_minor_grid_width(double);
double x_minor_grid_width();
double x_minor_interval();
double x_minor_interval();
svg_boxplot & x_minor_interval(double);
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_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_length(double);
double x_minor_tick_length();
svg_boxplot & x_minor_tick_width(double);
double x_minor_tick_width();
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_num_minor_ticks(int);
int x_num_minor_ticks();
svg_boxplot & x_order_color(const svg_color &);

svg_color x_order_color();
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_order_on(bool);
bool x_order_on();
svg_boxplot & x_plusminus_color(const svg_color &);

svg_color x_plusminus_color();
svg_boxplot & x_plusminus_color(const svg_color &);

svg_color x_plusminus_color();
svg_boxplot & x_plusminus_on(bool);
bool x_plusminus_on();

```

```

svg_boxplot & x_plusminus_on(bool);
bool x_plusminus_on();
const std::string x_prefix();
const std::string x_prefix();
svg_boxplot & x_range(double, double);
std::pair< double, double > x_range();
svg_boxplot & x_range(double, double);
std::pair< double, double > x_range();
const std::string x_separator();
const std::string x_separator();
svg_boxplot & x_size(unsigned int);
svg_boxplot & x_size(int);
unsigned int x_size();
svg_boxplot & x_steps(int);
int x_steps();
svg_boxplot & x_steps(int);
int x_steps();
const std::string x_suffix();
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_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_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_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_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_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_font_size(int);
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_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_ticks_values_precision(int);
int x_ticks_values_precision();
svg_boxplot & x_tight(double);
double x_tight();
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_font_size(int);

```

```

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_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_value_precision(int);
int x_value_precision();
svg_boxplot & x_values_color(const svg_color &);
svg_color x_values_color();
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_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_font_size(int);
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_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_on(bool);
bool x_values_on();
svg_boxplot & x_values_precision(int);
int x_values_precision();
svg_boxplot & x_values_precision(int);
int x_values_precision();
svg_boxplot & x_values_rotation(rotate_style);
int x_values_rotation();
svg_boxplot & x_values_rotation(rotate_style);
int x_values_rotation();
svg_boxplot & x_with_zero(bool);
bool x_with_zero();
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_color(const svg_color &);
svg_color y_axis_color();
svg_boxplot & y_axis_on(bool);
bool y_axis_on();
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();
svg_boxplot & y_label_units(const std::string &);

std::string y_label_units();
bool y_labels_strip_e0s();
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();
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);
svg_boxplot & y_size(int);
unsigned int y_size();

// protected member functions
void adjust_limits(double &, double &);

void adjust_limits(double &, double &);

void clear_background();
void clear_background();
void clear_grids();
void clear_grids();
void clear_legend();
void clear_legend();
void clear_plot_background();
void clear_plot_background();
void clear_points();
void clear_points();
void clear_title();
void clear_title();
void clear_x_axis();
void clear_x_axis();
void clear_y_axis();
void clear_y_axis();
void draw_legend();
void draw_legend();
void draw_plot_point(double, double, g_element &, const plot_point_style &);

void draw_plot_point(double, double, g_element &, plot_point_style &,
                     unc< false >, unc< false >);

void draw_plot_point(double, double, g_element &, const plot_point_style &);

void draw_plot_point(double, double, g_element &, plot_point_style &,
                     unc< false >, unc< false >);

void draw_plot_point_value(double, double, g_element &, value_style &,
                           plot_point_style &, Meas);

void draw_plot_point_value(double, double, g_element &, value_style &,
                           plot_point_style &, Meas);

void draw_plot_point_values(double, double, g_element &, g_element &,
                           const value_style &, const value_style &, Meas,
                           Meas);

void draw_plot_point_values(double, double, g_element &, g_element &,
                           const value_style &, const value_style &, Meas,
                           unc< false >);

void draw_plot_point_values(double, double, g_element &, g_element &,
                           const value_style &, const value_style &, Meas,
                           const value_style &, const value_style &, Meas,
                           Meas);

```

```

    Meas);
void draw_plot_point_values(double, double, g_element &, g_element &,
                           const value_style &, const value_style &, Meas,
                           unc< false >);
void draw_x_major_tick(double, path_element &, path_element &);
void draw_x_minor_tick(double, path_element &, path_element &);
void draw_x_minor_tick(double, path_element &, path_element &);
void place_legend_box();
void place_legend_box();
void size_legend_box();
void size_legend_box();
void transform_point(double &, double &);
void transform_point(double &, double &);

// public data members
text_style a_style_; // Default text style (font etc).
double alpha_; // for calculating confidence intervals (for both X and Y values), default = 0.05;
bool autoscale_check_limits_; // If true then to check autoscale values for infinity, NaN, max, min.
double autoscale_plusminus_; // For uncertain values, allow for text_plusminus ellipses showing 67%, 95% □
and 99% confidence limits.

```

For example, if a max value is 1.2 +or- 0.02, then 1.4 will be used for autoscaling the maximum.

Similarly, if a min value is 1.2 +or- 0.02, then 1.0 will be used for autoscaling the minimum.

```

svg_style axis_style_; // line style for axis.
svg_style box_style_; // style (color ...) of box used to show limit of most data items.
double box_width_; // Width of box used to show limit of most data items.
double epsilon_; // for calculating confidence intervals (for both X and Y values). default= 0.01;
plot_point_style ext_outlier_; // Style of points that are extreme outliers.
bool extreme_outlier_values_on_; // true if values of extreme outlier data are shown.
svg image_; // svg image data (stored so as to avoid rewriting style information constantly).
box_style image_border_; // rectangular border of all image width, color...
bool isNoisyDigit_; // If extra 'noisy' digit is required, default = false;;
svg_style max_whisker_style_; // line style for maximum whisker.
svg_style median_style_; // style (color ...) used to show median.
bool median_values_on_; // true if median of data is shown as well as a line marker.
plot_point_style mild_outlier_; // Style of points that are mild outliers.
svg_style min_whisker_style_; // line style for minimum whisker.
bool outlier_values_on_; // true if outlier values of data are shown.
double plot_bottom_; // SVG coordinate of bottom of plot window.
double plot_left_; // SVG coordinate of left of plot window.
double plot_right_; // SVG coordinate of right of plot window.
double plot_top_; // SVG coordinate of top of plot window.
box_style plot_window_border_; // rectangular border of plot window width, color...
std::string plot_window_clip_; // name of plot window clip (all lines and text is clipped outside).
bool plot_window_on_; // true if a central plot window (rather than the whole image rectangle) to be used.
text_style point_symbols_style_; // text style (font) used for symbol used for data point marking.
int quartile_definition_; // Quartile definition type.
std::vector<svg_boxplot_series> series; // Storing the data points for sorting by value.
text_style series_style_; // font style for name of series.
double text_margin_; // Marginal character space around text items like title, (Use text_margin_* font_size □
to get distance in svg units).
double text_plusminus_;
text_element title_info_; // plot title info.
bool title_on_; // If true then should the plot title (set true if a title requested).
text_style title_style_; // text style (font) used for plot title.
int uncSigDigits_; // significant digits for uncertainty. default = 2;
text_style value_style_; // style used for data point value label.
value_style values_style_; // Data point value marking style.
double whisker_length_; // length of whisker showing outliers.
double x_auto_max_value_; // Value calculated by scale_axis, and is used only if x_autoscale == true.
double x_auto_min_value_; // Value calculated by scale_axis, and is used only if x_autoscale == true.
double x_auto_tick_interval_; // tick major interval calculated by scale_axis.

```

```

int x_auto_ticks_; // Number of ticks calculated by scale_axis.
bool x_autoscale_; // If true then to use any X-axis autoscale values.
axis_line_style x_axis_; // style of X axis line.
text_style x_axis_label_style_; // text style (font) used for Y axis label.
double x_axis_position_; // Intersection of X axis, or not.
bool x_include_zero_; // If true and autoscaled, include zero.
text_element x_label_info_; // X axis label info.
int x_min_ticks_; // If autoscaled, set a minimum number of X ticks.
double x_scale_; // Used by function transform()
double x_shift_; // to go from Cartesian to SVG coordinates.
int x_steps_; // If autoscaled, set any prescaling to decimal 1, 2, 5, 10 etc.
ticks_labels_style x_ticks_; // style of X axis ticks.
double x_tight_; // How much a point is allowed beyond the X minimum or maximum before another tick is used.
text_element x_units_info_; // X axis unit label, For example: "mm".
text_style x_value_label_style_; // text style (font) used for X data point value label.
double y_auto_max_value_; // Value calculated by scale_axis, and is used only if y_autoscale == true.
double y_auto_min_value_; // Value calculated by scale_axis, and is used only if y_autoscale == true.
double y_auto_tick_interval_; // tick major interval calculated by scale_axis.
int y_auto_ticks_; // Number of ticks calculated by scale_axis.
bool y_autoscale_; // If true then to use any y_axis autoscale values.
axis_line_style y_axis_; // style of Y axis line.
text_style y_axis_label_style_; // text style (font) used for Y axis label.
double y_axis_position_; // Intersection of Y axis, or not.
bool y_include_zero_; // If autoscaled, include zero.
text_element y_label_info_; // Y axis label info (used for 2-D only).
int y_min_ticks_; // If autoscaled, set a minimum number of Y ticks.
double y_scale_; // SVG coordinates origin is at top left,.
double y_shift_; // and y coordinates increase DOWN the page!
int y_steps_; // If autoscaled, set any prescaling to decimal 1, 2, 5, 10 etc.
ticks_labels_style y_ticks_; // style of X axis ticks. (Added to permit shared code!)
double y_tight_; // How much a point is allowed beyond the Y minimum or maximum before another tick is used.
text_element y_units_info_; // Y axis unit label (used for 2-D only).
text_style y_value_label_style_; // text style (font) used for Y data point value label.
};


```

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. `bool autoscale();`

Returns: true if to use autoscale values autoscaling for X-axis.

Returns: true if to use autoscale values for X-axis. autoscale() is same as x_autoscale.
 Returns: true if to use autoscale values for X-axis. autoscale() is same as x_autoscale.

4. `svg_boxplot & autoscale(bool b);`

Set true if to use autoscale values for X-axis.

Set whether to use X autoscaled values. Same as x_autoscale, and used by boxplot too.

Set whether to use X autoscaled values. Same as x_autoscale, and used by boxplot too.

5. `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'.

6. `bool autoscale_check_limits();`

Returns: true if to check that values used for autoscaling are within limits.

7. `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'.

Set to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed.

Set to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed.

8. `bool autoscale_check_limits();`

Returns: true if to check that values used for autoscaling are within limits.

Returns: to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed.

Returns: to check that values used for autoscale are within limits. Default is true, but can switch off checks for speed.

9. `svg_boxplot & autoscale_plusminus(double);`

Set how many std_dev or standard deviations to allow for ellipses when autoscaling.

10. `double autoscale_plusminus();`

Returns: How many std_dev or standard deviations allowed for ellipses when autoscaling.

11. `svg_boxplot & autoscale_plusminus(double);`

Set how many std_dev or standard deviations to allow for ellipses when autoscaling.

Set how many std_dev or standard deviation to allow for ellipse when autoscaling. Default is 3 for 99% confidence.

Set how many std_dev or standard deviation to allow for ellipse when autoscaling. Default is 3 for 99% confidence.

12. `double autoscale_plusminus();`

Returns: How many std_dev or standard deviations allowed for ellipses when autoscaling.
 Returns: how many std_dev or standard deviation to allow for ellipse when autoscaling. Default is 3 for 99% confidence.
 Returns: how many std_dev or standard deviation to allow for ellipse when autoscaling. Default is 3 for 99% confidence.

13. `svg_boxplot & axes_on(bool is);`

Set true if to draw **both** x and y axes (note plural axes).

14. `bool axes_on();`

Returns: true if to draw **both** x and y axis on.

15. `svg_boxplot & axes_on(bool is);`

Set true if to draw **both** x and y axes (note plural axes).

If set true, draw **both** x and y axes (note plural axes).

If set true, draw **both** x and y axes (note plural axes).

16. `bool axes_on();`

Returns: true if to draw **both** x and y axis on.

Returns: true if **both** x and y axis on.

Returns: true if **both** x and y axis on.

17. `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.

18. `svg_color axis_color();`

Returns: Color of vertical whisker axis line in box.

19. `svg_boxplot & axis_width(double l);`

Set width of the box, not the border.

Returns: Reference to `svg_boxplot` to make chainable.

20. `double axis_width();`

Returns: Width of the box, not the border.

21. `svg_boxplot & background_border_color(const svg_color & col);`

Set SVG image background border color.

Returns: Reference to `svg_boxplot` to make chainable.

22 `svg_color background_border_color();`

Returns: Color of the border of the background for the SVG image.

23 `svg_boxplot & background_border_width(double w);`

Set plot background border width.

24 `double background_border_width();`

Returns: Plot background border width.

25 `svg_boxplot & background_border_width(double w);`

Set plot background border width.

Set plot background border width.

Set plot background border width.

26 `double background_border_width();`

Returns: Plot background border width.

Returns: plot background border width.

Returns: plot background border width.

27 `svg_boxplot & background_color(const svg_color & col);`

Set SVG image background color.

28 `svg_color background_color();`

Returns: Color of the background for the SVG image.

29 `svg_boxplot & boost_license_on(bool l);`

Set true if the Boost license conditions should be included in the SVG document.

30 `bool boost_license_on();`

Returns: true if the Boost license conditions should be included in the SVG document.

31 `svg_boxplot & boost_license_on(bool l);`

Set true if the Boost license conditions should be included in the SVG document.

Set if the Boost license conditions should be included in the SVG document.

Set if the Boost license conditions should be included in the SVG document.

32 `bool boost_license_on();`

Returns: true if the Boost license conditions should be included in the SVG document.
 Returns: if the Boost license conditions should be included in the SVG document. To set see axis_plot_frame::boost_license_on(bool).
 Returns: if the Boost license conditions should be included in the SVG document. To set see axis_plot_frame::boost_license_on(bool).

33. `svg_boxplot & box_border(const svg_color & color);`

Set color of box border.

Returns: Reference to `svg_boxplot` to make chainable.

34. `svg_color box_border();`

Returns: Color of box border.

35. `svg_boxplot & box_fill(const svg_color & color);`

Set color of box fill (not border).

Returns: Reference to `svg_boxplot` to make chainable.

36. `svg_color box_fill();`

Returns: Color of box fill (not border).

37. `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.

38. `double box_width();`

Returns: width of the box. (Width of the box, not the border).

39. `void calculate_plot_window();`

Calculate the position of the plot window.

40. `void clear_all();`

Clear all the previous information and rebuild the SVG image.

When writing to multiple documents, the contents of the plot may change significantly between. Rather than figuring out what has and has not changed, just erase the contents of the legend, title... in the document and start over.

41. `svg_boxplot & confidence(double);`

Set confidence alpha for display of confidence intervals (default 0.05 for 95%).

42. `double confidence();`

Returns: Confidence alpha for display of confidence intervals (default 0.05 for 95%).

43. `svg_boxplot & confidence(double);`

Set confidence alpha for display of confidence intervals (default 0.05 for 95%).

Set alpha for displaying confidence intervals. Default is 0.05 for 95% confidence.

Set alpha for displaying confidence intervals. Default is 0.05 for 95% confidence.

44. `double confidence();`

Returns: Confidence alpha for display of confidence intervals (default 0.05 for 95%).

Returns: alpha for displaying confidence intervals. Default is 0.05.

Returns: alpha for displaying confidence intervals. Default is 0.05.

45. `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.

46. `int coord_precision();`

Returns: precision of SVG coordinates in decimal digits.

47. `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.

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.

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.

48. `int coord_precision();`

Returns: precision of SVG coordinates in decimal digits.

Returns: precision of SVG coordinates in decimal digits.

Returns: precision of SVG coordinates in decimal digits.

49. `svg_boxplot & copyright_date(const std::string d);`

Writes copyright date to the SVG document. and as metadata:<meta name="date" content="2007" />

50. `const std::string copyright_date();`

Returns: SVG document copyright_date.

51. `svg_boxplot & copyright_date(const std::string d);`

Writes copyright date to the SVG document. and as metadata:<meta name="date" content="2007" />

Writes copyright date to the document. and asmetadata <meta name="date" content="2007" />

Writes copyright date to the document. and asmetadata <meta name="date" content="2007" />

52. `const std::string copyright_date();`

Returns: SVG document copyright_date.

Returns: copyright_date.

Returns: copyright_date.

53. `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" />

54. `const std::string copyright_holder();`

Returns: SVG document copyright holder.

55. `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" />

Writes copyright_holder to the document (as<!-- SVG Plot Copyright Paul A. Bristow 2007 -->) and asmetadata: <meta name="copyright" content="Paul A. Bristow" /meta>

Writes copyright_holder to the document (for header as<!-- SVG Plot Copyright Paul A. Bristow 2007 -->) and asmetadata: <meta name="copyright" content="Paul A. Bristow" /meta>

56. `const std::string copyright_holder();`

Returns: SVG document copyright holder.

Returns: copyright holder.

Returns: copyright holder.

57. `svg_boxplot & data_lines_width(double width);`

Set the width of lines joining data points.

58. `double data_lines_width();`

Returns: the width of lines joining data points.

59. `svg_boxplot & data_lines_width(double width);`

Set the width of lines joining data points.

Set the width of lines joining data points.

Set the width of lines joining data points.

60. `double data_lines_width();`

Returns: the width of lines joining data points.
Returns: the width of lines joining data points.
Returns: the width of lines joining data points.

61. `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. Sadly this has the most unwelcome effect of terminally confusing the Visual Studio Intellisense, and sometimes the debugger as well :-(

62. `const svg_boxplot & derived() const;`

const version of derived()

63. `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. Sadly this has the most unwelcome effect of terminally confusing the Visual Studio Intellisense, and sometimes the debugger as well :-(

64. `const svg_boxplot & derived() const;`

const version of derived()

65. `svg_boxplot & description(const std::string d);`

Writes description to the document for header as.

<desc> My Description </desc>.

66. `const std::string & description();`

Returns: Description of the document for header as<desc> My description </desc>.

67. `svg_boxplot & description(const std::string d);`

Writes description to the document for header as.

<desc> My Description </desc>.

Writes description to the SVG document for header, for example: <desc> My Data </desc>

Writes description to the document for header, for example: <desc> My Data </desc>

68. `const std::string & description();`

Returns: Description of the document for header as<desc> My description </desc>.
Returns: Description of the document for title as<desc> ... </desc>
Returns: description of the document for header as<desc> ... </desc>

69. `svg_boxplot & document_title(const std::string d);`

Set document title to the document for header as.

`<title> My Title </title>.`

70. `std::string document_title();`

Returns: Document title to the document for header as`<title> My Title </title>.`

71. `svg_boxplot & document_title(const std::string d);`

Set document title to the document for header as.

`<title> My Title </title>.`

Write document title to the SVG document for title as`<title> My Title </title>`

Write document title to the SVG document for header as`<title> My Title </title>`

72. `std::string document_title();`

Returns: Document title to the document for header as`<title> My Title </title>.`

Returns: Get document title to the document for title as`<title> My Title </title>`

Returns: Get document title to the document for header as`<title> My Title </title>`

73. `void draw_box(double q1, double q3, double x, double width,
const svg_style & box_styl);`

Draw the box border and any fill color.

74. `void draw_boxplot(svg_boxplot_series & a_series, double x_offset);`

Draw a whole boxplot, box, median line, axis whiskers, and outliers.

75. `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.

76. `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.

Annotate plot with a line from SVG Coordinates X1, Y1 to X2, Y2.

Default color black.

Annotate plot with a line from SVG Coordinates X1, Y1 to X2, Y2.

Default color black.

77.

```
void draw_median(double median, double x_offset, double box_width,
                 const svg_style & median_style,
                 const value_style & values_style);
```

Draw the median of the data series line within the box, and optionally the median value.

78.

```
svg_boxplot &
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.

79.

```
svg_boxplot &
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.

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.

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.

80.

```
void draw_outliers(double x, const std::vector< double > & outliers,
                    const std::vector< double > & extreme_outliers,
                    const plot_point_style & mild_style,
                    const plot_point_style & extreme_style,
                    const value_style & values_style);
```

Draw any outliers, both mild and extreme.

81.

```
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.

82

```
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.

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.

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.

83.

```
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.

84.

```
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.

Annotate plot with a line from user's Cartesian Coordinates X1, Y1 to X2, Y2.

For example, -10, -10, +10, +10, Default color 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.

85.

```
void draw_title();
```

Update title_info_ with position.

Assumes align = center_align. Note: center_align will ensure that will center correctly even if original string is long because contains Unicode like because the browser render engine does the centering.

86.

```
void draw_whiskers(double min, double max, double length, double x,
                    const svg_style & min_whisker,
                    const svg_style & max_whisker,
                    const svg_style & axis_whisker);
```

Draw the whiskers for the boxplot.

87.

```
void draw_x_axis();
```

Draw the horizontal X-axis line.

88. `void draw_x_axis_label();`

Draw an axis label (and optional units) for example "length (km)".

89. `void draw_x_major_tick(double i, path_element & tick_path,
text_element & series_info);`

Draw X axis major tick, and optional boxplot label.

90. `void draw_y_axis();`

Draw the vertical Y-axis line.

91. `void draw_y_axis_label();`

Draw vertical y_axis label, and optional y units.

92. `void draw_y_major_tick(double value, path_element & tick_path,
path_element & grid_path);`

Draw a Y axis major tick, tick value labels & horizontal grid.

93. `void draw_y_minor_tick(double value, path_element & tick_path,
path_element & grid_path);`

Draw a Y-axis minor tick and optional grid.

94. `svg_boxplot & extreme_outlier_color(const svg_color & color);`

Color of extreme outlier.

Returns: Reference to `svg_boxplot` to make chainable.

95. `svg_color extreme_outlier_color();`

Returns: Color of extreme outlier.

96. `svg_boxplot & extreme_outlier_fill(const svg_color & color);`

Set color of extreme outlier fill.

Returns: Reference to `svg_boxplot` to make chainable.

97. `svg_color extreme_outlier_fill();`

Returns: Color of extreme outlier fill.

98. `svg_boxplot & extreme_outlier_shape(point_shape shape);`

Set shape of extreme outlier marker.

Returns: Reference to `svg_boxplot` to make chainable.

9. `point_shape extreme_outlier_shape();`

Returns: Shape of extreme outlier marker.

10 `svg_boxplot & extreme_outlier_size(int size);`

Set size of extreme outlier marker.

Returns: Reference to `svg_boxplot` to make chainable.

11 `int extreme_outlier_size();`

Returns: Size of extreme outlier marker.

12 `svg_boxplot & extreme_outlier_values_on(bool cmd);`

Set true to show extreme outlier values.

Returns: Reference to `svg_boxplot` to make chainable.

13 `bool extreme_outlier_values_on();`

Returns: true if to show extreme outlier value(s).

14 `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.

15 `double image_border_margin();`

Returns: the margin around the plot window border (svg units, default pixels).

16 `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.

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.

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.

17 `double image_border_margin();`

Returns: the margin around the plot window border (svg units, default pixels).

Returns: the margin around the plot window border (svg units, default pixels).

Returns: the margin around the plot window border (svg units, default pixels).

18 `svg_boxplot & image_border_width(double w);`

Set the svg image border width (svg units, default pixels).

19 `double image_border_width();`

Returns: the svg image border width (svg units, default pixels).

10 `svg_boxplot & image_border_width(double w);`

Set the svg image border width (svg units, default pixels).

Set the svg image border width (svg units, default pixels).

Set the svg image border width (svg units, default pixels).

11 `double image_border_width();`

Returns: the svg image border width (svg units, default pixels).

Returns: the svg image border width (svg units, default pixels).

Returns: the svg image border width (svg units, default pixels).

12 `unsigned int image_x_size();`

Obselete - deprecated use x_size()

13 `svg_boxplot & image_x_size(unsigned int i);`

Obselete - deprecated - use x_size().

Set SVG image X-axis size (SVG units, default pixels).

14 `int image_x_size();`

Obselete - deprecated use x_size()

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

Returns: SVG image X-axis size as horizontal width (SVG units, default pixels).

15 `svg_boxplot & image_x_size(int i);`

Obselete - deprecated - use x_size().

Set SVG image X-axis size (SVG units, default pixels).

16 `unsigned int image_y_size();`

Obselete - deprecated - use y_size()

17 `svg_boxplot & image_y_size(unsigned int i);`

Obselete - deprecated - use y_size()

Set SVG image Y-axis size (SVG units, default pixels).

18 `int image_y_size();`

Obselete - deprecated - use `y_size()`

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

Returns: SVG image Y-axis size as vertical height (SVG units, default pixels).

19 `svg_boxplot & image_y_size(int i);`

Obselete - deprecated - use `y_size()`

Set SVG image Y-axis size (SVG units, default pixels).

20 `svg_boxplot & legend_background_color(const svg_color & col);`

Set the background fill color of the legend box.

21 `svg_color legend_background_color();`

Returns: the background fill color of the legend box.

22 `svg_boxplot & legend_background_color(const svg_color & col);`

Set the background fill color of the legend box.

Set the background fill color of the legend box.//

Set the background fill color of the legend box.

23 `svg_color legend_background_color();`

Returns: the background fill color of the legend box.

Returns: the background fill color of the legend box.

Returns: the background fill color of the legend box.

24 `svg_boxplot & legend_border_color(const svg_color & col);`

Set the border stroke color of the legend box.

25 `svg_color legend_border_color();`

Returns: the border stroke color of the legend box.

26 `svg_boxplot & legend_border_color(const svg_color & col);`

Set the border stroke color of the legend box.

Set the border stroke color of the legend box.

Set the border stroke color of the legend box.

12 `svg_color legend_border_color();`

Returns: the border stroke color of the legend box.
 Returns: the border stroke color of the legend box.
 Returns: the border stroke color of the legend box.

13 `const std::pair< double, double > legend_bottom_right();`

Returns: SVG coordinate (default pixels) of bottom right of legend box.

14 `const std::pair< double, double > legend_bottom_right();`

Returns: SVG coordinate (default pixels) of bottom right of legend box.
 Returns: svg coordinate (default pixels) of Bottom right of legend box.
 Returns: svg coordinate (default pixels) of Bottom right of legend box.

15 `bool legend_box_fill_on();`

Returns: true if legend box has a background fill color.

16 `bool legend_box_fill_on();`

Returns: true if legend box has a background fill color.
 Returns: true if legend box has a background fill color.
 Returns: true if legend box has a background fill color.

17 `svg_boxplot & legend_color(const svg_color & col);`

Set the color of the title of the legend.

18 `svg_color legend_color();`

Returns: the color of the title of the legend.

19 `svg_boxplot & legend_color(const svg_color & col);`

Set the color of the title of the legend.

Set the color of the title of the legend.

Set the color of the title of the legend.

20 `svg_color legend_color();`

Returns: the color of the title of the legend.
 Returns: the color of the title of the legend.
 Returns: the color of the title of the legend.

21 `svg_boxplot & legend_font_family(const std::string & family);`

Set the font family for the legend title.

17 `const std::string & legend_font_family();`

Returns: the font family for the legend title.

18 `svg_boxplot & legend_font_family(const std::string & family);`

Set the font family for the legend text.

Set the font family for the legend title.

Set the font family for the legend title.

19 `const std::string & legend_font_family();`

Returns: the font family for the legend title.

Returns: the font family for the legend title.

Returns: the font family for the legend title.

20 `svg_boxplot & legend_font_size(int size);`

Set Font size for **both** the legend title and legend text (svg units, default pixels).

Returns: Font size for the legend title and legend text.

21 `int legend_font_size();`



Note

If function .legend_font_size(size) has been used to set both title and text font sizes, then these will be the same (unless altered by a call of .legend_text_font_size(size)).

Returns: Font size for the legend title (svg units, default pixels).

Returns: Font size for the legend title (svg units, default pixels).

22 `svg_boxplot & legend_font_weight(const std::string & weight);`

Set the font weight for the legend title.

23 `const std::string & legend_font_weight();`

Returns: Font weight for the legend title.

24 `svg_boxplot & legend_font_weight(const std::string & weight);`

Set the font weight for the legend title.

Set the font weight for the legend title.

Set the font weight for the legend title.

25 `const std::string & legend_font_weight();`

Returns: Font weight for the legend text.

Returns: the font weight for the legend title.
 Returns: the font weight for the legend title.

16 `svg_boxplot & legend_header_font_size(int size);`

Set legend header font size (svg units, default pixels).

Set legend header font size (svg units, default pixels).

17 `int legend_header_font_size();`

Returns: legend header font size (svg units, default pixels).
 Returns: legend header font size (svg units, default pixels).

18 `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.

19 `bool legend_lines();`

Returns: true if legend should include samples of the lines joining data points.

20 `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.

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.

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.

21 `bool legend_lines();`

Returns: true if legend should include samples of the lines joining data points.
 Returns: true if legend should include samples of the lines joining data points.
 Returns: true if legend should include samples of the lines joining data points.

22 `svg_boxplot & legend_on(bool cmd);`

Set true if a legend is wanted.

23 `bool legend_on();`

Returns: true if a legend is wanted.

14 `svg_boxplot & legend_on(bool cmd);`

Set true if a legend is wanted.

Set true if a legend is wanted.

Set true if a legend is wanted.

15 `bool legend_on();`

Returns: true if a legend is wanted.

Returns: true if a legend is wanted.

Returns: true if a legend is wanted.

16 `bool legend_outside();`

Returns: if the legend should be outside the plot area.

17 `bool legend_outside();`

Returns: if the legend should be outside the plot area.

Returns: if the legend should be outside the plot area.

Returns: if the legend should be outside the plot area.

18 `svg_boxplot & legend_place(legend_places l);`

Set the position of the legend,.

See Also:

`boost::svg::legend_places`

19 `legend_places legend_place();`

See Also:

`boost::svg::legend_places`

Returns: the position of the legend,

10 `svg_boxplot & legend_place(legend_places l);`

Set the position of the legend,.

See Also:

`boost::svg::legend_places`

Set the position of the legend,

See Also:

`boost::svg::legend_places`

Set the position of the legend,

See Also:

boost::svg::legend_places

11 `legend_places legend_place();`

See Also:

boost::svg::legend_places

See Also:

boost::svg::legend_places

See Also:

boost::svg::legend_places

Returns: the position of the legend,
 Returns: the position of the legend,
 Returns: the position of the legend,

12 `svg_boxplot & legend_text_font_size(int size);`

Set the font size for the legend title (svg units, default pixels).

Returns: Font size for the legend text.

13 `int legend_text_font_size();`

Returns: Font size for the legend text (svg units, default pixels).
 Returns: the font size for the legend title (svg units, default pixels).

14 `svg_boxplot & legend_text_font_weight(const std::string & weight);`

Set the font weight for the legend text. Example: my_plot.legend_text_font_size(20);

15 `const std::string & legend_text_font_weight();`

Returns: the font weight for the legend text.

16 `svg_boxplot & legend_title(const std::string title);`

Set the title for the legend.

17 `const std::string legend_title();`

Returns: Title for the legend.

18 `svg_boxplot & legend_title(const std::string title);`

Set the title for the legend.

Set the title for the legend.

Set the title for the legend.

19 `const std::string legend_title();`

Returns: Title for the legend.
 Returns: the title for the legend.
 Returns: the title for the legend.

20 `svg_boxplot & legend_title_font_size(unsigned int size);`

Set the font size for the legend title (svg units, default pixels).
 Returns: Font family for the legend title.

21 `unsigned int legend_title_font_size();`

Returns: Font size for the legend title (svg units, default pixels).

22 `svg_boxplot & legend_title_font_size(int size);`

Set the font size for the legend title (svg units, default pixels).
 Returns: Font size for the legend title.

23 `int legend_title_font_size();`

Returns: Font size for the legend title (svg units, default pixels).
 Returns: the font size for the legend title (svg units, default pixels).
 Returns: the font size for the legend title (svg units, default pixels).

24 `svg_boxplot & legend_title_font_weight(const std::string & weight);`

Set the font weight for the legend title.

Set the font weight for the legend title. Example: `my_plot.legend_title_font_size(10);`

25 `const std::string & legend_title_font_weight();`

Returns: Font weight for the legend title.
 Returns: the font weight for the legend title.

26 `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).

27 `const std::pair< double, double > legend_top_left();`

Returns: SVG coordinate (default pixels) of top left of legend box.

28 `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).

Set position of top left of legend box (svg coordinates, default pixels). Bottom right is controlled by contents, so the user cannot set it.

Set position of top left of legend box (svg coordinates, default pixels). Bottom right is controlled by contents, so the user cannot set it.

19 `const std::pair< double, double > legend_top_left();`

Returns: SVG coordinate (default pixels) of top left of legend box.

Returns: svg coordinate (default pixels) of top left of legend box.

Returns: svg coordinate (default pixels) of top left of legend box.

20 `svg_boxplot & legend_width(double width);`

Set the width for the legend box.

21 `double legend_width();`

Returns: Width for the legend box.

22 `svg_boxplot & legend_width(double width);`

Set the width for the legend box.

Set the width for the legend.

Set the width for the legend.

23 `double legend_width();`

Returns: Width for the legend box.

Returns: the width for the legend.

Returns: the width for the legend.

24 `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.

25 `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.

Set license conditions for reproduction, attribution, commercial use, and derivative works, usually "permits", "requires", or "prohibits", and set license_on == true.

Set license conditions for reproduction, attribution, commercial use, and derivative works, usually "permits", "requires", or "prohibits", and set license_on == true.

16 `const std::string license_attribution();`

Returns: SVG document attribution license conditions, usually "permits", "requires", or "prohibits".

17 `const std::string license_attribution();`

Returns: SVG document attribution license conditions, usually "permits", "requires", or "prohibits".

Returns: attribution license conditions, usually "permits", "requires", or "prohibits".

Returns: attribution license conditions, usually "permits", "requires", or "prohibits".

18 `const std::string license_commercialuse();`

Returns: SVG document commercial use license conditions, usually "permits", "requires", or "prohibits".

19 `const std::string license_commercialuse();`

Returns: SVG document commercial use license conditions, usually "permits", "requires", or "prohibits".

Returns: commercial use license conditions, usually "permits", "requires", or "prohibits".

Returns: commercial use license conditions, usually "permits", "requires", or "prohibits".

20 `const std::string license_distribution();`

Returns: SVG document distribution license conditions, usually "permits", "requires", or "prohibits".

21 `const std::string license_distribution();`

Returns: SVG document distribution license conditions, usually "permits", "requires", or "prohibits".

Returns: distribution license conditions, usually "permits", "requires", or "prohibits".

Returns: distribution license conditions, usually "permits", "requires", or "prohibits".

22 `svg_boxplot & license_on(bool l);`

Set if license conditions should be included in the SVG document.

23 `bool license_on();`

Returns: true if license conditions should be included in the SVG document.

24 `svg_boxplot & license_on(bool l);`

Set if license conditions should be included in the SVG document.

Set if license conditions should be included in the SVG document.

See Also:

`axis_plot_frame::license`

Set if license conditions should be included in the SVG document.

See Also:

`axis_plot_frame::license`

15 `bool license_on();`

See Also:

`axis_plot_frame::license`

See Also:

`axis_plot_frame::license`

Returns: true if license conditions should be included in the SVG document.

Returns: true if license conditions should be included in the SVG document.

Returns: true if license conditions should be included in the SVG document.

16 `const std::string license_reproduction();`

Returns: SVG document reproduction license conditions, usually "permits", "requires", or "prohibits".

17 `const std::string license_reproduction();`

Returns: SVG document reproduction license conditions, usually "permits", "requires", or "prohibits".

Returns: reproduction license conditions, usually "permits", "requires", or "prohibits".

Returns: reproduction license conditions, usually "permits", "requires", or "prohibits".

18 `svg_boxplot & limit_color(const svg_color &);`

Set the color for 'at limit' point stroke color.

19 `svg_color limit_color();`

Returns: the color for the 'at limit' point stroke color.

20 `svg_boxplot & limit_color(const svg_color &);`

Set the color for 'at limit' point stroke color.

Set the color for 'at limit' point stroke color.

Set the color for 'at limit' point stroke color.

21 `svg_color limit_color();`

Returns: the color for the 'at limit' point stroke color.

Returns: the color for the 'at limit' point stroke color.

Returns: the color for the 'at limit' point stroke color.

22 `svg_boxplot & limit_fill_color(const svg_color &);`

Set the color for 'at limit' point fill color.

23 `svg_color limit_fill_color();`

Returns: the color for the 'at limit' point fill color.

24 `svg_boxplot & limit_fill_color(const svg_color &);`

Set the color for 'at limit' point fill color.

Set the color for 'at limit' point fill color.

Set the color for 'at limit' point fill color.

25 `svg_color limit_fill_color();`

Returns: the color for the 'at limit' point fill color.

Returns: the color for the 'at limit' point fill color.

Returns: the color for the 'at limit' point fill color.

26 `svg_boxplot & median_color(const svg_color & color);`

Set color of median line in box.

Returns: Reference to `svg_boxplot` to make chainable.

27 `svg_color median_color();`

Returns: color of median line in box.

28 `svg_boxplot & median_values_on(bool cmd);`

Set true to show median value(s).

Returns: Reference to `svg_boxplot` to make chainable.

29 `bool median_values_on();`

Returns: true if to show median value(s).

30 `svg_boxplot & median_width(double l);`

Set width of the box (not the border).

Returns: Reference to `svg_boxplot` to make chainable.

31 `double median_width();`

Returns: width of the box (not the border).

32 `svg_boxplot & one_sd_color(const svg_color &);`

Set the color for the one standard deviation (~67% confidence) ellipse fill.

33 `svg_color one_sd_color();`

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

24 `svg_boxplot & one_sd_color(const svg_color &);`

Set the color for the one standard deviation (~67% confidence) ellipse fill.

Set the color for the one standard deviation (~67% confidence) ellipse fill.

Set the color for the one standard deviation (~67% confidence) ellipse fill.

25 `svg_color one_sd_color();`

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

Returns: Color for the one standard deviation (~67% confidence) ellipse fill.

26 `svg_boxplot & outlier_color(const svg_color & color);`

Set color of outlier.

Returns: Reference to `svg_boxplot` to make chainable.

27 `svg_color outlier_color();`

Returns: Color of mild outlier.

28 `svg_boxplot & outlier_fill(const svg_color & color);`

Set color of mild outlier fill.

Returns: Reference to `svg_boxplot` to make chainable.

29 `svg_color outlier_fill();`

Returns: Color of outlier fill.

30 `svg_boxplot & outlier_shape(point_shape shape);`

Set shape of outlier marker.

Returns: Reference to `svg_boxplot` to make chainable.

31 `point_shape outlier_shape();`

Returns: Outlier marker shape.

32 `svg_boxplot & outlier_size(int size);`

Set size of outlier marker.

Returns: Reference to `svg_boxplot` to make chainable.

33 `int outlier_size();`

Returns: Size of outlier marker.

24 `svg_boxplot & outlier_style(plot_point_style & os);`

Set outlier style.

Returns: Reference to `svg_boxplot` to make chainable.

25 `plot_point_style & outlier_style();`

<
Returns: Outlier_style.

26 `svg_boxplot & outlier_values_on(bool cmd);`

Set true to show mild outlier values.

Returns: Reference to `svg_boxplot` to make chainable.

27 `bool outlier_values_on();`

Returns: true if to show mild outlier value(s).

28 `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.

29 `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.

20 `svg_boxplot & plot_background_color(const svg_color & col);`

Set plot window background color.

Returns: Reference to `svg_boxplot` to make chainable.

21 `svg_color plot_background_color();`

Returns: Color of the background for the plot.

22 `svg_boxplot & plot_border_color(const svg_color & col);`

Set plot window border color.

Returns: Reference to `svg_boxplot` to make chainable.

23 `svg_color plot_border_color();`

Returns: Color of the border of the background for the plot.

24 `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.

25 `double plot_border_width();`

Returns: the width for the plot window border (svg units, default pixels).

26 `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.

Set the width for the plot window border (svg units, default pixels).

Set the width for the plot window border (svg units, default pixels).

27 `double plot_border_width();`

Returns: the width for the plot window border (svg units, default pixels).
 Returns: the width for the plot window border (svg units, default pixels).
 Returns: the width for the plot window border (svg units, default pixels).

28 `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).

29 `bool plot_window_on();`

Returns: true if a plot window is wanted (or false if the whole image is to be used).

20 `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).

Set true if a plot window is wanted (or false if the whole image is to be used).

Set true if a plot window is wanted (or false if the whole image is to be used).

21 `bool plot_window_on();`

Returns: true if a plot window is wanted (or false if the whole image is to be used).

Returns: true if a plot window is wanted (or false if the whole image is to be used).

Returns: true if a plot window is wanted (or false if the whole image is to be used).

22 `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.

23 `std::pair< double, double > plot_window_x();`

Returns: both the left and right (X axis) of the plot window.

24 `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.

Set the minimum and maximum (cartesian data units) for the plot window X axis. This is normally calculated from other plot values.

Set the minimum and maximum (cartesian data units) for the plot window X axis. This is normally calculated from other plot values.

25 `std::pair< double, double > plot_window_x();`

Returns: both the left and right (X axis) of the plot window.

Returns: both the left and right (X axis) of the plot window.

Returns: both the left and right (X axis) of the plot window.

26 `double plot_window_x_left();`

Returns: left of the plot window.

27 `double plot_window_x_left();`

Returns: left of the plot window.

Returns: left of the plot window.

Returns: left of the plot window.

28 `double plot_window_x_right();`

Returns: right of the plot window.

29 `double plot_window_x_right();`

Returns: right of the plot window.

Returns: right of the plot window.

Returns: right of the plot window.

30 `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.

31 `std::pair< double, double > plot_window_y();`

Returns: both the top and bottom (Y axis) of the plot window.

32 `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.

Set the minimum and maximum (cartesian data units) for the plot window Y axis. This is normally calculated from other plot values.

Set the minimum and maximum (cartesian data units) for the plot window Y axis. This is normally calculated from other plot values.

33 `std::pair< double, double > plot_window_y();`

Returns: both the top and bottom (Y axis) of the plot window.

Returns: both the top and bottom (Y axis) of the plot window.

Returns: both the top and bottom (Y axis) of the plot window.

34 `double plot_window_y_bottom();`

Returns: top of the plot window.

35 `double plot_window_y_bottom();`

Returns: top of the plot window.

Returns: Top of the plot window.

Returns: Top of the plot window.

26 `double plot_window_y_top();`

Returns: top of the plot window.

27 `double plot_window_y_top();`

Returns: top of the plot window.

Returns: top of the plot window.

Returns: top of the plot window.

28 `svg_boxplot & quartile_definition(int def);`

Set definition of quartile.

Returns: Reference to `svg_boxplot` to make chainable.

29 `int quartile_definition();`

Returns: Definition # of quartile.

30 `svg_boxplot & size(int x, int y);`

Set SVG image size (SVG units, default pixels).

Set SVG image size (SVG units, default pixels).

31 `std::pair< double, double > size();`

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

Returns: SVG image size, both horizontal width and vertical height (SVG units, default pixels).

32 `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.

33 `svg_boxplot & three_sd_color(const svg_color &);`

Set the color for three standard deviation (~99% confidence) ellipse fill.

34 `svg_color three_sd_color();`

Returns: Color for three standard deviation (~99% confidence) ellipse fill.

35 `svg_boxplot & three_sd_color(const svg_color &);`

Set the color for three standard deviation (~99% confidence) ellipse fill.

Set the color for three standard deviation (~99% confidence) ellipse fill.

Set the color for three standard deviation (~99% confidence) ellipse fill.

26 `svg_color three_sd_color();`

Returns: Color for three standard deviation (~99% confidence) ellipse fill.
 Returns: Color for three standard deviation (~99% confidence) ellipse fill.
 Returns: Color for three standard deviation (~99% confidence) ellipse fill.

27 `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, for example:

`my_plot.title("Plot of © function (°C)");`

Unicode symbols are at <http://unicode.org/charts/symbols.html>.

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 © function (°C)");`

Unicode symbols are at <http://unicode.org/charts/symbols.html>.



Note

Only set plot title once.

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 © function (°C)");`

Unicode symbols are at <http://unicode.org/charts/symbols.html>.



Note

Only set plot title once.

28 `svg_boxplot & title(const std::string & str);`

Set title text for plot.

Returns: Reference to `svg_boxplot` to make chainable.

29 `std::string title();`

Returns: title of the plot.

20 `svg_boxplot & title_color(const svg_color & col);`

Set boxplot title color.

Returns: Reference to `svg_boxplot` to make chainable.

21 `svg_color title_color();`

Returns: color of the title.

22 `svg_boxplot & title_font_alignment(align_style alignment);`

Set the alignment for the title.

23 `align_style title_font_alignment();`

Returns: the alignment for the title.

24 `svg_boxplot & title_font_alignment(align_style alignment);`

Set the alignment for the title.

Set the alignment for the title.

Set the alignment for the title.

25 `align_style title_font_alignment();`

Returns: the alignment for the title.

Returns: the alignment for the title.

Returns: the alignment for the title.

26 `svg_boxplot & title_font_decoration(const std::string & decoration);`

Set the font decoration for the title (default normal, or underline, overline or strike-thru).

27 `const std::string & title_font_decoration();`

Returns: Font decoration for the title (default normal, or underline, overline or strike-thru).

28 `svg_boxplot & title_font_decoration(const std::string & decoration);`

Set the font decoration for the title (default normal, or underline, overline or strike-thru).

Set the font decoration for the title (default normal, or underline, overline or strike-thru).

Set the font decoration for the title (default normal, or underline, overline or strike-thru).

29 `const std::string & title_font_decoration();`

Returns: Font decoration for the title (default normal, or underline, overline or strike-thru).

Returns: the font decoration for the title (default normal, or underline, overline or strike-thru).

Returns: the font decoration for the title (default normal, or underline, overline or strike-thru).

30 `svg_boxplot & title_font_family(const std::string & family);`

Set the font family for the title (for example: .title_font_family("Lucida Sans Unicode");).

21 `const std::string & title_font_family();`

Returns: Font family for the title.

22 `svg_boxplot & title_font_family(const std::string & family);`

Set the font family for the title (for example: .title_font_family("Lucida Sans Unicode");).

Set the font family for the title (for example: .title_font_family("Lucida Sans Unicode");).

Set the font family for the title (for example: .title_font_family("Lucida Sans Unicode");).

23 `const std::string & title_font_family();`

Returns: Font family for the title.

Returns: the font family for the title

Returns: the font family for the title

24 `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...).

25 `int title_font_rotation();`

Returns: the rotation for the title font (degrees).

26 `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...).

Set the rotation for the title font (degrees, 0 to 360).

Set the rotation for the title font (degrees, 0 to 360).

27 `int title_font_rotation();`

Returns: the rotation for the title font (degrees).

Returns: the rotation for the title font (degrees).

Returns: the rotation for the title font (degrees).

28 `svg_boxplot & title_font_size(unsigned int i);`

Sets the font size for the title (SVG units, default pixels).

Sets the font size for the title (svg units, default pixels).

29 `unsigned int title_font_size();`

Returns: Font size for the title (SVG units, default pixels).

30 `svg_boxplot & title_font_size(int i);`

Sets the font size for the title (SVG units, default pixels).

Sets the font size for the title (svg units, default pixels).

21 `int title_font_size();`

return Font size for the title (SVG units, default pixels). Example: /code std::cout << my_plot.title_font_size() ...

Returns: the font size for the title (svg units, default pixels).

Returns: the font size for the title (svg units, default pixels).

22 `svg_boxplot & title_font_stretch(const std::string & stretch);`

Set the font stretch for the title (default normal), wider or narrow.

23 `const std::string & title_font_stretch();`

Returns: Font stretch for the title.

24 `svg_boxplot & title_font_stretch(const std::string & stretch);`

Set the font stretch for the title (default normal), wider or narrow.

Set the font stretch for the title (default normal), wider or narrow.

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.

Returns: the font stretch for the title.

Returns: the font stretch for the title.

26 `svg_boxplot & title_font_style(const std::string & style);`

Set the font style for the title (default normal).

27 `const std::string & title_font_style();`

Returns: Font style for the title (default normal).

28 `svg_boxplot & title_font_style(const std::string & style);`

Set the font style for the title (default normal).

Set the font style for the title (default normal).

Set the font style for the title (default normal).

29 `const std::string & title_font_style();`

Returns: Font style for the title (default normal).

Returns: the font style for the title (default normal).

Returns: the font style for the title (default normal).

30 `svg_boxplot & title_font_weight(const std::string & weight);`

Set the font weight for the title (default normal).

31 `const std::string & title_font_weight();`

Returns: Font weight for the title.

32 `svg_boxplot & title_font_weight(const std::string & weight);`

Set the font weight for the title (default normal).

Set the font weight for the title (default normal).

Set the font weight for the title (default normal).

33 `const std::string & title_font_weight();`

Returns: Font weight for the title.

Returns: the font weight for the title.

Returns: the font weight for the title.

34 `bool title_on();`

If true, will show a title for the plot.

If true, will show a title for the plot.

Returns: true if will show a title for the plot.

35 `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.

36 `svg_boxplot & title_size(unsigned int size);`

Set font size for title text.

Returns: Reference to `svg_boxplot` to make chainable.

37 `text_style & title_style();`

Returns: All style info for the title, font, famil, size ... (SVG units, default pixels).

Returns: Font size for the title (svg units, default pixels). Example: `std::cout << "title style " << my_2d_plot.title_style() << std::endl;` Outputs: title style text_style(10, "Lucida Sans Unicode", "", "normal", "", "", 900)

38 `svg_boxplot & title_text_length(double);`

Sets the text_length for the title (SVG units, default pixels).

Sets the the estimated text length for the title (svg units, default pixels).

39 `double title_text_length();`

return text_length for the title (SVG units, default pixels).

Returns: the estimated text length for the title (svg units, default pixels).

30 `void transform_x(double & x);`

Transform X coordinate from Cartesian to SVG.

31 `void transform_y(double & y);`

Transform Y coordinate from Cartesian to SVG.

32 `svg_boxplot & two_sd_color(const svg_color &);`

Set the color for two standard deviation (~95% confidence) ellipse fill.

33 `svg_color two_sd_color();`

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

34 `svg_boxplot & two_sd_color(const svg_color &);`

Set the color for two standard deviation (~95% confidence) ellipse fill.

Set the color for two standard deviation (~95% confidence) ellipse fill.

Set the color for two standard deviation (~95% confidence) ellipse fill.

35 `svg_color two_sd_color();`

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

Returns: Color for two standard deviation (~95% confidence) ellipse fill.

36 `void update_image();`

Update the entire SVG image.

37 `svg_boxplot & whisker_length(double width);`

Set the length of the whisker.

Returns: Reference to `svg_boxplot` to make chainable.

38 `double whisker_length();`

Get whisker length (width of the box, not the margin).

<

Returns: Length of whisker.

39 `svg_boxplot & write(const std::string & file);`

Write SVG image to file.

40 `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.

41 `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>".

42 `svg_color x_addlimits_color();`

Returns: the color of X confidence limits of value, for example, the color of "<1.23, 1.45>".

43 `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>".

Set the color of X confidence limits value, for example, the color of "<1.23 , 1.34>".

Set the color of X confidence limits value, for example, the color of "<1.23 , 1.34>".

44 `svg_color x_addlimits_color();`

Get the color of X confidence limits of value, for example, the color of "<1.23 , 1.34>"".

Get the color of X confidence limits of value, for example, the color of "<1.23 , 1.34>"".

Returns: the color of X confidence limits of value, for example, the color of "<1.23, 1.45>".

45 `svg_boxplot & x_addlimits_on(bool b);`

Set if to append confidence limits to data point X values near data points markers.

46 `bool x_addlimits_on();`

Returns: true if to append confidence limits estimate to data point X values near data points markers.

47 `svg_boxplot & x_addlimits_on(bool b);`

Set if to append confidence limits to data point X values near data points markers.

Set if to append confidence limits to data point X values near data points markers. (May not be implemented yet).

Set if to append confidence limits to data point X values near data points markers. (May not be implemented yet).

38 `bool x_addlimits_on();`

Returns: true if to append confidence limits estimate to data point X values near data points markers.
 Returns: if to append confidence limits to data point X values near data points markers. (May not be implemented yet).
 Returns: if to append confidence limits to data point X values near data points markers. (May not be implemented yet).

39 `double x_auto_max_value();`

Returns: X-axis maximum value computed by autoscale.

40 `double x_auto_max_value();`

Returns: X-axis maximum value computed by autoscale.
 Returns: the X-axis maximum value computed by autoscale.
 Returns: the X-axis maximum value computed by autoscale.

41 `double x_auto_min_value();`

Returns: X-axis minimum value computed by autoscale.

42 `double x_auto_min_value();`

Returns: X-axis minimum value computed by autoscale.
 Returns: the X-axis minimum value computed by autoscale.
 Returns: the X-axis minimum value computed by autoscale.

43 `double x_auto_tick_interval();`

Returns: the X-axis major tick interval computed by autoscale.

44 `double x_auto_tick_interval();`

Returns: the X-axis major tick interval computed by autoscale.
 Returns: the X-axis major tick interval computed by autoscale.
 Returns: the X-axis major tick interval computed by autoscale.

45 `int x_auto_ticks();`

Returns: the X-axis number of major ticks computed by autoscale.

46 `int x_auto_ticks();`

Returns: the X-axis number of major ticks computed by autoscale.
 Returns: the X-axis number of major ticks computed by autoscale.
 Returns: the X-axis number of major ticks computed by autoscale.

47 `bool x_autoscale();`

Returns: true if to use autoscale value for X-axis.

48 `svg_boxplot & x_autoscale(bool b);`

Set true if to use autoscale values for X-axis.

39 `svg_boxplot & x_autoscale(std::pair< double, double > p);`

autoscale X axis using a pair of doubles.

40 `svg_boxplot & x_autoscale(const T & container);`

<

41 `svg_boxplot & x_autoscale(const T & begin, const T & end);`

<

42 `bool x_autoscale();`

Returns: true if to use autoscale value for X-axis.

Returns: true if to use autoscale value for X-axis.

Returns: true if to use autoscale value for X-axis.

43 `svg_boxplot & x_autoscale(bool b);`

Set true if to use autoscale values for X-axis.

Set true if to use autoscaled values for X-axis.

Set true if to use autoscaled values for X-axis.

Returns: Reference to caller to make chainable.

Returns: Reference to caller to make chainable.

44 `svg_boxplot & x_autoscale(std::pair< double, double > p);`

autoscale X axis using a pair of doubles.

Set to use X min & max pair of double values to autoscale X-axis.

Set to use X min & max pair of double values to autoscale X-axis.

45 `svg_boxplot & x_autoscale(const T & container);`

<

Data series (all values) to use to calculate autoscaled X-axis values.

By default, use these calculated values.

Data series (all values) to use to calculate autoscaled X-axis values.

By default, use these calculated values.

46 `svg_boxplot & x_autoscale(const T & begin, const T & end);`

<

Data series (range accessed using iterators) to use to calculate autoscaled X-axis values.

By default, use these calculated values.

Data series (range accessed using iterators) to use to calculate autoscaled X-axis values.

By default, use these calculated values.

37 `svg_boxplot & x_axis_color(const svg_color & col);`

Set the color of the X-axis line.

38 `svg_color x_axis_color();`

Returns: the color of the X-axis line.

39 `svg_boxplot & x_axis_color(const svg_color & col);`

Set the color of the X-axis line.

Set the color of the X-axis line.

Set the color of the X-axis line.

40 `svg_color x_axis_color();`

Returns: the color of the X-axis line.

Returns: the color of the X-axis line.

Returns: the color of the X-axis line.

41 `svg_boxplot & x_axis_label_color(const svg_color & col);`

Set X axis label color, for example, red.

42 `svg_color x_axis_label_color();`

Returns: X axis label color. X-axis ticks values label style.

43 `svg_boxplot & x_axis_label_color(const svg_color & col);`

Set X axis label color, for example, red.

Set X axis label color.

Set X axis label color.

44 `svg_color x_axis_label_color();`

Returns: X axis label color. X-axis ticks values label style.

Returns: X axis label color.

Returns: X axis label color.

35 `svg_boxplot & x_axis_on(bool is);`

If set true, draw a horizontal X-axis line.

36 `bool x_axis_on();`

Returns: true if will draw a horizontal X-axis line.

37 `svg_boxplot & x_axis_on(bool is);`

If set true, draw a horizontal X-axis line.

If set true, draw a horizontal X-axis line.

If set true, draw a horizontal X-axis line.

38 `bool x_axis_on();`

If true, draw a horizontal X-axis line.

If true, draw a horizontal X-axis line.

Returns: true if will draw a horizontal X-axis line.

39 `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.

40 `double x_axis_position();`

Returns: position of the horizontal X-axis line (on the border).

41 `svg_boxplot & x_axis_vertical(double fraction);`

Set vertical position of X-axis for 1D as fraction of plot window.

42 `bool x_axis_vertical();`

Returns: vertical position of X-axis for 1D as fraction of plot window.

43 `svg_boxplot & x_axis_vertical(double fraction);`

Set vertical position of X-axis for 1D as fraction of plot window.

Set vertical position of X-axis for 1D as fraction of plot window.

Set vertical position of X-axis for 1D as fraction of plot window.

44 `bool x_axis_vertical();`

Returns: vertical position of X-axis for 1D as fraction of plot window.
 Returns: vertical position of X-axis for 1D as fraction of plot window.
 Returns: vertical position of X-axis for 1D as fraction of plot window.

35 `svg_boxplot & x_axis_width(double width);`

Set the width of X-axis lines.

36 `double x_axis_width();`

Returns: the width of X-axis lines.

37 `svg_boxplot & x_axis_width(double width);`

Set the width of X-axis lines.

Set the width of X-axis lines.

Set the width of X-axis lines.

38 `double x_axis_width();`

Returns: the width of X-axis lines.
 Returns: the width of X-axis lines.
 Returns: the width of X-axis lines.

39 `svg_boxplot & x_datetime_color(const svg_color & col);`

Set the color of X date time , for example, the color of text in "".

40 `svg_color x_datetime_color();`

Returns: the color of X date time, for example, the color of text in "".

41 `svg_boxplot & x_datetime_color(const svg_color & col);`

Set the color of X date time , for example, the color of text in "".

Set the color of X point datetime, for example, the color of text in "2004-Jan-1 05:21:33.20".

Set the color of X point datetime, for example, the color of text in "2004-Jan-1 05:21:33.20".

42 `svg_color x_datetime_color();`

Get the color of X point date time, for example, the color of text in "2004-Jan-1 05:21:33.20".

Get the color of X point date time, for example, the color of text in "2004-Jan-1 05:21:33.20".

Returns: the color of X date time, for example, the color of text in "".

43 `svg_boxplot & x_datetime_on(bool b);`

Set true if to append date time to data point X values near data points markers.

34 `bool x_datetime_on();`

Returns: true if to append an date time to data point X values near data points markers.

35 `svg_boxplot & x_datetime_on(bool b);`

Set true if to append date time to data point X values near data points markers.

Set true if to append a datetime to data point X values near data points markers. (May not be implemented yet).

Set true if to append a datetime to data point X values near data points markers. (May not be implemented yet).

36 `bool x_datetime_on();`

Returns: true if to append an date time to data point X values near data points markers.

Returns: true if to append a ID or name to data point X values near data points markers. (May not be implemented yet).

Returns: true if to append a ID or name to data point X values near data points markers. (May not be implemented yet).

37 `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 "\u00A0;", for example, "\u00A0;" rather than ASCII space", ". If 1st char in separator == , then Y values and info will be on a newline below.

38 `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 "\u00A0;", for example, "\u00A0;" rather than ASCII space", ". If 1st char in separator == , then Y values and info will be on a newline below.

Set prefix, separator and suffix for x_style



Note

if you want a space, you must use a Unicode space "\u00A0;", for example, "\u00A0;" rather than just ", ".

Set prefix, separator and suffix for x_style



Note

if you want a space, you must use a Unicode space "\u00A0;", for example, "\u00A0;" rather than just ", ".

39 `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_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)".

Set the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

Set the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

32 `svg_color x_df_color();`

Get the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

Get the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

Returns: the color of X degrees of freedom, for example, the color of 9 in "1.23 +-0.02 (9)".

33 `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.

34 `bool x_df_on();`

Returns: true if to append a degrees of freedom estimate to data point X values near data points markers.

35 `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.

Set true if to append a degrees of freedom estimate to data point X values near data points markers.

Set true if to append a degrees of freedom estimate to data point X values near data points markers.

36 `bool x_df_on();`

Returns: true if to append a degrees of freedom estimate to data point X values near data points markers.

Returns: true if to append a degrees of freedom estimate to data point X values near data points markers. (May not be implemented yet).

Returns: true if to append a degrees of freedom estimate to data point X values near data points markers. (May not be implemented yet).

37 `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".

38 `svg_color x_id_color();`

Returns: the color of X X id or name, for example, the color of text in "my_id".

39 `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".

Set the color of X ID or name, for example, the color of text in "My_id".

Set the color of X ID or name, for example, the color of text in "My_id".

30 `svg_color x_id_color();`

Get the color of X ID or name, for example, the color of text in "My_id".

Get the color of X ID or name, for example, the color of text in "My_id".

Returns: the color of X X id or name, for example, the color of text in "my_id".

31 `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.

32 `bool x_id_on();`

Returns: true if to append an ID or name to data point X values near data points markers.

33 `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.

Set true if to append a id or name to data point X values near data points markers. (May not be implemented yet).

Set true if to append a id or name to data point X values near data points markers. (May not be implemented yet).

34 `bool x_id_on();`

Returns: true if to append an ID or name to data point X values near data points markers.

Returns: true if to append an ID or name to data point X values near data points markers. (May not be implemented yet).

Returns: true if to append an ID or name to data point X values near data points markers. (May not be implemented yet).

35 `std::string x_label();`

Returns: the text to label the X-axis.

Returns: the text to label the X-axis.

Returns: the text to label the X-axis.

36 `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

See Also:

`x_label_units`

Returns: Reference to `svg_boxplot` to make chainable.

37 `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.

38 `svg_color x_label_color();`

Returns: Color of the X axis label.

39 `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"
```

40 `const std::string & x_label_font_family();`

Returns: X tick value label font family.

41 `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"
```

Set X tick value label font family.

Set X tick value label font family.

42 `const std::string & x_label_font_family();`

Returns: X tick value label font family.

Returns: X tick value label font family.

Returns: X tick value label font family.

43 `svg_boxplot & x_label_font_size(unsigned int i);`

Set X axis label font size (svg units, default pixels).

Set X axis label font size (svg units, default pixels).

44 `unsigned int x_label_font_size();`

Returns: X axis label font size (svg units, default pixels).

45 `svg_boxplot & x_label_font_size(int i);`

Set X axis label font size (svg units, default pixels).

Set X axis label font size (svg units, default pixels).

46 `int x_label_font_size();`

Returns: X axis label font size (svg units, default pixels).

Returns: X axis label font size (svg units, default pixels).

Returns: X axis label font size (svg units, default pixels).

47 `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.

48 `bool x_label_on();`

Returns: label for the X axis.

49 `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.

50 `std::string x_label_text();`

Returns: Text of label for X axis.

51 `svg_boxplot & x_label_units(const std::string & str);`

Set the text to add units to the X-axis label.

52 `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.

53 `svg_boxplot & x_label_units(const std::string & str);`

Set the text to add units to the X-axis label.

Set the text to add units to the X-axis label.

Set the text to add units to the X-axis label.

54 `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.

Returns: the text to add units to the X-axis label. The label will only be shown if `x_label_on()` == true.

Returns: the text to add units to the X-axis label. The label will only be shown if `x_label_on()` == true.

45 `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.

46 `bool x_label_units_on();`

Set true if want X axis label to include units (as well as label like "length").

47 `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.

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.

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.

48 `bool x_label_units_on();`

Set true if want X axis label to include units (as well as label like "length").

Set true if want X axis label to include units (as well as label like "length").

Set true if want X axis label to include units (as well as label like "length").

49 `svg_boxplot & x_label_width(double width);`

Set the width (boldness) of X-axis label (including any units). (not recommended until browsers implement better).

50 `double x_label_width();`

Returns: the width (boldness) of X-axis label (including any units).

51 `svg_boxplot & x_label_width(double width);`

Set the width (boldness) of X-axis label (including any units). (not recommended until browsers implement better).

Set the width (boldness) of X-axis label (including any units). (not recommended until browsers implement better).

Set the width (boldness) of X-axis label (including any units). (not recommended until browsers implement better).

42 `double x_label_width();`

Returns: the width (boldness) of X-axis label (including any units).
 Returns: The width (boldness) of X-axis label (including any units).
 Returns: The width (boldness) of X-axis label (including any units).

43 `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.

44 `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.

Set true if want to strip redundant zeros, signs and exponents. **Example:** reducing "1.2e+000" to "1.2". This markedly reduces visual clutter, and is the default.

Set true if want to strip redundant zeros, signs and exponents. **Example:** reducing "1.2e+000" to "1.2". This markedly reduces visual clutter, and is the default.

45 `svg_boxplot & x_major_grid_color(const svg_color & col);`

Set the color of X-axis major grid lines.

46 `svg_color x_major_grid_color();`

Set the color of X-axis major grid lines.

47 `svg_boxplot & x_major_grid_color(const svg_color & col);`

Set the color of X-axis major grid lines.

Set the color of X-axis major grid lines.

Set the color of X-axis major grid lines.

48 `svg_color x_major_grid_color();`

Set the color of X-axis major grid lines.

Returns: the color of X-axis major grid lines.
 Returns: the color of X-axis major grid lines.

49 `svg_boxplot & x_major_grid_on(bool is);`

If set true, will include a major X-axis grid.

50 `bool x_major_grid_on();`

Returns: true if will include a major X-axis grid.

41 `svg_boxplot & x_major_grid_on(bool is);`

If set true, will include a major X-axis grid.

If set true, will include a major X-axis grid.

If set true, will include a major X-axis grid.

42 `bool x_major_grid_on();`

If true, will include a major X-axis grid.

If true, will include a major X-axis grid.

Returns: true if will include a major X-axis grid.

43 `svg_boxplot & x_major_grid_width(double w);`

Set the width of X-axis major grid lines.

44 `double x_major_grid_width();`

Returns: the color of X-axis major grid lines.

45 `svg_boxplot & x_major_grid_width(double w);`

Set the width of X-axis major grid lines.

Set the width of X-axis major grid lines.

Set the width of X-axis major grid lines.

46 `double x_major_grid_width();`

Returns: the color of X-axis major grid lines.

Returns: the color of X-axis major grid lines.

Returns: the color of X-axis major grid lines.

47 `svg_boxplot & x_major_interval(double inter);`

Set the interval between X-axis major ticks.

48 `double x_major_interval();`

Returns: the interval between X-axis major ticks.

49 `svg_boxplot & x_major_interval(double inter);`

Set the interval between X-axis major ticks.

Set the interval between X-axis major ticks.

Set the interval between X-axis major ticks.

40 `double x_major_interval();`

Returns: the interval between X-axis major ticks.
 Returns: the interval between X-axis major ticks.
 Returns: the interval between X-axis major ticks.

41 `svg_boxplot & x_major_label_rotation(rotate_style rot);`

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

`rotate_style`

42 `rotate_style x_major_label_rotation();`

See Also:

`rotate_style`

Returns: rotation for X ticks major value labels.

43 `svg_boxplot & x_major_label_rotation(rotate_style rot);`

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

`rotate_style`

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

`rotate_style`

Set rotation for X ticks major value labels. (Default horizontal).

See Also:

`rotate_style`

44 `rotate_style x_major_label_rotation();`

See Also:

`rotate_style`

See Also:

`rotate_style`

See Also:

`rotate_style`

Returns: rotation for X ticks major value labels.
 Returns: rotation for X ticks major value labels.
 Returns: rotation for X ticks major value labels.

45 `int x_major_labels();`

Returns: which side of the X axis for labels.

46 `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.

47 `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.

48 `int x_major_labels_side();`

Returns: the side for X ticks major value labels.

49 `svg_boxplot & x_major_labels_side(int side);`

Position of labels for X major ticks on horizontal X axis line.

Position of labels for major ticks on horizontal X-axis line.

- `place > 0` labels to left of Y-axis line (`left_side`) (default),
- `place = 0` (false) no major tick labels on Y-axis.
- `place > 0` labels to right of Y-axis line (`right_side`).

See Also:

`enum boost::svg::place for named`

Position of labels for major ticks on horizontal X-axis line.

- `place > 0` labels to left of Y-axis line (`left_side`) (default),
- `place = 0` (false) no major tick labels on Y-axis.
- `place > 0` labels to right of Y-axis line (`right_side`).

See Also:

`enum boost::svg::place for named`

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.

50 `int x_major_labels_side();`

Returns: the side for X ticks major value labels.

Returns: The side for X ticks major value labels (see `enum boost::svg::side`).

Returns: The side for X ticks major value labels (see `enum boost::svg::side`).

41 `svg_boxplot & x_major_tick(double d);`

Set interval (Cartesian units) between major ticks.

42 `double x_major_tick();`

Returns: interval (Cartesian units) between major ticks.

43 `svg_boxplot & x_major_tick(double d);`

Set interval (Cartesian units) between major ticks.

Set interval (Cartesian units) between major ticks.

Set interval (Cartesian units) between major ticks.

44 `double x_major_tick();`

Returns: interval (Cartesian units) between major ticks.

Returns: interval (Cartesian units) between major ticks.

Returns: interval (Cartesian units) between major ticks.

45 `svg_boxplot & x_major_tick_color(const svg_color & col);`

Set the color of X-axis major ticks.

46 `svg_color x_major_tick_color();`

Returns: the color of X-axis major ticks.

47 `svg_boxplot & x_major_tick_color(const svg_color & col);`

Set the color of X-axis major ticks.

Set the color of X-axis major ticks.

Set the color of X-axis major ticks.

48 `svg_color x_major_tick_color();`

Returns: the color of X-axis major ticks.

Returns: the color of X-axis major ticks.

Returns: the color of X-axis major ticks.

49 `svg_boxplot & x_major_tick_length(double length);`

Set length of X major ticks (SVG units, default pixels).

50 `double x_major_tick_length();`

Set length of X major ticks (SVG units, default pixels).

41 `svg_boxplot & x_major_tick_length(double length);`

Set length of X major ticks (SVG units, default pixels).

Set length of X major ticks.

Set length of X major ticks.

42 `double x_major_tick_length();`

Set length of X major ticks (SVG units, default pixels).

Returns: length of X major ticks.

Returns: length of X major ticks.

43 `svg_boxplot & x_major_tick_width(double width);`

Set width of X major ticks (SVG units, default pixels).

Set width of X major ticks.

Set width of X major ticks.

44 `double x_major_tick_width();`

Returns: Width of major ticks on the X axis.

45 `svg_boxplot & x_max(double x);`

Set the maximum value on the X-axis.

46 `double x_max();`

autoscale set & get parameters, Note: all these *MUST* precede x_autoscale(data) call.

Returns: the maximum value on the X-axis.

47 `svg_boxplot & x_max(double x);`

Set the maximum value on the X-axis.

Set the maximum value on the X-axis.

Set the maximum value on the X-axis.

48 `double x_max();`

autoscale set & get parameters, Note: all these *MUST* precede x_autoscale(data) call.

Returns: the maximum value on the X-axis.

Returns: the maximum value on the X-axis. (Can also get both minimum and maximum as a std::pair).

Returns: the maximum value on the X-axis. (Can also get both minimum and maximum as a std::pair).

49 `svg_boxplot & x_min(double min_x);`

Set the minimum value on the X-axis.

40 `double x_min();`

Returns: the minimum value on the X-axis.

41 `svg_boxplot & x_min(double min_x);`

Set the minimum value on the X-axis.

Set the minimum value on the X-axis.

Set the minimum value on the X-axis.

42 `double x_min();`

Returns: the minimum value on the X-axis.

Returns: the minimum value on the X-axis. (Can also get both minimum and maximum as a std::pair).

Returns: the minimum value on the X-axis. (Can also get both minimum and maximum as a std::pair).

43 `svg_boxplot & x_min_ticks(int min_ticks);`

Set X-axis autoscale to include at least minimum number of ticks (default = 6).

44 `int x_min_ticks();`

Returns: X-axis autoscale minimum number of ticks.

45 `svg_boxplot & x_min_ticks(int min_ticks);`

Set X-axis autoscale to include at least minimum number of ticks (default = 6).

Set X-axis autoscale to include at least minimum number of ticks (default = 6). Must precede x_autoscale(data) call.

Set X-axis autoscale to include at least minimum number of ticks (default = 6). Must precede x_autoscale(data) call.

46 `int x_min_ticks();`

Returns: X-axis autoscale minimum number of ticks.

Returns: X-axis autoscale minimum number of ticks.

Returns: X-axis autoscale minimum number of ticks.

47 `svg_boxplot & x_minor_grid_color(const svg_color & col);`

Set the color of X-axis minor grid lines.

48 `svg_color x_minor_grid_color();`

Returns: the color of X-axis minor grid lines.

49 `svg_boxplot & x_minor_grid_color(const svg_color & col);`

Set the color of X-axis minor grid lines.

Set the color of X-axis minor grid lines.

Set the color of X-axis minor grid lines.

40 `svg_color x_minor_grid_color();`

Returns: the color of X-axis minor grid lines.

Returns: the color of X-axis minor grid lines.

Returns: the color of X-axis minor grid lines.

41 `svg_boxplot & x_minor_grid_on(bool is);`

If set true, will include a minor X-axis grid.

42 `bool x_minor_grid_on();`

Returns: true if will include a major X-axis grid.

43 `svg_boxplot & x_minor_grid_on(bool is);`

If set true, will include a minor X-axis grid.

If set true, will include a minor X-axis grid.

If set true, will include a minor X-axis grid.

44 `bool x_minor_grid_on();`

If true, will include a minor X-axis grid.

If true, will include a minor X-axis grid.

Returns: true if will include a major X-axis grid.

45 `svg_boxplot & x_minor_grid_width(double w);`

Set the width of X-axis minor grid lines.

46 `double x_minor_grid_width();`

Returns: the width of X-axis minor grid lines.

47 `svg_boxplot & x_minor_grid_width(double w);`

Set the width of X-axis minor grid lines.

Set the width of X-axis minor grid lines.

Set the width of X-axis minor grid lines.

48 `double x_minor_grid_width();`

Returns: the width of X-axis minor grid lines.

Returns: the width of X-axis minor grid lines.

Returns: the width of X-axis minor grid lines.

49 `double x_minor_interval();`

Returns: interval between X minor ticks.

50 `double x_minor_interval();`

Returns: interval between X minor ticks.

Returns: interval between X minor ticks.

Returns: interval between X minor ticks.

51 `svg_boxplot & x_minor_interval(double interval);`

Set interval between X-axis minor ticks.

52 `svg_boxplot & x_minor_interval(double interval);`

Set interval between X-axis minor ticks.

Set interval between X-axis minor ticks.

Set interval between X-axis minor ticks.

53 `svg_boxplot & x_minor_tick_color(const svg_color & col);`

Set the color of X-axis minor ticks.

54 `svg_color x_minor_tick_color();`

Returns: the color of X-axis minor ticks.

55 `svg_boxplot & x_minor_tick_color(const svg_color & col);`

Set the color of X-axis minor ticks.

Set the color of X-axis minor ticks.

Set the color of X-axis minor ticks.

56 `svg_color x_minor_tick_color();`

Returns: the color of X-axis minor ticks.

Returns: the color of X-axis minor ticks.

Returns: the color of X-axis minor ticks.

57 `svg_boxplot & x_minor_tick_length(double length);`

Set length of X minor ticks (SVG units, default pixels).

58 `double x_minor_tick_length();`

Returns: length of X minor ticks (SVG units, default pixels).

49 `svg_boxplot & x_minor_tick_length(double length);`

Set length of X minor ticks (SVG units, default pixels).

Set length of X minor ticks.

Set length of X minor ticks.

50 `double x_minor_tick_length();`

Returns: length of X minor ticks (SVG units, default pixels).

Returns: length of X minor ticks.

Returns: length of X minor ticks.

51 `svg_boxplot & x_minor_tick_width(double width);`

Set width of X minor ticks (SVG units, default pixels).

52 `double x_minor_tick_width();`

Returns: width of X minor ticks (SVG units, default pixels).

53 `svg_boxplot & x_minor_tick_width(double width);`

Set width of X minor ticks (SVG units, default pixels).

Set width of X minor ticks.

Set width of X minor ticks.

54 `double x_minor_tick_width();`

Returns: width of X minor ticks (SVG units, default pixels).

Returns: width of X minor ticks.

Returns: width of X minor ticks.

55 `svg_boxplot & x_num_minor_ticks(unsigned int num);`

Set number of X-axis minor ticks between major ticks.

Set number of X-axis minor ticks between major ticks.

56 `unsigned int x_num_minor_ticks();`

Returns: number of X-axis minor ticks between major ticks.

57 `svg_boxplot & x_num_minor_ticks(int num);`

Set number of X-axis minor ticks between major ticks.

Set number of X-axis minor ticks between major ticks.

58 `int x_num_minor_ticks();`

Returns: number of X-axis minor ticks between major ticks.
 Returns: number of X-axis minor ticks between major ticks. Note: NOT float or double!
 Returns: number of X-axis minor ticks between major ticks. Note: NOT float or double!

59 `svg_boxplot & x_order_color(const svg_color & col);`

Set the color of X order #, for example, the color of #42.

60 `svg_color x_order_color();`

Returns: the color of X order #, for example, the color of #42.

61 `svg_boxplot & x_order_color(const svg_color & col);`

Set the color of X order #, for example, the color of #42.

Set the color of X point order in sequence, for example, #3.

Set the color of X point order in sequence, for example, #3.

62 `svg_color x_order_color();`

Get the color of X point order in sequence, for example, #3.

Get the color of X point order in sequence, for example, #3.

Returns: the color of X order #, for example, the color of #42.

63 `svg_boxplot & x_order_on(bool b);`

Set true if to append append an order # to data point X values near data points markers.

64 `bool x_order_on();`

Returns: true if to append an order # to data point X values near data points markers.

65 `svg_boxplot & x_order_on(bool b);`

Set true if to append append an order # to data point X values near data points markers.

Set true if to append an order # to data point X values near data points markers.

Set true if to append an order # to data point X values near data points markers.

66 `bool x_order_on();`

Returns: true if to append an order # to data point X values near data points markers.
 Returns: true if to append an order # to data point X values near data points markers.
 Returns: true if to append an order # to data point X values near data points markers.

67 `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)".

58 `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)".

59 `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)".

Set the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

Set the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

60 `svg_color x_plusminus_color();`

Get the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

Get the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

Returns: the color of X std_dev of value, for example, the color of 0.02 in "1.23 +-0.02 (9)".

61 `svg_boxplot & x_plusminus_on(bool b);`

Set if to append std_dev estimate to data point X values near data points markers.

62 `bool x_plusminus_on();`

Returns: true if to append std_dev estimate to data point X values near data points markers.

63 `svg_boxplot & x_plusminus_on(bool b);`

Set if to append std_dev estimate to data point X values near data points markers.

Set if to append std_dev estimate to data point X values near data points markers. (May not be implemented yet).

Set if to append std_dev estimate to data point X values near data points markers. (May not be implemented yet).

64 `bool x_plusminus_on();`

Returns: true if to append std_dev estimate to data point X values near data points markers.

Returns: if to append std_dev estimate to data point X values near data points markers. (May not be implemented yet).

Returns: if to append std_dev estimate to data point X values near data points markers. (May not be implemented yet).

65 `const std::string x_prefix();`

Returns: the prefix.

66 `const std::string x_prefix();`

Get the prefix (only used if separator != "")

Get the prefix (only used if separator != "")

Returns: the prefix.

57 `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.

58 `std::pair< double, double > x_range();`

Returns: the range of values on the X-axis.

59 `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.

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 doubles, and the range must not be too small.

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 doubles, and the range must not be too small.

60 `std::pair< double, double > x_range();`

Returns: the range of values on the X-axis.

Returns: the range of values on the X-axis. (Need to use boost::svg::detail::operator<< to display this).

Returns: the range of values on the X-axis. (Need to use boost::svg::detail::operator<< to display this).

61 `const std::string x_separator();`

Returns: the separator, perhaps including Unicode.

62 `const std::string x_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;", for example, "\u00A0;" rather than ", ".

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;", for example, "\u00A0;" rather than ", ".

Returns: the separator, perhaps including Unicode.

63 `svg_boxplot & x_size(unsigned int i);`

Set SVG image X-axis size (SVG units, default pixels).

Set SVG image X-axis size (SVG units, default pixels).

64 `svg_boxplot & x_size(int i);`

Set SVG image X-axis size (SVG units, default pixels).

Set SVG image X-axis size (SVG units, default pixels).

55 `unsigned int x_size();`

Returns: width of the SVG image.

56 `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).

57 `int x_steps();`

Returns: autoscale to set ticks in steps.

58 `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).

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).

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).

59 `int x_steps();`

Returns: autoscale to set ticks in steps.
Returns: autoscale to set ticks in steps.
Returns: autoscale to set ticks in steps.

60 `const std::string x_suffix();`

Returns: the suffix (only used if separator != "")

61 `const std::string x_suffix();`

Get the suffix (only used if separator != "")

Get the suffix (only used if separator != "")
Returns: the suffix (only used if separator != "")

62 `svg_boxplot & x_tick_color(const svg_color & col);`

Returns: Y major ticks color.
Returns: Reference to `svg_boxplot` to make chainable.

63 `svg_color x_tick_color();`

Returns: Color of ticks on the X axis.

64 `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.

65 `double x_tick_length();`

Returns: Length of major ticks on the X axis.

66 `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.

67 `svg_boxplot & x_ticks_down_on(bool cmd);`

Set true if Y major ticks should mark upwards.

58 `bool x_ticks_down_on();`

Returns: true if Y major ticks should mark upwards.

59 `svg_boxplot & x_ticks_down_on(bool cmd);`

Set true if Y major ticks should mark upwards.

Set true if X major ticks should mark downwards.

Set true if X major ticks should mark downwards.

60 `bool x_ticks_down_on();`

Returns: true if Y major ticks should mark upwards.

Returns: true if X major ticks should mark downwards.

Returns: true if X major ticks should mark downwards.

61 `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.

62 `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.

63 `svg_boxplot & x_ticks_on_window_or_axis(int side);`

Set position of X ticks on window or axis.

Set X ticks on window or axis

- cmd -1 bottom of plot window,
- cmd 0 on X axis.

• cmd +1 top of plot window.

Set X ticks on window or axis

- cmd -1 bottom of plot window,
- cmd 0 on X axis.

• cmd +1 top of plot window.

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.

64 `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.

Returns: if X axis ticks wanted on the window (rather than on axis).

-1 bottom of plot window, 0 on X axis, +1 top of plot window.

Returns: if X axis ticks wanted on the window (rather than on axis).

-1 bottom of plot window, 0 on X axis, +1 top of plot window.

55 `svg_boxplot & x_ticks_up_on(bool cmd);`

Set true if X major ticks should mark upwards.

56 `bool x_ticks_up_on();`

Returns: true if X major ticks should mark upwards.

57 `svg_boxplot & x_ticks_up_on(bool cmd);`

Set true if X major ticks should mark upwards.

Set true if X major ticks should mark upwards.

Set true if X major ticks should mark upwards.

58 `bool x_ticks_up_on();`

Returns: true if X major ticks should mark upwards.

Returns: true if X major ticks should mark upwards.

Returns: true if X major ticks should mark upwards.

59 `svg_boxplot & x_ticks_values_color(const svg_color & col);`

Set X axis tick value label color.

60 `svg_color x_ticks_values_color();`

Returns: X-axis ticks value label color.

61 `svg_boxplot & x_ticks_values_color(const svg_color & col);`

Set X axis tick value label color.

Set X axis tick value label color.

Set X axis tick value label color.

62 `svg_color x_ticks_values_color();`

Returns: X-axis ticks value label color.

Returns: X-axis ticks value label color.

Returns: X-axis ticks value label color.

63 `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"`

54 `const std::string & x_ticks_values_font_family();`

Returns: X ticks value label font family.

55 `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"
```

Set X ticks value label font family.

Set X ticks value label font family.

56 `const std::string & x_ticks_values_font_family();`

Returns: X ticks value label font family.

Returns: X ticks value label font family.

Returns: X ticks value label font family.

57 `svg_boxplot & x_ticks_values_font_size(unsigned int i);`

Set X ticks value label font size (svg units, default pixels).

Set X ticks value label font size (svg units, default pixels).

58 `unsigned int x_ticks_values_font_size();`

Set X ticks value label font size (svg units, default pixels).

59 `svg_boxplot & x_ticks_values_font_size(int i);`

Set X ticks value label font size (svg units, default pixels).

Set X ticks value label font size (svg units, default pixels).

60 `int x_ticks_values_font_size();`

Set X ticks value label font size (svg units, default pixels).

Returns: X ticks value label font size (svg units, default pixels).

Returns: X ticks value label font size (svg units, default pixels).

61 `svg_boxplot & x_ticks_values_ioflags(std::ios_base::fmtflags f);`

Set iostream format flags of data point X values near data points markers.

62 `std::ios_base::fmtflags x_ticks_values_ioflags();`

Returns: iostream format flags of data point X values near data points markers.

53 `svg_boxplot & x_ticks_values_ioflags(std::ios_base::fmtflags f);`

Set ostream format flags of data point X values near data points markers.

Set ostream format flags of data point X values near data points markers. Useful to set hexadecimal, fixed and scientific, (std::ios::scientific).

Set ostream format flags of data point X values near data points markers. Useful to set hexadecimal, fixed and scientific, (std::ios::scientific).

54 `std::ios_base::fmtflags x_ticks_values_ioflags();`

Returns: ostream format flags of data point X values near data points markers.

Returns: ostream format flags of data point X values near data points markers. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

Returns: ostream format flags of data point X values near data points markers. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

55 `svg_boxplot & x_ticks_values_precision(int p);`

Set ostream decimal digits precision of data point X values near data points markers.

56 `int x_ticks_values_precision();`

Returns: ostream decimal digits precision of data point X values near data points markers.

57 `svg_boxplot & x_ticks_values_precision(int p);`

Set ostream decimal digits precision of data point X values near data points markers.

Set ostream decimal digits precision of data point X values near data points markers.

Set ostream decimal digits precision of data point X values near data points markers.

58 `int x_ticks_values_precision();`

Returns: ostream decimal digits precision of data point X values near data points markers.

Returns: ostream decimal digits precision of data point X values near data points markers.

Returns: ostream decimal digits precision of data point X values near data points markers.

59 `svg_boxplot & x_tight(double tight);`

Set tolerance to autoscale to permit data points slightly outside both end ticks.

60 `double x_tight();`

Returns: tolerance given to autoscale to permit data points slightly outside both end ticks. Get results of autoscaling.

61 `svg_boxplot & x_tight(double tight);`

Set tolerance to autoscale to permit data points slightly outside both end ticks.

Set tolerance to autoscale to permit data points slightly outside both end ticks. default 0. Must precede x_autoscale(data) call.

Set tolerance to autoscale to permit data points slightly outside both end ticks. default 0. Must preceed x_autoscale(data) call.

§2 `double x_tight();`

Returns: tolerance given to autoscale to permit data points slightly outside both end ticks. Get results of autoscaling.
 Returns: tolerance given to autoscale to permit data points slightly outside both end ticks.
 Returns: tolerance given to autoscale to permit data points slightly outside both end ticks.

§3 `svg_boxplot & x_value_font_size(unsigned int i);`

Set X tick value label font size (svg units, default pixels).

Set X tick value label font size (svg units, default pixels).

§4 `unsigned int x_value_font_size();`

Returns: X tick value label font size (svg units, default pixels).

§5 `svg_boxplot & x_value_font_size(int i);`

Set X tick value label font size (svg units, default pixels).

Set X tick value label font size (svg units, default pixels).

§6 `int x_value_font_size();`

Returns: X tick value label font size (svg units, default pixels).
 Returns: X tick value label font size (svg units, default pixels).
 Returns: X tick value label font size (svg units, default pixels).

§7 `svg_boxplot & 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)`

§8 `std::ios_base::fmtflags x_value_ioflags();`

Returns: stream std::ios::fmtflags for control of format of X value labels.

§9 `svg_boxplot & 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)`

Set ostream std::ios::fmtflags for X value label (default decimal == 0X201). Mainly useful for changing to scientific, fixed or hexadecimal format. For example: `.x_value_ioflags(std::ios::dec | std::ios::scientific)`

Set iostream std::ios::fmtflags for X value label (default decimal == 0X201). Mainly useful for changing to scientific, fixed or hexadecimal format. For example: .x_value_ioflags(std::ios::dec | std::ios::scientific)

50 `std::ios_base::fmtflags x_value_ioflags();`

Returns: stream std::ios::fmtflags for control of format of X value labels.
 Returns: stream ioflags for control of format of X value labels.
 Returns: stream ioflags for control of format of X value labels.

51 `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.

52 `int x_value_precision();`

Returns: Precision of X-tick label values in decimal digits

53 `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.

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.

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.

54 `int x_value_precision();`

Returns: Precision of X-tick label values in decimal digits
 Returns: precision of X tick label values in decimal digits
 Returns: precision of X tick label values in decimal digits

55 `svg_boxplot & x_values_color(const svg_color & col);`

Set the color of data point X values near data points markers.

56 `svg_color x_values_color();`

Returns: the color of data point X values near data points markers.

57 `svg_boxplot & x_values_color(const svg_color & col);`

Set the color of data point X values near data points markers.

Set the color of data point X values near data points markers.

Set the color of data point X values near data points markers.

58 `svg_color x_values_color();`

Returns: the color of data point X values near data points markers.
 Returns: the color of data point X values near data points markers.
 Returns: the color of data point X values near data points markers.

59 `svg_boxplot & x_values_font_family(const std::string & family);`

Set font family of data point X values near data points markers.

60 `const std::string & x_values_font_family();`

Returns: font family of data point X values near data points markers.

61 `svg_boxplot & x_values_font_family(const std::string & family);`

Set font family of data point X values near data points markers.

Set font family of data point X values near data points markers.

Set font family of data point X values near data points markers.

62 `const std::string & x_values_font_family();`

Set font family of data point X values near data points markers.

Set font family of data point X values near data points markers.

Returns: font family of data point X values near data points markers.

63 `svg_boxplot & x_values_font_size(unsigned int i);`

Set font size of data point X values near data points markers.

Set font size of data point X values near data points markers.

64 `unsigned int x_values_font_size();`

Returns: font size of data point X values near data points markers.

65 `svg_boxplot & x_values_font_size(int i);`

Set font size of data point X values near data points markers.

Set font size of data point X values near data points markers.

66 `int x_values_font_size();`

Returns: font size of data point X values near data points markers.
 Returns: font size of data point X values near data points markers.
 Returns: font size of data point X values near data points markers.

67 `svg_boxplot & x_values_ioflags(std::ios_base::fmtflags f);`

Set iostream format flags of data point X values near data points markers.

68 `std::ios_base::fmtflags x_values_ioflags();`

Returns: iostream format flags of data point X values near data points markers.

69 `svg_boxplot & x_values_ioflags(std::ios_base::fmtflags f);`

Set iostream format flags of data point X values near data points markers.

Set iostream format flags of data point X values near data points markers. Useful to set hexadecimal, fixed and scientific, (std::ios::scientific).

Set iostream format flags of data point X values near data points markers. Useful to set hexadecimal, fixed and scientific, (std::ios::scientific).

70 `std::ios_base::fmtflags x_values_ioflags();`

Returns: iostream format flags of data point X values near data points markers.

Returns: iostream format flags of data point X values near data points markers. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

Returns: iostream format flags of data point X values near data points markers. Might be used to set hexadecimal, fixed and scientific, (std::ios::scientific).

71 `svg_boxplot & x_values_on(bool b);`

Set true to show data point values near data points markers.

72 `bool x_values_on();`

Returns: true if to show data point values near data points markers.

73 `svg_boxplot & x_values_on(bool b);`

Set true to show data point values near data points markers.

Returns: true if values of X data points are shown (for example: 1.23).

Returns: true if values of X data points are shown (for example: 1.23).

74 `bool x_values_on();`

If true, show data point values near data points markers.

If true, show data point values near data points markers.

Returns: true if to show data point values near data points markers.

75 `svg_boxplot & x_values_precision(int p);`

Set iostream decimal digits precision of data point X values near data points markers.

76 `int x_values_precision();`

Returns: iostream decimal digits precision of data point X values near data points markers.

67 `svg_boxplot & x_values_precision(int p);`

Set ostream decimal digits precision of data point X values near data points markers.

Set ostream decimal digits precision of data point X values near data points markers.

Set ostream decimal digits precision of data point X values near data points markers.

68 `int x_values_precision();`

Returns: ostream decimal digits precision of data point X values near data points markers.

Returns: ostream decimal digits precision of data point X values near data points markers.

Returns: ostream decimal digits precision of data point X values near data points markers.

69 `svg_boxplot & x_values_rotation(rotate_style rotate);`

Returns: the rotation (rotate_style) of data point X values near data points markers.

70 `int x_values_rotation();`

Set the rotation (rotate_style) of data point X values near data points markers.

71 `svg_boxplot & x_values_rotation(rotate_style rotate);`

Returns: the rotation (rotate_style) of data point X values near data points markers.

Returns: the rotation (rotate_style) of data point X values near data points markers. (Degrees: 0 to 360 in 45 steps).

Returns: the rotation (rotate_style) of data point X values near data points markers. (Degrees: 0 to 360 in 45 steps).

72 `int x_values_rotation();`

Set the rotation (rotate_style) of data point X values near data points markers.

Returns: the rotation of data point X values near data points markers.

Returns: the rotation of data point X values near data points markers.

73 `svg_boxplot & x_with_zero(bool b);`

Set X-axis autoscale to include zero (default = false).

74 `bool x_with_zero();`

Returns: true if X-axis autoscale to include zero (default = false).

75 `svg_boxplot & x_with_zero(bool b);`

Set X-axis autoscale to include zero (default = false).

Set X-axis autoscale to include zero (default = false). Must preceed x_autoscale(data) call.

Set X-axis autoscale to include zero (default = false). Must preceed x_autoscale(data) call.

76 `bool x_with_zero();`

Returns: true if X-axis autoscale to include zero (default = false).
 Returns: true if X-axis autoscale to include zero (default = false).
 Returns: true if X-axis autoscale to include zero (default = false).

67 `bool y_autoscale();`

Returns: true if Y-axis to use autoscaling.

68 `svg_boxplot & y_autoscale(bool b);`

Set true if Y axis is to use autoscale.

Returns: Reference to `svg_boxplot` to make chainable.

69 `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.

70 `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.

71 `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.

72 `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.

73 `svg_boxplot & y_axis_color(const svg_color & col);`

Set the color of the Y-axis line.

74 `svg_color y_axis_color();`

Returns: the color of the Y-axis line.

75 `svg_boxplot & y_axis_color(const svg_color & col);`

Set the color of the Y-axis line.

Set the color of the Y-axis line.

Set the color of the Y-axis line.

66 `svg_color y_axis_color();`

Returns: the color of the Y-axis line.

Returns: The color of the Y-axis line.

Returns: The color of the Y-axis line.

67 `svg_boxplot & y_axis_on(bool is);`

If set true, draw a vertical Y-axis line.

68 `bool y_axis_on();`

Returns: true if will draw a horizontal X-axis line.

69 `svg_boxplot & y_axis_on(bool is);`

If set true, draw a vertical Y-axis line.

If set true, draw a vertical Y-axis line.

If set true, draw a vertical Y-axis line.

70 `bool y_axis_on();`

If true, draw a vertical Y-axis line.

If true, draw a vertical Y-axis line.

Returns: true if will draw a horizontal X-axis line.

71 `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.

72 `double y_axis_position();`

Returns: Position of the vertical Y-axis line (on the border).

73 `std::string y_label();`

Returns: the text for the Y-axis label. The label will only be shown if `y_label_on() == true`.

Returns: the text for the Y-axis label. The label will only be shown if `y_label_on() == true`.

Returns: the text for the Y-axis label. The label will only be shown if `y_label_on() == true`.

74 `svg_boxplot & y_label(const std::string & str);`

Set Y axis label.

Returns: Reference to `svg_boxplot` to make chainable.

65 `svg_color y_label_color();`

Returns: the color of Y-axis label (including any units).

Returns: the color of Y-axis label (including any units).

Returns: the color of Y-axis label (including any units).

66 `svg_boxplot & y_label_color(const svg_color & col);`

Set font color for Y axis label.

Returns: Reference to `svg_boxplot` to make chainable.

67 `svg_boxplot & y_label_font_size(unsigned int size);`

Set font size for Y axis label.

Returns: Reference to `svg_boxplot` to make chainable.

68 `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.

69 `std::string y_label_text();`

Returns: Text of label for Y axis.

70 `svg_boxplot & y_label_units(const std::string & str);`

Set the text to add units to the Y-axis label.

71 `std::string y_label_units();`

Returns: the text to add units to the X-axis label.

72 `svg_boxplot & y_label_units(const std::string & str);`

Set the text to add units to the Y-axis label.

Set the text to add units to the Y-axis label.

Set the text to add units to the Y-axis label.

73 `std::string y_label_units();`

Returns: the text to add units to the X-axis label.

Returns: the text to add units to the X-axis label.

Returns: the text to add units to the X-axis label.

64 `bool y_labels_strip_e0s();`

Returns: if to strip redundant zeros, signs and exponents, for example, reducing "1.2e+000" to "1.2"

65 `bool y_labels_strip_e0s();`

Returns: if to strip redundant zeros, signs and exponents, for example, reducing "1.2e+000" to "1.2"

Returns: true if set to strip redundant zeros, signs and exponents. **Example:** reducing "1.2e+000" to "1.2".

Returns: true if set to strip redundant zeros, signs and exponents. **Example:** reducing "1.2e+000" to "1.2".

66 `svg_boxplot & y_major_interval(double inter);`

Set the interval between major ticks on the Y axis.

67 `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.

68 `svg_boxplot & y_major_tick_color(const svg_color & col);`

Set Y major ticks color.

Returns: Reference to `svg_boxplot` to make chainable.

69 `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.

70 `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.

71 `double y_minor_interval();`

Returns: interval between Y minor ticks.

72 `double y_minor_interval();`

Returns: interval between Y minor ticks.

Returns: interval between Y minor ticks.

Returns: interval between Y minor ticks.

73 `svg_boxplot & y_minor_tick_color(const svg_color & col);`

Set Y minor ticks color.

Returns: Reference to `svg_boxplot` to make chainable.

64 `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.

65 `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.

66 `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.

67 `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.

68 `std::pair< double, double > y_range();`

Set `y_range` using a pair of doubles (and do not use autoscale).

69 `svg_boxplot & y_size(unsigned int i);`

Set SVG image Y-axis size (SVG units, default pixels).

Set SVG image Y-axis size (SVG units, default pixels).

70 `svg_boxplot & y_size(int i);`

Set SVG image Y-axis size (SVG units, default pixels).

Set SVG image Y-axis size (SVG units, default pixels).

71 `unsigned int y_size();`

Returns: height of the SVG image.

svg_boxplot protected member functions

1. `void adjust_limits(double & x, double & y);`

2. `void adjust_limits(double & x, double & y);`

If value of a data point reaches limit of max, min, infinity, use the appropriate plot min or max value instead.

If value of a data point reaches limit of max, min, infinity, use the appropriate plot min or max value instead.

3. `void clear_background();`

Clear the whole image background layer of the SVG plot.

4. `void clear_background();`

Clear the whole image background layer of the SVG plot.

< Clear the whole image background layer of the SVG plot.

< Clear the whole image background layer of the SVG plot.

5. `void clear_grids();`

Clear the grids layer of the SVG plot.

6. `void clear_grids();`

Clear the grids layer of the SVG plot.

< Clear the grids layer of the SVG plot.

< Clear the grids layer of the SVG plot.

7. `void clear_legend();`

Clear the legend layer of the SVG plot.

8. `void clear_legend();`

Clear the legend layer of the SVG plot.

< Clear the legend layer of the SVG plot.

< Clear the legend layer of the SVG plot.

9. `void clear_plot_background();`

Clear the plot area background layer of the SVG plot.

10. `void clear_plot_background();`

Clear the plot area background layer of the SVG plot.

< Clear the plot area background layer of the SVG plot.

< Clear the plot area background layer of the SVG plot.

11. `void clear_points();`

Clear the data points layer of the SVG plot.

12. `void clear_points();`

Clear the data points layer of the SVG plot.

< Clear the data points layer of the SVG plot.

< Clear the data points layer of the SVG plot.

13. `void clear_title();`

Clear the plot title layer of the SVG plot.

14. `void clear_title();`

Clear the Y axis layer of the SVG plot.

Clear the plot title layer of the SVG plot.

< Clear the plot title layer of the SVG plot.

< Clear the plot title layer of the SVG plot.

15. `void clear_x_axis();`

Clear the X axis layer of the SVG plot.

16. `void clear_x_axis();`

Clear the X axis layer of the SVG plot.

< Clear the X axis layer of the SVG plot.

< Clear the X axis layer of the SVG plot.

17. `void clear_y_axis();`

Clear the Y axis layer of the SVG plot.

18. `void clear_y_axis();`

< Clear the Y axis layer of the SVG plot.

< Clear the Y axis layer of the SVG plot.

19. `void draw_legend();`

20. `void draw_legend();`

Draw the legend border and background, (using the size and position computed by function `size_legend_box`), and legend text title (if any and if required), and any data point marker lines, and any shapes for data point markers, and any data series descriptor text(s).

Draw the legend border, text header (if any) and data point marker lines and/or shapes.

21. `void draw_plot_point(double x, double y, g_element & g_ptr,
const plot_point_style & sty);`

22. `void draw_plot_point(double x, double y, g_element & g_ptr,
plot_point_style & sty, unc< false > ux,
unc< false > uy);`

23. `void draw_plot_point(double x, double y, g_element & g_ptr,
const plot_point_style & sty);`

24. `void draw_plot_point(double x, double y, g_element & g_ptr,
plot_point_style & sty, unc< false > ux,
unc< false > uy);`

Draw a plot data point marker shape or symbol whose size and stroke and fill colors are specified in `plot_point_style` `sty`, possibly including uncertainty ellipses showing multiples of standard deviation.

Draw a plot data point marker shape or symbol whose size and stroke and fill colors are specified in `plot_point_style` `sty`, possibly including uncertainty ellipses showing multiples of standard deviation.

25. `void draw_plot_point_value(double x, double y, g_element & g_ptr,
value_style & val_style,
plot_point_style & point_style, Meas uvalue);`

26. `void draw_plot_point_value(double x, double y, g_element & g_ptr,
value_style & val_style,
plot_point_style & point_style, Meas uvalue);`

`void draw_plot_point_value(double x, double y, g_element& g_ptr, value_style& val_style, plot_point_style& point_style, unc<false> uvalue)` Write one data point (X or Y) value as a string, for example "1.23e-2", near the data point marker. Unnecessary e, +, & leading exponent zeros may optionally be stripped, and the position and rotation controlled. std_dev estimate, typically standard deviation (approximately half conventional 95% confidence "plus or minus") may be optionally be appended. Degrees of freedom estimate (number of replicates) may optionally be appended. ID or name of point, order in sequence, and datetime may also be added. For example: "3.45 +-0.1(10)"

The precision and format (scientific, fixed), and color and font type and size can be controlled too.

Unicode space text_plusminus glyph.

`void draw_plot_point_value(double x, double y, g_element& g_ptr, value_style& val_style, plot_point_style& point_style, unc<false> uvalue)` Write one data point (X or Y) value as a string, for example "1.23e-2", near the data point marker. Unnecessary e, +, & leading exponent zeros may optionally be stripped (highly recommended), and the position and rotation controlled. std_dev estimate, typically standard deviation (approximately half conventional 95% confidence "plus or minus") may be optionally be appended. Degrees of freedom estimate (number of replicates) may optionally be appended. ID or name of point, order in sequence, and datetime may also be added. For example: "3.45 +-0.1(10)"

The precision and format (scientific, fixed), and color and font type and size can be controlled too.

Unicode space and text_plusminus glyph.

27.

```
void draw_plot_point_values(double x, double y, g_element & x_g_ptr,
    g_element & y_g_ptr, const value_style & x_sty,
    const value_style & y_sty, Meas uncx, Meas unc);
```

28.

```
void draw_plot_point_values(double x, double y, g_element & x_g_ptr,
    g_element & y_g_ptr, const value_style & x_sty,
    const value_style & y_sty, Meas uncx,
    unc< false > unc);
```

29.

```
void draw_plot_point_values(double x, double y, g_element & x_g_ptr,
    g_element & y_g_ptr, const value_style & x_sty,
    const value_style & y_sty, Meas uncx, Meas unc);
```

30.

```
void draw_plot_point_values(double x, double y, g_element & x_g_ptr,
    g_element & y_g_ptr, const value_style & x_sty,
    const value_style & y_sty, Meas uncx,
    unc< false > unc);
```

Write the **pair** of data points X and Y values as a string.

The x parameter also carries the measurement information for the pair, and so is a **Meas**, not just an **unc<false>** as is the Y parameter. If a separator starting with newline, then both on the same line, for example "1.23, 3.45", or "[5.6, 7.8] X value_style is used to provide the prefix and separator, and Y value_style to provide the suffix. For example,

x_style prefix("[X=", and separator ",\u00A0;Y= ", " and Y value_style = "]" will produce a value label like "[X=-1.23, Y=4.56]"



Note

You need to use a Unicode space for get space for all browsers. For a long a string you may need to make the total image size bigger, and to orient the value labels with care. `draw_plot_point_values` is only when both X and Y pairs are wanted.

Also strip unnecessary e, + and leading exponent zeros, if required.

Unicode space text_plusminus glyph.

Write the **pair** of data points X and Y values as a string.

The x parameter also carries the measurement information for the pair, and so is a **Meas**, not just an **unc<false>** as is the Y parameter. If a separator starting with newline, then both on the same line, for example "1.23, 3.45", or "[5.6, 7.8] X value_style is used to provide the prefix and separator, and Y value_style to provide the suffix. For example,

x_style prefix("[X=", and separator ",\u00A0;Y= ", " and Y value_style = "]" will produce a value label like "[X=-1.23, Y=4.56]"



Note

You need to use a Unicode space for get space for all browsers. For a long a string you may need to make the total image size bigger, and to orient the value labels with care. draw_plot_point_values is only when both X and Y pairs are wanted.

Also strip unnecessary e, + and leading exponent zeros, if required.

Unicode space text_plusminus glyph.

31. `void draw_x_major_tick(double i, path_element & tick_path,
path_element & grid_path);`

Draw major ticks - and grid too if wanted. If major_value_labels_side then value shown beside the major tick.

Draw major ticks - and grid too if wanted. If major_value_labels_side then value shown beside the major tick.

32. `void draw_x_minor_tick(double j, path_element & tick_path,
path_element & grid_path);`

33. `void draw_x_minor_tick(double j, path_element & tick_path,
path_element & grid_path);`

< Draw X-axis minor ticks, and optional grid. (Value is NOT (yet) shown beside the minor tick).

< Draw X-axis minor ticks, and optional grid. (Value is NOT (yet) shown beside the minor tick).

34. `void place_legend_box();`

35. `void place_legend_box();`

Place legend box (if required). Default legend position is outside top right, level with plot window.

Place legend box (if required).

36. `void size_legend_box();`

37. `void size_legend_box();`

Calculate how big the legend box needs to be to hold the legend title and the data point markers (symbols or shapes), and any line marks showing lines used joining points, and any data series descriptor text(s).

Calculate how big the legend box needs to be to hold the title and any point markers (symbols or lines) and any series descriptor text.

38. `void transform_point(double & x, double & y);`

39. `void transform_point(double & x, double & y);`

< Scale & shift both X & Y to graph Cartesian coordinates.

< Scale & shift both X & Y to graph Cartesian coordinates.

svg_boxplot public public data members

1. `double text_plusminus_;`

Number of standard deviations used for text_plusminus text display.

Nominal factor of 1. (default) corresponds to 67% confidence limit, 2 (strictly 1.96) corresponds to 95% confidence limit.

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();

// public data members
svg_style axis_style_; // line widths and colors of X and Y axes.
svg_style box_style_; // line widths and colors of box.
double box_width_; // Width of boxplot box.
plot_point_style ext_outlier_; // Style (shape, color...) for marking 'extreme' outliers.
bool extreme_outlier_values_on_; // True if extreme outliers are to have their values labelled.
std::vector< double > extreme_outliers_; // Any data values that are judged extreme outliers.
svg_style max_whisker_style_; // Color and width etc of boxplot minimum 'whisker'.
double median_; // 2nd middle quartile.
svg_style median_style_; // line widths and colors of median marker.
plot_point_style mild_outlier_; // Style (shape, color...) for marking 'mild' outliers.
svg_style min_whisker_style_; // Color and width etc of boxplot minimum 'whisker'.
bool outlier_values_on_; // True if mild outliers are to have their values labelled.
std::vector< double > outliers_; // Any data values that are judged outliers.
double q1_; // 1st lower quartile.
double q3_; // 3rd upper quartile.
int quartile_definition_; // The definition of the quartile can be selected.
std::vector< double > series_; // Data series for the boxplot.
text_element series_info_; // information about the data series.
text_style series_style_; // Style (font etc) for text.
double text_margin_; // Margin (SVG units, default pixels) around text items.
value_style values_style_; // Style for data value labels.
double whisker_length_; // Length of boxplot 'whisker'.
double whisker_max_; // Maximum of whisker.
double whisker_min_; // Minimum of whisker.
};


```

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.

Returns: 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. See <https://www.w3schools.com/colors/> Example:
`my_plot.background_color(darkgreen).legend_background_color(lightgray).title_color(white)`

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>

#define BOOST_SVG_STYLE_DIAGNOSTICS for diagnostic output.

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, outside_window, 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 };

    static const fp_type aspect_ratio; // aspect_ratio is a guess at average height to width of font. used to estimate the
                                     // svg length of a title or header string from the font size. (This can only be quite approximate as varies on
                                     // type of font (narrow or bold) and the mix of characters widths (unless monospace font).)
    See https://www.w3.org/TR/SVG/text.html#GlyphsMetricshttps://www.w3.org/TR/SVG/text.html#InterfaceSVGTextContentElement
    text_style no_style; // Text style that uses all constructor defaults.
    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 &);

}
}

```

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/destroy
    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/destroy

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 (pixels).

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 = 1., 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 = 1., 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 &= "");

    // 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 rendered 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 x and/or y data point value(s) & optional uncertainty).

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 = "");

Unicode symbol(s) (letters, digits, squiggles etc), (default letter x).

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). See also `stroke_color` .`fill_color(red)`.`stroke_color(black)`

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). Example: `.shape(circlet).size(10).stroke_color(green).fill_color(red)`

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). `.stroke_color(black).fill_color(red)`

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)`;

Assign a `text_style` to data point marker symbol(s).

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 data 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!=(const svg_style &);
    bool operator==(const 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/destroy

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 size, font family, font weight, font 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 & = "", double = 0);
    text_style(const text_style &);
    text_style & operator=(const text_style &);

    // 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(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 &);
    text_style & text_length(double);
    double text_length() const;

    // 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 Unicode".
    int font_size_; // Font size (SVG units, default pixels).
    std::string stretch_; // Font stretch, examples: normal | wider | narrower. (Not supported by all browsers).
    std::string style_; // Font style, examples: normal | bold | italic | oblique.
    double text_length_; // Estimate of SVG length of text used to compress or expand into this width.
    std::string weight_; // Font weight examples: "bold", "normal".
};

};
```

Description

/*! text font family (for example: "Lucida Sans Unicode", "arial" ...). 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 ∞. Symbols are used to show data points and most shapes use Unicode. 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"
```

<http://www.fileformat.info/info/unicode/font/index.htm> provides a fuller listing of support by many fonts; those awarded 4 or 5 stars are probably most useful.

text_style public construct/copy/destruct

1.

```
text_style(int font_size = 12, const std::string & font = default_font,
           const std::string & weight = "", const std::string & style = "",
           const std::string & stretch = "",
           const std::string & decoration = "", double text_length = 0);
```

Estimated length of text string.

2.

```
text_style(const text_style & rhs);
```

text_style Copy constructor.

3.

```
text_style & operator=(const text_style & rhs);
```

Assignment operator=.

Returns: **text_style**& to make chainable.

text_style public member functions

1.

```
text_style & font_decoration(const std::string & s);
```

Set font decoration: "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. Example .font_decoration("underline");

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><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><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: **text_style**'s font family as **std::string**, for example: "Arial", "Times New Roman", "Verdana", "Lucida Sans Unicode" ..

5.

```
text_style & font_size(int i);
```

Set font size (svg units usually pixels) default 10.

Returns: reference to `text_style`& to make chainable.

6. `int font_size() const;`

Returns: `text_style`'s font size (svg units, usually pixels).

7. `text_style & font_stretch(const std::string & s);`

`font-stretch:"normal" | "wider" | "narrower"` Examples: `.font_stretch("wider")` Note that implementation by browsers varies.

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;`

Returns: font style, default normal. `font-style: normal | bold | italic | oblique.` Example: "normal" is default.

11. `text_style & font_weight(const std::string & s);`

`svg font-weight: "normal" | "bold" | "bolder" | "lighter" | "100" | "200" .. "900"` Examples: `.font_weight("bold");`
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).

15. `text_style & text_length(double);`

Set `text_length` to be rendered from an estimate of length from number of characters in string.

Returns: reference to `text_style` to make chainable.

16. `double text_length() const;`

Returns: `text_length` to be rendered from an estimate of length from number of characters in string.

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 SVG units (default 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 aspect_ratio

boost::svg::aspect_ratio — aspect_ratio is a guess at average height to width of font. used to estimate the svg length of a title or header string from the font size. (This can only be quite approximate as varies on type of font (narrow or bold) and the mix of characters widths (unless monospace font). See <https://www.w3.org/TR/SVG/text.html#GlyphsMetricshttps://www.w3.org/TR/SVG/text.html#InterfaceSVGTextContentElement>

Synopsis

```
// In header: <boost/svg_plot/svg_style.hpp>
static const fp_type aspect_ratio;
```

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;
```

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; std::cout << my_svg_style << std::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).

Example: `text_style` `ts(12, "Arial", "italic", "bold", "", "", 0); std::cout << t << std::endl;` Outputs: `text_style(18, "Arial", "italic", "bold", "", "", 0)`

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; std::cout << p << std::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 = aspect_ratio * 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 aspect_ratio = 0.7 is used as a good approximation.

Header <[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;

typedef unc< true > uncorr; // Uncertainties are NOT correlated. Uncorrelated is the normal case when uncertainties add.
typedef unc< false > uncurl;

static const double plusminus; // Nominal factor of 2 (strictly 1.96) corresponds to 95% confidence limit.

// Uncertainties ARE correlated. Correlated is an unusual case where the sum of uncertainties is fixed.
template<typename correlated>
std::ostream & operator<<(std::ostream & os, const unc< false > & u);
template<typename correlated>
std::ostream & operator<<(std::ostream & os, const unc< true > & u);
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 std::ostream &
operator<<(std::ostream & os,
            const std::pair< unc< false >, unc< false > > & u);

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 df = (std::numeric_limits< unsigned short int >::max)(),
        short unsigned ty = 0U);
    unc & operator=(const unc &);

    // public member functions
    short unsigned deg_free() const;
    void deg_free(short);
    bool operator<(const unc &) const;
    bool operator<=(const 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

// DETAIL

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).
 Returns: 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

Two helper functions to provide values and uncertainties as pairs.

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`.
 Returns: value as a `double`.
 Returns: 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.
6. 2014 Improved handling of values outside the plot window.

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, 536
a_path
 Struct a_path, 154

B

bar_style
 Class bar_style, 538
box_style
 Class box_style, 540

C

circle_element
 Class circle_element, 156
clip_path_element
 Class clip_path_element, 158
close_to
 Class template close_to, 148
c_path

Struct c_path, 155

E

ellipse_element

 Class ellipse_element, 160

G

g_element

 Class g_element, 163, 164

H

histogram_style

 Class histogram_style, 542

h_path

 Struct h_path, 168

L

line_element

 Class line_element, 170

l_path

 Struct l_path, 169

M

m_path

 Struct m_path, 172

P

path_element

 Class path_element, 173

path_point

 Struct a_path, 154

 Struct c_path, 155

 Struct h_path, 168

 Struct l_path, 169

 Struct m_path, 172

 Struct path_point, 178

 Struct q_path, 185

 Struct s_path, 192

 Struct t_path, 195

 Struct v_path, 207

 Struct z_path, 208

plot_line_style

 Class plot_line_style, 543

plot_point_style

 Class plot_point_style, 545

polygon_element

 Struct polygon_element, 180

polyline_element

 Class polyline_element, 183

poly_path_point

 Struct poly_path_point, 179

Q

curve_element

 Class curve_element, 186

q_path

Struct q_path, 185

R

rect_element

 Class rect_element, 189

S

smallest

 Class template smallest, 149

svg

 Class circle_element, 156

 Class clip_path_element, 158

 Class ellipse_element, 160

 Class g_element, 163

 Class line_element, 170

 Class path_element, 173

 Class polyline_element, 183

 Class quurve_element, 186

 Class rect_element, 189

 Class text_element, 196

 Class text_element_text, 200

 Class tspan_element, 202

Struct a_path, 154

Struct c_path, 155

Struct h_path, 168

Struct l_path, 169

Struct m_path, 172

Struct polygon_element, 180

Struct q_path, 185

Struct s_path, 192

Struct t_path, 195

Struct v_path, 207

Struct z_path, 208

svg_1d_plot

 Class svg_1d_plot, 215

svg_1d_plot_series

 Class svg_1d_plot_series, 308

svg_2d_plot

 Class svg_2d_plot, 312

svg_2d_plot_series

 Class svg_2d_plot_series, 419

svg_boxplot

 Class svg_boxplot, 425

svg_boxplot_series

 Class svg_boxplot_series, 521

svg_element

 Class circle_element, 156

 Class clip_path_element, 158

 Class ellipse_element, 160

 Class g_element, 163

 Class line_element, 170

 Class path_element, 173

 Class polyline_element, 183

 Class quurve_element, 186

 Class rect_element, 189

 Class svg_element, 193

 Class text_element, 196

Class tspan_element, 202
Struct polygon_element, 180
svg_style
 Class svg_style, 547
s_path
 Struct s_path, 192

T

text_element
 Class text_element, 196
text_element_text
 Class text_element_text, 200
text_parent
 Class text_element_text, 200
 Class text_parent, 201
 Class tspan_element, 202
text_style
 Class text_style, 550
ticks_labels_style
 Class ticks_labels_style, 553
tspan_element
 Class tspan_element, 202
t_path
 Struct t_path, 195

U

unc
 Class Meas, 95
 Class template unc, 118, 123, 562

V

value_style
 Class value_style, 556
v_path
 Struct v_path, 207

Z

z_path
 Struct z_path, 208

Function Index

Symbols

10
 1-D Auto scaling Examples, 36
 Function scale_axis, 140
 Function template scale_axis, 141, 142, 143
2D
 Header < boost/svg_plot/detail/functors.hpp >, 151
50
 Definitions of the Quartiles, 86
 Header < boost/svg_plot/quartile.hpp >, 210

A

abnormal

Class `svg_1d_plot`, 215
absolute
 Struct `polygon_element`, 180, 181
`add_g_element`
 Class `g_element`, 163, 164
`adjust_limits`
 Class `svg_1d_plot`, 215, 301
 Class `svg_2d_plot`, 312, 412
 Class `svg_boxplot`, 425, 515, 516
`alignment`
 Class `text_element`, 196, 197
`always`
 Function template `unc_of`, 129, 566
`area_fill`
 Class `bar_style`, 538, 539
 Class `plot_line_style`, 543
 Class `svg_2d_plot_series`, 419, 420
`author`
 Implementation and other notes, 577
`autoscale`
 1-D Autoscaling Various Containers Examples, 40
 Class `svg_1d_plot`, 215, 228
 Class `svg_2d_plot`, 312, 328
 Class `svg_boxplot`, 425, 438, 439
 Demonstration of using 2D data that includes information about its uncertainty, 76
 Header < `boost/quan/unc.hpp` >, 107
 Header < `boost/quan/uncs.hpp` >, 132
`autoscale_check_limits`
 1-D Auto scaling Examples, 31
 Class `svg_1d_plot`, 215, 228, 229
 Class `svg_2d_plot`, 312, 328, 329
 Class `svg_boxplot`, 425, 439
 Demonstration of using 2D data that includes information about its uncertainty, 76
`autoscale_plusminus`
 Class `svg_1d_plot`, 215, 229
 Class `svg_2d_plot`, 312, 329
 Class `svg_boxplot`, 425, 439
 Demonstration of using 2D data that includes information about its uncertainty, 76
`axes`
 Class `svg_1d_plot`, 229, 230
 Class `svg_2d_plot`, 329
 Class `svg_boxplot`, 440
`axes_on`
 Class `svg_1d_plot`, 215, 229, 230
 Class `svg_2d_plot`, 312, 329, 330
 Class `svg_boxplot`, 425, 440
`axis`
 1-D Auto scaling Examples, 31
 Class `svg_boxplot`, 483, 515
 Tutorial: 1D Gridlines & Axes - more examples, 22
`axis_color`
 Class `svg_boxplot`, 425, 440
 Class `svg_boxplot_series`, 521, 523
`axis_width`
 Class `svg_boxplot`, 425, 440
 Class `svg_boxplot_series`, 521, 523

B

background
 Simple Code Example, 58, 59

background_border_color
 Class `svg_1d_plot`, 215, 230
 Class `svg_2d_plot`, 312, 330
 Class `svg_boxplot`, 425, 440, 441

background_border_width
 Class `svg_1d_plot`, 215, 230, 231
 Class `svg_2d_plot`, 312, 330
 Class `svg_boxplot`, 425, 441

background_color
 Class `svg_1d_plot`, 215, 231
 Class `svg_2d_plot`, 312, 331
 Class `svg_boxplot`, 425, 441

bar_area_fill
 Class `svg_2d_plot_series`, 419, 420, 421

bar_color
 Class `svg_2d_plot_series`, 419, 421

bar_opt
 Class `bar_style`, 538, 539
 Class `svg_2d_plot_series`, 419, 421

bar_width
 Class `svg_2d_plot_series`, 419, 421

begin
 Class `svg_2d_plot_series`, 420

Bezier
 Class `path_element`, 176

bezier_on
 Class `plot_line_style`, 543, 544
 Class `svg_1d_plot_series`, 308, 309
 Class `svg_2d_plot_series`, 419, 421

boost_license_on
 Class `svg_1d_plot`, 215, 231
 Class `svg_2d_plot`, 312, 331
 Class `svg_boxplot`, 425, 441

border
 Class `svg_1d_plot`, 215, 230, 231, 237, 238, 240, 252, 303
 Class `svg_2d_plot`, 312, 328, 330, 331, 338, 339, 340, 341, 352, 414
 Class `svg_boxplot`, 425, 440, 441, 442, 446, 450, 451, 452, 453, 462, 465, 466, 479, 512, 517
 Class `ticks_labels_style`, 553

border_on
 Class `box_style`, 540

bottom
 Class `svg_1d_plot`, 215, 254
 Class `svg_2d_plot`, 354
 Class `svg_boxplot`, 467

box
 Class `box_style`, 540, 541
 Class `svg_1d_plot`, 215, 246, 306
 Class `svg_2d_plot`, 312, 346, 417
 Class `svg_boxplot`, 458, 462, 520

box_border
 Class `svg_boxplot`, 425, 442
 Class `svg_boxplot_series`, 521, 523

box_fill
 Class `svg_boxplot`, 425, 442

Class `svg_boxplot_series`, 521, 523, 524
`box_style`
 Class `box_style`, 540
 Class `svg_boxplot_series`, 521, 524
`box_width`
 Class `svg_boxplot`, 425, 442
 Class `svg_boxplot_series`, 521, 524

C

`c`
 Class `path_element`, 173, 174

`C`
 Class `path_element`, 173, 174

`calculate_plot_window`
 Class `svg_1d_plot`, 215, 232
 Class `svg_2d_plot`, 312, 331
 Class `svg_boxplot`, 425, 442

`calculate_quantiles`
 Class `svg_boxplot_series`, 521, 524

`calculate_transform`
 Class `svg_1d_plot`, 215, 232

`case`
 Class `svg_color`, 530

`circle`
 Class `g_element`, 163, 164

`circle_element`
 Class `circle_element`, 156

`class_id`
 Class `circle_element`, 156, 157
 Class `clip_path_element`, 158, 159
 Class `ellipse_element`, 160, 161
 Class `g_element`, 163, 164
 Class `line_element`, 170, 171
 Class `path_element`, 173, 174
 Class `polyline_element`, 183, 184
 Class `curve_element`, 186, 187
 Class `rect_element`, 189, 190
 Class `svg_element`, 193, 194
 Class `text_element`, 196, 197
 Class `tspan_element`, 202, 203
 Struct `polygon_element`, 180, 181

`clear`
 Class `g_element`, 163, 164

`clear_all`
 Class `svg_1d_plot`, 215, 301
 Class `svg_2d_plot`, 312, 412
 Class `svg_boxplot`, 425, 442

`clear_background`
 Class `svg_1d_plot`, 215, 301
 Class `svg_2d_plot`, 312, 412
 Class `svg_boxplot`, 425, 516

`clear_grids`
 Class `svg_1d_plot`, 215, 302
 Class `svg_2d_plot`, 312, 413
 Class `svg_boxplot`, 425, 516

`clear_legend`
 Class `svg_1d_plot`, 215, 302

Class `svg_2d_plot`, 312, 413
Class `svg_boxplot`, 425, 516
`clear_plot_background`
 Class `svg_1d_plot`, 215, 302
 Class `svg_2d_plot`, 312, 413
 Class `svg_boxplot`, 425, 516
`clear_points`
 Class `svg_1d_plot`, 215, 302
 Class `svg_2d_plot`, 312, 413
 Class `svg_boxplot`, 425, 517
`clear_title`
 Class `svg_1d_plot`, 215, 302
 Class `svg_2d_plot`, 312, 413, 414
 Class `svg_boxplot`, 425, 517
`clear_x_axis`
 Class `svg_1d_plot`, 215, 303
 Class `svg_2d_plot`, 312, 414
 Class `svg_boxplot`, 425, 517
`clear_y_axis`
 Class `svg_1d_plot`, 215, 303
 Class `svg_2d_plot`, 312, 414
 Class `svg_boxplot`, 425, 517
`clip_id`
 Class `circle_element`, 156, 157
 Class `clip_path_element`, 158, 159
 Class `ellipse_element`, 160, 161
 Class `g_element`, 163, 164, 165
 Class `line_element`, 170, 171
 Class `path_element`, 173, 174
 Class `polyline_element`, 183, 184
 Class `curve_element`, 186, 187
 Class `rect_element`, 189, 190
 Class `svg_element`, 193, 194
 Class `text_element`, 196, 198
 Class `tspan_element`, 202, 203
 Struct `polygon_element`, 180, 181
`close_to`
 Class `template` `close_to`, 148
`color`
 Class `axis_line_style`, 536, 537
 Class `bar_style`, 538, 539
 Class `plot_line_style`, 543, 544
 Class `svg_2d_plot`, 312, 328, 330, 331, 336, 337, 338, 340, 341, 343, 349, 350, 351, 352, 355, 356, 357, 361, 362, 365, 366, 367, 368, 369, 370, 371, 374, 375, 376, 378, 380, 382, 383, 384, 385, 389, 393, 394, 397, 398, 399, 400, 401, 403, 405, 406, 407, 409, 410, 411, 415
 Class `svg_2d_plot_series`, 419, 420, 421, 422, 424
 Class `svg_color`, 528, 529, 530
 Class `svg_style`, 547, 548, 549
`colors`
 1-D Data Values Examples, 43
 2-D Data Values Examples, 67
 Class `svg_color`, 530
 Colors, 6
 Demonstration of using 2D data that includes information about its uncertainty, 76
 Function operator!=, 533
 Function operator==, 534
`confidence`
 Class `svg_1d_plot`, 215, 232

Class `svg_2d_plot`, 312, 332
 Class `svg_boxplot`, 425, 442, 443
constant_to_rgb
 Function `constant_to_rgb`, 532
 Header <`boost/svg_plot/svg_color.hpp`>, 528
container
 Class `svg_boxplot`, 438, 447, 448, 464, 465, 477, 511
coordinate
 Class `svg_1d_plot`, 240, 246
 Class `svg_2d_plot`, 312, 341, 346, 347
 Class `svg_boxplot`, 453, 458, 459
coordinates
 Class `axis_line_style`, 536
 Demonstration of adding lines and curves, typically a least squares fit, 80
 Struct `m_path`, 172
 Struct `poly_path_point`, 178
coord_precision
 Class `svg_1d_plot`, 215, 232, 233, 262
 Class `svg_2d_plot`, 312, 332
 Class `svg_boxplot`, 425, 443
copyright_date
 Class `svg_1d_plot`, 215, 233
 Class `svg_2d_plot`, 312, 333
 Class `svg_boxplot`, 425, 443, 444
copyright_holder
 Class `svg_1d_plot`, 215, 233, 234
 Class `svg_2d_plot`, 312, 333
 Class `svg_boxplot`, 425, 444
curve
 Demonstration of adding lines and curves, typically a least squares fit, 80

D

data
 Class `circle_element`, 156
 Class `ellipse_element`, 160, 161
 Class `rect_element`, 189
 Class `svg_1d_plot`, 214, 215, 228, 230, 232, 234, 235, 236, 242, 243, 250, 251, 253, 254, 256, 262, 263, 265, 268, 269, 270, 280, 281, 285, 286, 287, 288, 292, 293, 295, 296, 297, 298, 301, 302, 303, 304, 305, 307
 Class `svg_boxplot`, 424, 425, 443, 444, 445, 446, 447, 455, 464, 465, 466, 467, 475, 476, 477, 480, 481, 482, 483, 492, 493, 497, 498, 500, 504, 505, 506, 507, 508, 509, 510, 516, 517, 518, 519, 520, 521
 Tutorial: 2D Special Features, 62
data_lines_width
 Class `svg_1d_plot`, 215, 234
 Class `svg_2d_plot`, 312, 333, 334
 Class `svg_boxplot`, 425, 444, 445
default_plot_point_style
 Header <`boost/svg_plot/svg_style.hpp`>, 534
definition
 Definitions of the Quartiles, 86
deg_free
 Class `Meas`, 95, 96, 97
 Class `template unc`, 118, 119, 120, 123, 125, 562, 563
derived
 Class `svg_1d_plot`, 215, 234
 Class `svg_2d_plot`, 312, 334
 Class `svg_boxplot`, 425, 445
description

Class `svg_1d_plot`, 215, 234, 235
Class `svg_2d_plot`, 312, 334, 335
Class `svg_boxplot`, 425, 445
Implementation and other notes, 577
deviation
 Class `Meas`, 95, 97
 Class `svg_1d_plot`, 249, 255, 261, 304
 Class `svg_2d_plot`, 350, 355, 356, 361, 362, 415
 Class `svg_boxplot`, 462, 463, 468, 469, 474, 518
 Class `template unc`, 118, 120, 123, 125, 561
digits
 Class `svg_1d_plot`, 232, 294
 Class `svg_2d_plot`, 332, 393, 410
 Class `svg_boxplot`, 443, 507
 Demonstration of using 1D data that includes information about its Uncertainty, 54
document
 Class `svg_1d_plot`, 233
 Class `svg_2d_plot`, 333
 Class `svg_boxplot`, 444
 Using Inkscape, 570
document_title
 Class `svg_1d_plot`, 215, 235
 Class `svg_2d_plot`, 312, 335
 Class `svg_boxplot`, 425, 446
 Implementation and other notes, 577
double
 Class `svg_1d_plot`, 250
draw_axes
 Class `svg_1d_plot`, 215, 235
draw_bars
 Class `svg_2d_plot`, 312, 335
draw_bezier_lines
 Class `svg_2d_plot`, 312, 335
draw_box
 Class `svg_boxplot`, 425, 446
draw_boxplot
 Class `svg_boxplot`, 425, 446
draw_histogram
 Class `svg_2d_plot`, 312, 335
draw_legend
 Class `svg_1d_plot`, 215, 303
 Class `svg_2d_plot`, 312, 414
 Class `svg_boxplot`, 425, 517
draw_line
 Class `svg_1d_plot`, 215, 236
 Class `svg_2d_plot`, 312, 336
 Class `svg_boxplot`, 425, 446
draw_median
 Class `svg_boxplot`, 425, 447
draw_note
 Class `svg_1d_plot`, 215, 236
 Class `svg_2d_plot`, 312, 336
 Class `svg_boxplot`, 425, 447
draw_plot_curve
 Class `svg_1d_plot`, 215, 237
 Class `svg_2d_plot`, 312, 337
 Class `svg_boxplot`, 425, 447, 448
draw_plot_line

Class `svg_1d_plot`, 215, 237
Class `svg_2d_plot`, 312, 337
Class `svg_boxplot`, 425, 448
`draw_plot_lines`
 Class `svg_2d_plot`, 312, 337
`draw_plot_point`
 Class `svg_1d_plot`, 215, 303, 304
 Class `svg_2d_plot`, 312, 414, 415
 Class `svg_boxplot`, 425, 518
`draw_plot_points`
 Class `svg_2d_plot`, 312, 337
`draw_plot_point_value`
 Class `svg_1d_plot`, 215, 304
 Class `svg_2d_plot`, 312, 415
 Class `svg_boxplot`, 425, 518
`draw_plot_point_values`
 Class `svg_1d_plot`, 215, 304, 305
 Class `svg_2d_plot`, 312, 415, 416
 Class `svg_boxplot`, 425, 519
 Class `value_style`, 556
`draw_straight_lines`
 Class `svg_2d_plot`, 312, 338
 Class `svg_2d_plot_series`, 419, 420
`draw_title`
 Class `svg_1d_plot`, 215, 305
 Class `svg_2d_plot`, 312, 416
 Class `svg_boxplot`, 425, 448
`draw_whiskers`
 Class `svg_boxplot`, 425, 448
`draw_x_axis`
 Class `svg_1d_plot`, 215, 306
 Class `svg_2d_plot`, 312, 417
 Class `svg_boxplot`, 425, 448
`draw_x_axis_label`
 Class `svg_1d_plot`, 215, 306
 Class `svg_2d_plot`, 312, 417
 Class `svg_boxplot`, 425, 449
`draw_x_major_tick`
 Class `svg_1d_plot`, 215, 306
 Class `svg_2d_plot`, 312, 417
 Class `svg_boxplot`, 425, 449, 520
`draw_x_minor_tick`
 Class `svg_1d_plot`, 215, 306
 Class `svg_2d_plot`, 312, 417
 Class `svg_boxplot`, 425, 520
`draw_y_axis`
 Class `svg_2d_plot`, 312, 338
 Class `svg_boxplot`, 425, 449
`draw_y_axis_label`
 Class `svg_2d_plot`, 312, 338
 Class `svg_boxplot`, 425, 449
`draw_y_major_tick`
 Class `svg_2d_plot`, 312, 338
 Class `svg_boxplot`, 425, 449
`draw_y_minor_tick`
 Class `svg_2d_plot`, 312, 338
 Class `svg_boxplot`, 425, 449
`dx`

Class tspan_element, 202, 203

dy

Class tspan_element, 202, 203

E

element

Class text_element, 199

Class tspan_element, 203, 204, 205

ellipse

Class g_element, 163, 165

ellipse_element

Class ellipse_element, 160

epsilon

Header < boost/svg_plot/detail/fp_compare.hpp >, 147

equal

Class template close_to, 149

equality

Class text_style, 552

extreme_outlier_color

Class svg_boxplot, 425, 449

Class svg_boxplot_series, 521, 524

extreme_outlier_fill

Class svg_boxplot, 425, 449

Class svg_boxplot_series, 521, 524, 525

extreme_outlier_shape

Class svg_boxplot, 425, 449, 450

Class svg_boxplot_series, 521, 525

extreme_outlier_size

Class svg_boxplot, 425, 450

Class svg_boxplot_series, 521, 525

extreme_outlier_values_on

Class svg_boxplot, 425, 450

F

f

Simple Example, 84

Tutorial: 1D More Layout Examples, 20

Tutorial: Fuller Layout Example, 60

few

1-D Vector Example, 14

Demonstration of using 1D data that includes information about its Uncertainty, 54

file

Class svg_2d_plot, 362

Using Inkscape, 570

fill

Class box_style, 540, 541

Class svg_boxplot, 442, 446, 449, 452, 453, 461, 462, 463, 468, 469, 474, 518

Class svg_boxplot_series, 523, 524, 525, 527

fill_color

Class plot_point_style, 545, 546

Class svg_1d_plot_series, 308, 309

Class svg_2d_plot_series, 419, 421

Class svg_style, 547, 548

Class tspan_element, 202, 203, 204

fill_on

Class box_style, 540, 541

Class path_element, 173, 174, 175

Class `svg_style`, 547, 548
 Class `tspan_element`, 202, 204
font
 Class `svg_1d_plot`, 215, 232, 241, 242, 244, 245, 257, 258, 259, 260, 261, 271, 272, 291, 292, 293, 295, 296, 304
 Class `svg_2d_plot`, 312, 342, 343, 345, 346, 358, 359, 360, 361, 371, 372, 389, 390, 392, 394, 401, 402, 403, 409, 410, 411, 415
 Class `svg_boxplot`, 425, 453, 454, 455, 457, 458, 470, 471, 472, 473, 484, 485, 503, 504, 506, 508, 513, 518
 Class `text_style`, 549, 550, 551, 552
 Fonts, 8
 Global aspect_ratio, 146, 557
 Header <boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp>, 145
 Header <boost/svg_plot/svg_style.hpp>, 534
font_decoration
 Class `text_style`, 550, 551
font_family
 Class `text_style`, 550, 551
 Class `tspan_element`, 202, 204
font_size
 Class `text_style`, 550, 551, 552
 Class `tspan_element`, 202, 204
font_stretch
 Class `text_style`, 550, 552
font_style
 Class `text_style`, 550, 552
 Class `tspan_element`, 202, 204
font_weight
 Class `text_style`, 550, 552
 Class `tspan_element`, 202, 204, 205
fpt_abs
 Function template `fpt_abs`, 150
 Header <boost/svg_plot/detail/fp_compare.hpp>, 147
freedom
 2-D Data Values Examples, 67
 Class template `unc`, 564
 Function template `operator<<`, 565
from
 Class `line_element`, 169, 170, 171
 Class `curve_element`, 186, 188
functor
 Class `svg_2d_plot`, 351

G

g
 Class `g_element`, 163, 165
 Simple Example, 84
 Tutorial: 1D More Layout Examples, 20
 Tutorial: Fuller Layout Example, 60
generate_text
 Class `text_element`, 196, 197
Graphic
 Header <boost/svg_plot/detail/auto_axes.hpp>, 135
 Header <boost/svg_plot/svg_1d_plot.hpp>, 214
 Header <boost/svg_plot/svg_boxplot.hpp>, 424
g_element
 Class `g_element`, 163, 164

H

h

Class path_element, 173, 175
Tutorial: 1D More Layout Examples, 20
Tutorial: Fuller Layout Example, 60

H

Class path_element, 173, 175

header

Class svg_1d_plot, 303
Class svg_2d_plot, 414
Class svg_boxplot, 517

heat_flow_data

Real-life Heat flow data, 48

height

Class rect_element, 189, 190
Class svg_1d_plot, 239, 255, 300, 301
Class svg_2d_plot, 340, 355, 408
Class svg_boxplot, 452, 468

hexagon

Class g_element, 163, 165

histogram

Class histogram_style, 541, 542
Class svg_2d_plot_series, 419, 421

Hoaglin

Definitions of the Quartiles, 86

Hoaglin4

Definitions of the Quartiles, 86

Hoaglin5

Definitions of the Quartiles, 86

Hoaglin6

Definitions of the Quartiles, 86

Hoaglin7

Definitions of the Quartiles, 86

Hoaglin8

Definitions of the Quartiles, 86

horizontal

Class ellipse_element, 160
Class path_element, 175
Class svg_2d_plot, 404

I

id

Class circle_element, 156, 157
Class clip_path_element, 158, 159
Class ellipse_element, 160, 161
Class g_element, 163, 165
Class line_element, 170, 171
Class path_element, 173, 175
Class polyline_element, 183, 184
Class curve_element, 186, 187
Class rect_element, 189, 190
Class svg_element, 193, 194
Class text_element, 196, 198
Class tspan_element, 202, 205
Struct polygon_element, 180, 181, 182

inequality

Class text_style, 552

interval

Class svg_1d_plot, 278

Class `svg_2d_plot`, 312, 377
 Class `svg_boxplot`, 491
intervals
 Class `svg_1d_plot`, 232
 Class `svg_2d_plot`, 332
 Class `svg_boxplot`, 425, 442, 443
is
 Class `svg_1d_plot`, 232
 Class `template unc`, 118, 123
isfinite
 Function template `isfinite`, 136
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135
isnan
 Showing data values at Numerical Limits, 92
is_blank
 Class `svg_color`, 529, 530
 Function `is_blank`, 532
 Header <`boost/svg_plot/svg_color.hpp`>, 528
is_in_window
 Class `svg_2d_plot`, 312, 340
is_license
 Implementation and other notes, 577

L

l
 Class `path_element`, 173, 175
L
 Class `path_element`, 173, 175
label
 Class `svg_1d_plot`, 215, 232, 262, 266, 271, 272, 273, 274, 276, 277, 278, 290, 291, 292, 293, 294, 295, 299, 300, 305, 306
 Class `svg_2d_plot`, 312, 335, 338, 365, 366, 371, 372, 373, 374, 376, 377, 389, 390, 392, 393, 399, 400, 401, 402, 403, 404, 410, 412, 416, 417
 Class `svg_boxplot`, 425, 449, 478, 483, 484, 485, 486, 487, 489, 490, 503, 504, 506, 507, 512, 513, 514, 519
 Class `ticks_labels_style`, 553, 554
labels
 Class `svg_1d_plot`, 278
 Class `svg_2d_plot`, 312, 404
 Class `svg_boxplot`, 490
label_length
 Class `ticks_labels_style`, 553, 554
label_on
 Class `axis_line_style`, 536, 537
label_units_on
 Class `axis_line_style`, 536, 537
left
 Class `svg_boxplot`, 514
legend_background_color
 Class `svg_1d_plot`, 215, 239
 Class `svg_2d_plot`, 312, 340
 Class `svg_boxplot`, 425, 452
legend_border_color
 Class `svg_1d_plot`, 215, 240
 Class `svg_2d_plot`, 312, 340, 341
 Class `svg_boxplot`, 425, 452, 453
legend_box_fill_on
 Class `svg_1d_plot`, 215, 240
 Class `svg_2d_plot`, 312, 341

Class `svg_boxplot`, 425, 453
legend_color
 Class `svg_1d_plot`, 215, 240, 241
 Class `svg_2d_plot`, 312, 341
 Class `svg_boxplot`, 425, 453
legend_font_family
 Class `svg_1d_plot`, 215, 241
 Class `svg_2d_plot`, 312, 342
 Class `svg_boxplot`, 425, 453, 454
legend_font_size
 Class `svg_1d_plot`, 215, 241
 Class `svg_2d_plot`, 312, 342
 Class `svg_boxplot`, 425, 454
legend_font_weight
 Class `svg_1d_plot`, 215, 241, 242
 Class `svg_2d_plot`, 312, 342, 343
 Class `svg_boxplot`, 425, 454
legend_header_font_size
 Class `svg_1d_plot`, 215, 242
 Class `svg_2d_plot`, 312, 343
 Class `svg_boxplot`, 425, 455
legend_lines
 Class `svg_1d_plot`, 215, 242, 243
 Class `svg_2d_plot`, 312, 343
 Class `svg_boxplot`, 425, 455
legend_on
 Class `svg_1d_plot`, 215, 243
 Class `svg_2d_plot`, 312, 343, 344
 Class `svg_boxplot`, 425, 455, 456
legend_outside
 Class `svg_1d_plot`, 215, 243
 Class `svg_2d_plot`, 312, 344
 Class `svg_boxplot`, 425, 456, 456
legend_place
 Class `svg_1d_plot`, 215, 243, 244
 Class `svg_2d_plot`, 312, 344, 345
 Class `svg_boxplot`, 425, 456, 457
legend_text_font_size
 Class `svg_1d_plot`, 215, 244
 Class `svg_2d_plot`, 312, 345
 Class `svg_boxplot`, 425, 457
legend_text_font_weight
 Class `svg_1d_plot`, 215, 244
 Class `svg_2d_plot`, 312, 345
 Class `svg_boxplot`, 425, 457
legend_title
 Class `svg_1d_plot`, 215, 245
 Class `svg_2d_plot`, 312, 345, 346
 Class `svg_boxplot`, 425, 457, 458
legend_title_font_size
 Class `svg_1d_plot`, 215, 245
 Class `svg_2d_plot`, 312, 346
 Class `svg_boxplot`, 425, 458
legend_title_font_weight
 Class `svg_1d_plot`, 215, 245
 Class `svg_2d_plot`, 312, 346
 Class `svg_boxplot`, 425, 458
legend_top_left

Class `svg_1d_plot`, 215, 246
Class `svg_2d_plot`, 312, 346, 347
Class `svg_boxplot`, 425, 458, 459
`legend_width`
 Class `svg_1d_plot`, 215, 246
 Class `svg_2d_plot`, 312, 347
 Class `svg_boxplot`, 425, 459
`length`
 Class `svg_boxplot`, 474
 Class `svg_boxplot_series`, 528
 Class `tspan_element`, 206
`license`
 Class `svg_1d_plot`, 215, 246, 247
 Class `svg_2d_plot`, 312, 347
 Class `svg_boxplot`, 425, 459
`license_attribution`
 Class `svg_1d_plot`, 215, 247
 Class `svg_2d_plot`, 312, 348
 Class `svg_boxplot`, 425, 460
 Implementation and other notes, 577
`license_commercialuse`
 Class `svg_1d_plot`, 215, 247
 Class `svg_2d_plot`, 312, 348
 Class `svg_boxplot`, 425, 460
 Implementation and other notes, 577
`license_distribution`
 Class `svg_1d_plot`, 215, 247
 Class `svg_2d_plot`, 312, 348
 Class `svg_boxplot`, 425, 460
 Implementation and other notes, 577
`license_on`
 Class `svg_1d_plot`, 215, 247, 248
 Class `svg_2d_plot`, 312, 348, 349
 Class `svg_boxplot`, 425, 460, 461
`license_reproduction`
 Class `svg_1d_plot`, 215, 248
 Class `svg_2d_plot`, 312, 349
 Class `svg_boxplot`, 425, 461
 Implementation and other notes, 577
`limits_count`
 Class `svg_2d_plot_series`, 419, 422
`limit_color`
 Class `svg_1d_plot`, 215, 248, 249
 Class `svg_2d_plot`, 312, 349
 Class `svg_boxplot`, 425, 461
`limit_fill_color`
 Class `svg_1d_plot`, 215, 249
 Class `svg_2d_plot`, 312, 349, 350
 Class `svg_boxplot`, 425, 461, 462
`line`
 Class `axis_line_style`, 536, 538
 Class `bar_style`, 538
 Class `box_style`, 540
 Class `g_element`, 163, 165
 Class `plot_line_style`, 543, 544
 Class `svg_1d_plot`, 277, 278
 Class `svg_2d_plot`, 404
 Class `svg_2d_plot_series`, 420, 421, 422

Class `svg_boxplot`, 479, 490, 512
Class `value_style`, 556
Demonstration of adding lines and curves, typically a least squares fit, 80
lines
 Class `svg_2d_plot`, 331
line_color
 Class `svg_1d_plot_series`, 308, 309
 Class `svg_2d_plot_series`, 419, 422
line_element
 Class `line_element`, 170
line_on
 Class `plot_line_style`, 543, 544
 Class `svg_1d_plot_series`, 308, 309
 Class `svg_2d_plot_series`, 419, 422
line_style
 Class `svg_2d_plot_series`, 419, 422
line_width
 Class `svg_1d_plot_series`, 308, 309
 Class `svg_2d_plot_series`, 419, 422
longest_label
 Class `ticks_labels_style`, 553, 554
l_or_r
 Header <`boost/svg_plot/show_1d_settings.hpp`>, 212

M

M
 Class `path_element`, 173, 176
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`, 553, 554
margin
 Class `box_style`, 540, 541
marker
 Class `svg_1d_plot_series`, 308, 309, 310
 Class `svg_2d_plot_series`, 419, 420, 421, 422, 423, 424
 Function operator<<, 559
markers
 Class `svg_1d_plot`, 307
 Class `svg_2d_plot`, 418
 Class `svg_boxplot`, 520
maximum
 Class `svg_1d_plot`, 215, 253, 254, 263, 265, 280, 281, 286, 301
 Class `svg_2d_plot`, 312, 353, 354, 363, 364, 379, 380, 385, 386, 397, 398, 405, 407, 412
 Class `svg_boxplot`, 425, 448, 464, 466, 467, 476, 477, 492, 493, 499, 511, 515, 516
maxof3
 Header <`boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp`>, 145
max_whisker_color
 Class `svg_boxplot_series`, 521, 525
max_whisker_width
 Class `svg_boxplot_series`, 521, 525

Meas

- Class Meas, 95
- median_color
 - Class svg_boxplot, 425, 462
 - Class svg_boxplot_series, 521, 525
- median_style
 - Class svg_boxplot_series, 521, 526
- median_width
 - Class svg_boxplot, 425, 462
 - Class svg_boxplot_series, 521, 526
- middle
 - Class svg_1d_plot, 215
- min_value
 - Header < boost/svg_plot/detail/fp_compare.hpp >, 147
- min_whisker_color
 - Class svg_boxplot_series, 521, 526
- min_whisker_width
 - Class svg_boxplot_series, 521, 526
- mnnmx
 - Function template mnnmx, 137
 - Header < boost/svg_plot/detail/auto_axes.hpp >, 135
- my_blank
 - Class svg_color, 529
- my_color
 - Function operator<<, 533
- my_plot
 - Class svg_1d_plot, 230
 - Class svg_2d_plot, 330
- m_path
 - Struct m_path, 172

N

- not
 - Class svg_1d_plot_series, 308
 - Plotting Graphs in SVG format., 2
- NOT
 - Class svg_1d_plot, 306
 - Class svg_2d_plot, 417
 - Class svg_boxplot, 520

O

- of
 - Class template unc, 564
- one_sd_color
 - Class svg_1d_plot, 215, 249
 - Class svg_2d_plot, 312, 350
 - Class svg_boxplot, 425, 462, 463
- operator
 - Class template close_to, 148, 149
 - Class template smallest, 149, 150
 - Struct template lessAbs, 110
- optional
 - 1-D Vector Example, 14
- outFmtFlags
 - Function outFmtFlags, 135, 212, 213
 - Header < boost/quan/xiostream.hpp >, 132
 - Header < boost/svg_plot/show_1d_settings.hpp >, 212

outlier_color
 Class `svg_boxplot`, 425, 463
 Class `svg_boxplot_series`, 521, 526
outlier_fill
 Class `svg_boxplot`, 425, 463
 Class `svg_boxplot_series`, 521, 527
outlier_shape
 Class `svg_boxplot`, 425, 463
 Class `svg_boxplot_series`, 521, 527
outlier_size
 Class `svg_boxplot`, 425, 463
 Class `svg_boxplot_series`, 521, 527
outlier_style
 Class `svg_boxplot`, 425, 464
 Class `svg_boxplot_series`, 521, 527
outlier_values_on
 Class `svg_boxplot`, 425, 464

P

P
 Class `polyline_element`, 183, 185
 Struct `polygon_element`, 180, 182
p
 Function operator<<, 210
 Struct `polygon_element`, 182
p0
 Function operator<<, 209
part
 1-D Auto scaling Examples, 31
 1-D Axis Scaling, 26
parts
 Function template `uncs_of`, 130, 567
path
 Class `g_element`, 163, 166
path_element
 Class `path_element`, 173
pentagon
 Class `g_element`, 163, 166
place_legend_box
 Class `svg_1d_plot`, 215, 306
 Class `svg_2d_plot`, 312, 417
 Class `svg_boxplot`, 425, 520
plot
 Class `svg_1d_plot`, 215, 250, 251, 256, 257
 Class `svg_2d_plot`, 312, 350, 351, 356, 357
 Class `svg_boxplot`, 425, 464, 465
 Colors, 6
 Function template `scale_axis`, 141
 Showing data values at Numerical Limits, 92
 Simple Code Example, 58
 Tutorial: 2D Special Features, 62
plot_background_color
 Class `svg_1d_plot`, 215, 251
 Class `svg_2d_plot`, 312, 351, 352
 Class `svg_boxplot`, 425, 465
plot_border_color
 Class `svg_1d_plot`, 215, 252

Class `svg_2d_plot`, 312, 352
Class `svg_boxplot`, 425, 465
`plot_border_width`
 Class `svg_1d_plot`, 215, 252
 Class `svg_2d_plot`, 312, 352
 Class `svg_boxplot`, 425, 465
`plot_point_style`
 Class `plot_point_style`, 545
`plot_window_x_right`
 Class `svg_1d_plot`, 215, 254
 Class `svg_2d_plot`, 312, 354
 Class `svg_boxplot`, 425, 467
`plot_window_y`
 Class `svg_1d_plot`, 215, 254
 Class `svg_2d_plot`, 312, 354
 Class `svg_boxplot`, 425, 467
`plot_window_y_bottom`
 Class `svg_1d_plot`, 215, 254
 Class `svg_2d_plot`, 312, 354
 Class `svg_boxplot`, 425, 467
`plot_window_y_top`
 Class `svg_1d_plot`, 215, 254
 Class `svg_2d_plot`, 312, 355
 Class `svg_boxplot`, 425, 468
`point`
 Class `plot_point_style`, 545, 546
 Class `curve_element`, 186, 188
 Class `svg_1d_plot`, 304
 Class `svg_2d_plot`, 415
 Class `svg_boxplot`, 518
 Struct `h_path`, 168
 Struct `polygon_element`, 182
 Struct `v_path`, 207
`points`
 Class `svg_2d_plot`, 338
 Function `operator<<`, 210
`point_font_decoration`
 Class `svg_2d_plot_series`, 419, 422
`point_font_family`
 Class `svg_2d_plot_series`, 419, 422, 423
`point_font_stretch`
 Class `svg_2d_plot_series`, 419, 423
`point_font_style`
 Class `svg_2d_plot_series`, 419, 423
`point_font_weight`
 Class `svg_2d_plot_series`, 419, 423
`point_style`
 Class `svg_2d_plot_series`, 419, 423
`polygon`
 Class `g_element`, 163, 166
 Struct `polygon_element`, 180
`polygon_element`
 Struct `polygon_element`, 180
`polyline`
 Class `g_element`, 163, 166
`polyline_element`
 Class `polyline_element`, 183
`poly_path_point`

Struct poly_path_point, 179
position
 Class axis_line_style, 536, 537
 Class svg_1d_plot, 266, 267
 Class svg_2d_plot, 366
precision
 Class template unc, 118, 123
 Header < boost/svg_plot/detail/functors.hpp >, 151
 Real-life Heat flow data, 48
push_back
 Class g_element, 163, 166

Q

q
 Class path_element, 173, 176
Q
 Class path_element, 173, 176
quantile
 Definitions of the Quartiles, 86
 Function quantile, 211, 212
 Header < boost/svg_plot/quantile.hpp >, 210
quartile_definition
 Class svg_boxplot, 425, 468
 Class svg_boxplot_series, 521, 527
curve_element
 Class curve_element, 186

R

r
 Function operator<<, 209
range
 Class svg_2d_plot, 407
range_mx
 Function template range_mx, 137
 Header < boost/svg_plot/detail/auto_axes.hpp >, 135
rect
 Class g_element, 163, 166
rect_element
 Class rect_element, 189
rect_elements
 Class rect_element, 190
red
 Class svg_color, 529, 530
RGB
 1-D Auto scaling Examples, 31
 1-D Data Values Examples, 43
 2-D Data Values Examples, 67
 Function operator<<, 533
rhombus
 Class g_element, 163, 167
right
 Class svg_1d_plot, 253
 Class svg_2d_plot, 353
 Class svg_boxplot, 466
 Class ticks_labels_style, 554
rotation
 Class svg_1d_plot, 297

Class `svg_2d_plot`, 396
 Class `svg_boxplot`, 510
 Class `text_element`, 196, 198
 Class `tspan_element`, 202, 205
`rounddown10`
 Function `rounddown10`, 138
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135
`rounddown2`
 Function `rounddown2`, 138
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135
`rounddown5`
 Function `rounddown5`, 138
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135
`roundup10`
 Function `roundup10`, 139
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135
`roundup2`
 Function `roundup2`, 139
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135
`roundup5`
 Function `roundup5`, 139
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135

S

`s`
 2-D Autoscaling Examples, 73
 Class `path_element`, 173, 176
`S`
 Class `path_element`, 173, 176
`safe_fpt_division`
 Function template `safe_fpt_division`, 150
 Header <`boost/svg_plot/detail/fp_compare.hpp`>, 147
`same`
 Class `svg_1d_plot`, 241
 Class `svg_2d_plot`, 335, 342
 Class `svg_boxplot`, 454
`SAS`
 Definitions of the Quartiles, 86
`scale_axis`
 Class `svg_2d_plot`, 312
 Function `scale_axis`, 140
 Function template `scale_axis`, 141, 142
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135
`scientific`
 Class `svg_1d_plot`, 262
`series`
 1-D STL Containers Examples, 15
 Class `svg_1d_plot`, 215, 242, 250, 251, 262, 265, 303, 307
 Class `svg_1d_plot_series`, 307, 308, 310
 Class `svg_2d_plot`, 312, 335, 337, 338, 343, 350, 351, 365, 397, 398, 414, 418
 Class `svg_2d_plot_series`, 419, 420, 421, 422, 423, 424
 Class `svg_boxplot`, 424, 425, 447, 455, 464, 465, 477, 517, 520
 Function template `scale_axis`, 141, 142, 143
`series_count`
 Class `svg_1d_plot_series`, 308, 310
`series_limits_count`
 Class `svg_1d_plot_series`, 308, 310

set
Demonstration of using 1D data that includes information about its Uncertainty, 54
Demonstration of using 2D data that includes information about its uncertainty, 76

set_ids
Class `svg_1d_plot`, 215, 255
Class `svg_2d_plot`, 312, 355
Header <`boost/svg_plot/detail/svg_boxplot_detail.hpp`>, 151

shape
Class `plot_point_style`, 545, 546
Class `svg_1d_plot_series`, 308, 310
Class `svg_2d_plot_series`, 419, 423

show
Function template `show`, 143
Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135

shown
Class `svg_1d_plot`, 297
Class `svg_2d_plot`, 312, 395, 397, 411
Class `svg_boxplot`, 509

show_1d_plot_settings
1-D Auto scaling Examples, 31
1-D Data Values Examples, 43
Function `show_1d_plot_settings`, 213
Header <`boost/svg_plot/show_1d_settings.hpp`>, 212
Tutorial: 2D Special Features, 62

show_2d_plot_settings
2-D Data Values Examples, 67
Function `show_2d_plot_settings`, 214
Header <`boost/svg_plot/show_2d_settings.hpp`>, 213

show_all
Function template `show_all`, 144
Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135

sin
Simple Example, 84
To Do List, 582

size
Class `g_element`, 163, 167
Class `plot_point_style`, 545, 546
Class `svg_1d_plot`, 215, 238, 239, 242, 255, 272, 287, 291, 292, 293, 301
Class `svg_1d_plot_series`, 308, 310
Class `svg_2d_plot`, 312, 339, 340, 343, 355, 372, 386, 390, 392, 408
Class `svg_2d_plot_series`, 419, 423
Class `svg_boxplot`, 425, 451, 452, 455, 468, 484, 485, 499, 504, 506, 515
Class template `close_to`, 148, 149
Class template `smallest`, 149, 150
Class `text_style`, 550, 551, 552
Using Inkscape, 570

size_legend_box
Class `svg_1d_plot`, 215, 307
Class `svg_2d_plot`, 312, 418
Class `svg_boxplot`, 425, 520

smallest
Class template `smallest`, 149

sqrt
Tutorial: 1D More Layout Examples, 20
Tutorial: Fuller Layout Example, 60

strength
Class template `close_to`, 148, 149

string

Class `svg_1d_plot`, 236
Class `svg_2d_plot`, 336
Class `svg_boxplot`, 447
Function `operator<<`, 559
Function `strip_e0s`, 311
`string_svg_length`
 Function `string_svg_length`, 560
 Header <`boost/svg_plot/svg_style.hpp`>, 534
`strip_e0s`
 Function `strip_e0s`, 311
 Header <`boost/svg_plot/svg_1d_plot.hpp`>, 214
`stroke`
 Class `box_style`, 540, 541
`stroke_color`
 Class `plot_point_style`, 545, 546
 Class `svg_1d_plot_series`, 308, 310
 Class `svg_2d_plot_series`, 419, 424
 Class `svg_style`, 547, 548
`stroke_on`
 Class `svg_style`, 547, 548
`stroke_width`
 Class `svg_style`, 547, 549
`style`
 Class `circle_element`, 156, 157
 Class `clip_path_element`, 158, 159
 Class `ellipse_element`, 160, 161, 162
 Class `g_element`, 163, 167
 Class `line_element`, 170, 171
 Class `path_element`, 173, 176
 Class `plot_point_style`, 545, 546
 Class `polyline_element`, 183, 185
 Class `curve_element`, 186, 188
 Class `rect_element`, 189, 191
 Class `svg_boxplot`, 425
 Class `svg_element`, 193, 195
 Class `text_element`, 196, 198
 Class `ticks_labels_style`, 553
 Class `tspan_element`, 202, 205, 206
 Class `value_style`, 556
 Function `operator<<`, 209
 Implementation and other notes, 577
 Struct `polygon_element`, 180, 182
`suffix`
 Class `decor_printer`, 105
 Class `svg_1d_plot`, 289
 Class `svg_2d_plot`, 388, 408
 Class `svg_boxplot`, 501
`svg_2d_plot`
 Class `svg_2d_plot`, 312
`svg_color`
 Class `svg_color`, 529
 Function `constant_to_rgb`, 532
`svg_styles`
 Class `svg_style`, 548
`symbol`
 Class `plot_point_style`, 545, 546, 547
`symbols`
 Class `plot_point_style`, 545, 546, 547

Class `svg_1d_plot_series`, 308, 310

T

t

Class `path_element`, 173, 176
Function `operator<<`, 209

T

Class `path_element`, 173, 177

text

Class `g_element`, 163, 167
Class `svg_1d_plot`, 241, 244, 303, 306, 307
Class `svg_2d_plot`, 312, 342, 345, 414, 417, 418
Class `svg_boxplot`, 454, 457, 517, 520
Class `text_element`, 196, 198, 199
Class `tspan_element`, 202, 205, 206
Header <`boost/svg_plot/svg_style.hpp`>, 534

textstyle

Class `text_element`, 196, 199
Class `tspan_element`, 202, 206

text_element

Class `text_element`, 196

text_element_text

Class `text_element_text`, 200

text_length

Class `text_style`, 550, 553
Class `tspan_element`, 202, 206

text_style

Class `svg_1d_plot`, 261
Class `svg_2d_plot`, 312, 361
Class `svg_boxplot`, 473
Class `text_style`, 550

the

Class `path_element`, 173
Class `polyline_element`, 183
Tutorial: 1D Autoscale with Multiple Containers, 18

ticks

Class `svg_1d_plot`, 279, 280, 281, 283, 284
Class `svg_2d_plot`, 312, 378, 379, 380, 382, 383
Class `svg_boxplot`, 491, 492, 493, 495, 496
Class `ticks_labels_style`, 553

title

Class `svg_1d_plot`, 215, 232, 235, 240, 241, 242, 244, 245, 250, 251, 256, 257, 258, 259, 260, 261, 301, 302, 303, 305, 307
Class `svg_2d_plot`, 312, 328, 335, 341, 342, 343, 345, 346, 350, 351, 356, 357, 358, 359, 360, 361, 412, 413, 414, 416, 418
Class `svg_boxplot`, 425, 442, 445, 446, 453, 454, 455, 457, 458, 464, 465, 469, 470, 471, 472, 473, 474, 485, 517, 520
Class `svg_boxplot_series`, 521, 523, 528

titles

Demonstration of using 1D data that includes information about its Uncertainty, 54

title_color

Class `svg_1d_plot`, 215, 257
Class `svg_2d_plot`, 312, 357
Class `svg_boxplot`, 425, 469, 470

title_font_alignment

Class `svg_1d_plot`, 215, 257
Class `svg_2d_plot`, 312, 357
Class `svg_boxplot`, 425, 470

title_font_decoration

Class `svg_1d_plot`, 215, 257, 258

Class `svg_2d_plot`, 312, 358
 Class `svg_boxplot`, 425, 470
title_font_family
 Class `svg_1d_plot`, 215, 258
 Class `svg_2d_plot`, 312, 358
 Class `svg_boxplot`, 425, 470, 471
title_font_size
 Class `svg_1d_plot`, 215, 259
 Class `svg_2d_plot`, 312, 359
 Class `svg_boxplot`, 425, 471, 472
title_font_stretch
 Class `svg_1d_plot`, 215, 259
 Class `svg_2d_plot`, 312, 359, 360
 Class `svg_boxplot`, 425, 472
title_font_style
 Class `svg_1d_plot`, 215, 260
 Class `svg_2d_plot`, 312, 360
 Class `svg_boxplot`, 425, 472
title_on
 Class `svg_1d_plot`, 215, 260, 261
 Class `svg_2d_plot`, 312, 360, 361
 Class `svg_boxplot`, 425, 473
title_size
 Class `svg_boxplot`, 425, 473
title_text_length
 Class `svg_1d_plot`, 215, 261
 Class `svg_2d_plot`, 312, 361
 Class `svg_boxplot`, 425, 473, 474
to
 Class `path_element`, 174, 175, 176, 177
 Struct `a_path`, 153
 Struct `c_path`, 154
 Struct `q_path`, 185
 Struct `s_path`, 191
 Struct `t_path`, 195
transform_point
 Class `svg_1d_plot`, 215, 307
 Class `svg_2d_plot`, 312, 418
 Class `svg_boxplot`, 425, 520
transform_x
 Class `svg_1d_plot`, 215, 307
 Class `svg_2d_plot`, 312, 418
 Class `svg_boxplot`, 425, 474
transform_y
 Class `svg_1d_plot`, 215, 307
 Class `svg_2d_plot`, 312, 418
 Class `svg_boxplot`, 425, 474
triangle
 Class `g_element`, 163, 167
ts
 Function operator`<<`, 559
tspan
 Class `text_element`, 196, 199
tspan_element
 Class `tspan_element`, 202
type
 Class `svg_1d_plot`, 251
 Class `svg_boxplot`, 465

Function template `isfinite`, 136
types
 Class `Meas`, 95, 99
 Class template `unc`, 118, 121, 123, 126, 562, 563
`t_or_b`
 Header <`boost/svg_plot/show_1d_settings.hpp`>, 212

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
uncertainties
 Class template `unc`, 562
 Function template `uncs_of`, 129, 566, 567
 Header <`boost/svg_plot/uncertain.hpp`>, 560
`uncertainty`
 Class template `unc`, 118, 119, 121, 122, 123, 124, 126, 127, 561, 562, 563, 564
 Function template `unc_of`, 129, 565, 566
 Header <`boost/quan/unc.hpp`>, 107
Uncorrelated
 Demonstration of using 1D data that includes information about its Uncertainty, 54
 Demonstration of using 2D data that includes information about its uncertainty, 76
`uncun`
 Demonstration of using 2D data that includes information about its uncertainty, 76
`unc_of`
 Function template `unc_of`, 129, 565, 566
 Header <`boost/quan/unc.hpp`>, 107
 Header <`boost/svg_plot/uncertain.hpp`>, 560
units
 Class `svg_1d_plot`, 273, 274, 306
 Class `svg_2d_plot`, 373, 374, 417
 Class `svg_boxplot`, 486
 Class `ticks_labels_style`, 554
`update_image`
 Class `svg_1d_plot`, 215, 262
 Class `svg_2d_plot`, 312, 362
 Class `svg_boxplot`, 425, 474
`use_down_ticks`
 Class `ticks_labels_style`, 553, 554, 555
`use_up_ticks`
 Class `ticks_labels_style`, 553, 555

V

`v`
 Class `path_element`, 173, 177
`V`
 Class `path_element`, 173, 177
value
 2-D Data Values Examples, 67
 Class `axis_line_style`, 536
 Class `Meas`, 95, 99
 Class `plot_point_style`, 545
 Class `svg_boxplot`, 450, 462, 464
 Class template `unc`, 118, 122, 123, 127, 562, 563
 Class `ticks_labels_style`, 553
Tutorial: 2D Special Features, 62

values

- 1-D Data Values Examples, 43
- 2-D Data Values Examples, 67
- Class `svg_1d_plot_series`, 307
- Class `svg_2d_plot`, 409, 410
- Class `svg_2d_plot_series`, 419
- Class `svg_boxplot`, 511
- Function `scale_axis`, 140
- Function template `mnmx`, 137
- Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135
- Using Inkscape, 570

values_count

- Class `svg_2d_plot_series`, 419, 424

values_of

- Function template `values_of`, 131, 568, 569
- Header <`boost/quan/unc.hpp`>, 107
- Header <`boost/svg_plot/uncertain.hpp`>, 560

value_of

- Function template `value_of`, 130, 567, 568
- Header <`boost/quan/unc.hpp`>, 107
- Header <`boost/svg_plot/uncertain.hpp`>, 560

variables

- Function `constant_to_rgb`, 532

vertical

- Class `path_element`, 177

W

wanted

- Class `svg_1d_plot`, 252, 253
- Class `svg_2d_plot`, 353
- Class `svg_boxplot`, 466

way

- Tutorial: 1D Autoscale with Multiple Containers, 18

weight

- Class `svg_2d_plot`, 403

whisker_length

- Class `svg_boxplot`, 425, 474
- Class `svg_boxplot_series`, 521, 528

width

- Class `axis_line_style`, 536, 537, 538
- Class `bar_style`, 538, 539
- Class `box_style`, 540, 541
- Class `plot_line_style`, 543, 544
- Class `rect_element`, 189, 191
- Class `svg_1d_plot`, 238, 239, 274, 287
- Class `svg_2d_plot`, 339, 374, 386
- Class `svg_boxplot`, 451, 468, 486, 487
- Class `svg_style`, 549

Demonstration of using 1D data that includes information about its Uncertainty, 54

width_on

- Class `svg_style`, 547, 549

window

- Class `svg_1d_plot`, 215, 289, 290
- Class `svg_2d_plot`, 312, 340, 409
- Class `svg_boxplot`, 425, 502

with

- Class template `unc`, 564

Function template operator<<, 128, 565

write

Class circle_element, 156, 157
 Class clip_path_element, 158, 159
 Class ellipse_element, 160, 162
 Class g_element, 163, 167
 Class line_element, 170, 171
 Class path_element, 173, 177
 Class polyline_element, 183, 185
 Class curve_element, 186, 188
 Class rect_element, 189, 191
 Class svg_1d_plot, 215, 262
 Class svg_2d_plot, 312, 362
 Class svg_boxplot, 425, 475
 Class svg_color, 529, 530
 Class svg_element, 193, 195
 Class svg_style, 547, 549
 Class text_element, 196, 199
 Class text_element_text, 200
 Class text_parent, 201
 Class tspan_element, 202, 206
 Struct a_path, 154
 Struct c_path, 155
 Struct h_path, 168
 Struct l_path, 169
 Struct m_path, 172
 Struct path_point, 178
 Struct polygon_element, 180, 182
 Struct poly_path_point, 179
 Struct q_path, 185, 186
 Struct s_path, 192
 Struct t_path, 195
 Struct v_path, 207
 Struct z_path, 208

write_attributes

Class circle_element, 156, 158
 Class clip_path_element, 158, 160
 Class ellipse_element, 160, 162
 Class g_element, 163, 167
 Class line_element, 170, 172
 Class path_element, 173, 177
 Class polyline_element, 183, 185
 Class curve_element, 186, 188
 Class rect_element, 189, 191
 Class svg_element, 193, 194
 Class text_element, 196, 200
 Class tspan_element, 202, 207
 Struct polygon_element, 180, 182

X

X

Tutorial: 2D Special Features, 62

x

Class rect_element, 189, 191
 Class text_element, 196, 199
 Class tspan_element, 202, 206

xy_autoscale

Class `svg_2d_plot`, 312, 397
`xy_values_on`
 Class `svg_2d_plot`, 312, 397
`x_adddlimits_color`
 Class `svg_1d_plot`, 215, 262, 263
 Class `svg_2d_plot`, 312, 362
 Class `svg_boxplot`, 425, 475
`x_adddlimits_on`
 Class `svg_1d_plot`, 215, 263
 Class `svg_2d_plot`, 312, 362, 363
 Class `svg_boxplot`, 425, 475, 476
`x_autoscale`
 1-D Auto scaling Examples, 31
 Class `svg_1d_plot`, 215, 264, 265, 280, 281, 287, 288, 293, 298
 Class `svg_2d_plot`, 312, 364, 365, 379, 380, 387, 391, 396
 Class `svg_boxplot`, 425, 476, 477, 492, 493, 500, 505, 510
 Tutorial: 1D Autoscale with Multiple Containers, 18
`x_auto_max_value`
 Class `svg_1d_plot`, 215, 263
 Class `svg_2d_plot`, 312, 363
 Class `svg_boxplot`, 425, 476
`x_auto_min_value`
 Class `svg_1d_plot`, 215, 263
 Class `svg_2d_plot`, 312, 363
 Class `svg_boxplot`, 425, 476
`x_auto_ticks`
 Class `svg_1d_plot`, 215, 264
 Class `svg_2d_plot`, 312, 363, 364
 Class `svg_boxplot`, 425, 476
`x_auto_tick_interval`
 Class `svg_1d_plot`, 215, 263, 264
 Class `svg_2d_plot`, 312, 363
 Class `svg_boxplot`, 425, 476
`x_axis`
 Class `svg_2d_plot`, 312, 365
`x_axis_color`
 1-D Auto scaling Examples, 31
 Class `svg_1d_plot`, 215, 265
 Class `svg_2d_plot`, 312, 365
 Class `svg_boxplot`, 425, 478
`x_axis_label_color`
 Class `svg_1d_plot`, 215, 266
 Class `svg_2d_plot`, 312, 365, 366
 Class `svg_boxplot`, 425, 478
`x_axis_on`
 Class `svg_1d_plot`, 215, 266
 Class `svg_2d_plot`, 312, 366
 Class `svg_boxplot`, 425, 479
`x_axis_position`
 Class `svg_1d_plot`, 215, 266
 Class `svg_2d_plot`, 312, 366
 Class `svg_boxplot`, 425, 479
`x_axis_vertical`
 Class `svg_1d_plot`, 215, 267
 Class `svg_2d_plot`, 312, 366, 367
 Class `svg_boxplot`, 425, 479
`x_axis_width`
 Class `svg_1d_plot`, 215, 267

Class `svg_2d_plot`, 312, 367
Class `svg_boxplot`, 425, 480
`x_datetime_color`
 Class `svg_1d_plot`, 215, 267, 268
 Class `svg_2d_plot`, 312, 367, 368
 Class `svg_boxplot`, 425, 480
`x_datetime_on`
 Class `svg_1d_plot`, 215, 268
 Class `svg_2d_plot`, 312, 368
 Class `svg_boxplot`, 425, 480, 481
`x_decor`
 Class `svg_1d_plot`, 215, 268
 Class `svg_2d_plot`, 312, 368
 Class `svg_boxplot`, 425, 481
`x_df_color`
 Class `svg_1d_plot`, 215, 269
 Class `svg_2d_plot`, 312, 369
 Class `svg_boxplot`, 425, 481, 482
`x_df_on`
 Class `svg_1d_plot`, 215, 269, 270
 Class `svg_2d_plot`, 312, 369, 370
 Class `svg_boxplot`, 425, 482
`x_id_color`
 Class `svg_1d_plot`, 215, 270
 Class `svg_2d_plot`, 312, 370
 Class `svg_boxplot`, 425, 482, 483
`x_id_on`
 Class `svg_1d_plot`, 215, 270
 Class `svg_2d_plot`, 312, 370
 Class `svg_boxplot`, 425, 483
`x_label`
 Class `svg_1d_plot`, 215, 271
 Class `svg_2d_plot`, 312, 371
 Class `svg_boxplot`, 425, 483
`x_labels_strip_e0s`
 Class `svg_1d_plot`, 215, 274
 Class `svg_2d_plot`, 312, 374
 Class `svg_boxplot`, 425, 487
`x_label_color`
 Class `svg_1d_plot`, 215, 271
 Class `svg_2d_plot`, 312, 371
 Class `svg_boxplot`, 425, 484
`x_label_font_size`
 Class `svg_1d_plot`, 215, 272
 Class `svg_2d_plot`, 312, 372
 Class `svg_boxplot`, 425, 484, 485
`x_label_on`
 Class `svg_1d_plot`, 215, 271, 272, 273
 Class `svg_2d_plot`, 312, 371, 372, 373
 Class `svg_boxplot`, 425, 485
`x_label_size`
 Class `svg_boxplot`, 425, 485
`x_label_text`
 Class `svg_boxplot`, 425, 485
`x_label_units`
 Class `svg_1d_plot`, 215, 273
 Class `svg_2d_plot`, 312, 373
 Class `svg_boxplot`, 425, 485

x_major_grid_color
Class svg_1d_plot, 215, 275
Class svg_2d_plot, 312, 374, 375
Class svg_boxplot, 425, 487

x_major_grid_on
Class svg_1d_plot, 215, 275
Class svg_2d_plot, 312, 375
Class svg_boxplot, 425, 487, 488

x_major_grid_width
Class svg_1d_plot, 215, 276
Class svg_2d_plot, 312, 375, 376
Class svg_boxplot, 425, 488

x_major_interval
Class svg_1d_plot, 215, 276
Class svg_2d_plot, 312, 376
Class svg_boxplot, 425, 488, 489

x_major_labels
Class svg_boxplot, 425, 490

x_major_labels_side
Class svg_1d_plot, 215, 277, 278
Class svg_2d_plot, 312, 377
Class svg_boxplot, 425, 490

x_major_label_rotation
Class svg_1d_plot, 215, 276, 277
Class svg_2d_plot, 312, 376, 377
Class svg_boxplot, 425, 489

x_major_tick_color
Class svg_1d_plot, 215, 278, 279
Class svg_2d_plot, 312, 378
Class svg_boxplot, 425, 491

x_major_tick_length
Class svg_1d_plot, 215, 279
Class svg_2d_plot, 312, 378
Class svg_boxplot, 425, 491, 492

x_major_tick_width
Class svg_1d_plot, 215, 279, 280
Class svg_2d_plot, 312, 378, 379
Class svg_boxplot, 425, 492

x_max
Class svg_1d_plot, 215, 280
Class svg_2d_plot, 312, 379
Class svg_boxplot, 425, 492

x_min
Class svg_1d_plot, 215, 280
Class svg_2d_plot, 312, 379, 380
Class svg_boxplot, 425, 492, 493

x_minor_grid_color
Class svg_1d_plot, 215, 281
Class svg_2d_plot, 312, 380
Class svg_boxplot, 425, 493, 494

x_minor_grid_on
Class svg_1d_plot, 215, 281, 282
Class svg_2d_plot, 312, 381
Class svg_boxplot, 425, 494

x_minor_grid_width
Class svg_1d_plot, 215, 282
Class svg_2d_plot, 312, 381
Class svg_boxplot, 425, 494

x_minor_tick_color
 Class `svg_1d_plot`, 215, 283
 Class `svg_2d_plot`, 312, 382
 Class `svg_boxplot`, 425, 495
x_num_minor_ticks
 Class `svg_1d_plot`, 215, 284
 Class `svg_2d_plot`, 312, 383
 Class `svg_boxplot`, 425, 496
 Implementation and other notes, 577
x_order_color
 Class `svg_1d_plot`, 215, 284, 285
 Class `svg_2d_plot`, 312, 383, 384
 Class `svg_boxplot`, 425, 497
x_order_on
 Class `svg_1d_plot`, 215, 285
 Class `svg_2d_plot`, 312, 384
 Class `svg_boxplot`, 425, 497
x_plusminus_color
 Class `svg_1d_plot`, 215, 285, 286
 Class `svg_2d_plot`, 312, 384, 385
 Class `svg_boxplot`, 425, 497, 498
x_plusminus_on
 Class `svg_1d_plot`, 215, 286
 Class `svg_2d_plot`, 312, 385
 Class `svg_boxplot`, 425, 498
x_prefix
 Class `svg_1d_plot`, 215, 286
 Class `svg_2d_plot`, 312, 385
 Class `svg_boxplot`, 425, 498
x_range
 1-D Auto scaling Examples, 31
 1-D Autoscaling Various Containers Examples, 40
 1-D Axis Scaling, 26
 Class `svg_1d_plot`, 215, 286, 287
 Class `svg_2d_plot`, 312, 385, 386
 Class `svg_boxplot`, 425, 499
 Tutorial: 1D Autoscale with Multiple Containers, 18
x_size
 1-D Data Values Examples, 43
 Class `svg_1d_plot`, 215, 238, 239, 287
 Class `svg_2d_plot`, 312, 339, 386
 Class `svg_boxplot`, 425, 451, 499, 500
x_steps
 Class `svg_1d_plot`, 215, 287, 288
 Class `svg_2d_plot`, 312, 387, 388
 Class `svg_boxplot`, 425, 500, 501
x_suffix
 Class `svg_1d_plot`, 215, 289
 Class `svg_2d_plot`, 312, 388
 Class `svg_boxplot`, 425, 501
x_ticks
 Class `svg_2d_plot`, 312, 388
x_ticks_down_on
 Class `svg_1d_plot`, 215, 289
 Class `svg_2d_plot`, 312, 388
 Class `svg_boxplot`, 425, 501, 502
x_ticks_on_window_or_axis
 Class `svg_1d_plot`, 215, 289, 290

Class `svg_2d_plot`, 312, 388
Class `svg_boxplot`, 425, 502
`x_ticks_up_on`
 Class `svg_1d_plot`, 215, 290
 Class `svg_2d_plot`, 312, 389
 Class `svg_boxplot`, 425, 503
`x_ticks_values_color`
 Class `svg_1d_plot`, 215, 290, 291
 Class `svg_2d_plot`, 312, 389
 Class `svg_boxplot`, 425, 503
`x_ticks_values_font_family`
 Class `svg_1d_plot`, 215, 291
 Class `svg_2d_plot`, 312, 389, 390
 Class `svg_boxplot`, 425, 503, 504
`x_ticks_values_font_size`
 Class `svg_1d_plot`, 215, 291, 292
 Class `svg_2d_plot`, 312, 390
 Class `svg_boxplot`, 425, 504
`x_ticks_values_ioflags`
 Class `svg_1d_plot`, 215, 292
 Class `svg_2d_plot`, 312, 390, 391
 Class `svg_boxplot`, 425, 504, 505
`x_ticks_values_precision`
 Class `svg_1d_plot`, 215, 292, 293
 Class `svg_2d_plot`, 312, 391
 Class `svg_boxplot`, 425, 505
`x_tick_color`
 Class `svg_boxplot`, 425, 501
`x_tick_length`
 Class `svg_boxplot`, 425, 501
`x_tick_width`
 Class `svg_boxplot`, 425, 501
`x_tight`
 Class `svg_1d_plot`, 215, 293
 Class `svg_2d_plot`, 312, 391, 392
 Class `svg_boxplot`, 425, 505, 506
`x_values_color`
 Class `svg_1d_plot`, 215, 295
 Class `svg_2d_plot`, 312, 393
 Class `svg_boxplot`, 425, 507
`x_values_font_family`
 Class `svg_1d_plot`, 215, 295
 Class `svg_2d_plot`, 312, 394
 Class `svg_boxplot`, 425, 508
`x_values_font_size`
 Class `svg_1d_plot`, 215, 295, 296
 Class `svg_2d_plot`, 312, 394
 Class `svg_boxplot`, 425, 508
`x_values_ioflags`
 Class `svg_1d_plot`, 215, 296
 Class `svg_2d_plot`, 312, 394, 395
 Class `svg_boxplot`, 425, 508, 509
`x_values_on`
 Class `svg_1d_plot`, 215, 296, 297
 Class `svg_2d_plot`, 312, 395
 Class `svg_boxplot`, 425, 509
`x_values_precision`
 Class `svg_1d_plot`, 215, 297

Class `svg_2d_plot`, 312, 395, 396
Class `svg_boxplot`, 425, 509, 510
`x_values_rotation`
 Class `svg_1d_plot`, 215, 297
 Class `svg_2d_plot`, 312, 396
 Class `svg_boxplot`, 425, 510
`x_value_iflags`
 Class `svg_1d_plot`, 215, 293, 294
 Class `svg_2d_plot`, 312, 392, 393
 Class `svg_boxplot`, 425, 506, 507
`x_value_precision`
 Class `svg_1d_plot`, 215, 294
 Class `svg_2d_plot`, 312, 393
 Class `svg_boxplot`, 425, 507
`x_with_zero`
 Class `svg_1d_plot`, 215, 298
 Class `svg_2d_plot`, 312, 396
 Class `svg_boxplot`, 425, 510

Y

`y`
 Class `rect_element`, 189, 191
 Class `text_element`, 196, 199
 Class `tspan_element`, 202, 206, 207
`y_addlimits_color`
 Class `svg_2d_plot`, 312, 397
`y_addlimits_on`
 Class `svg_2d_plot`, 312, 397
`y_autoscale`
 Class `svg_2d_plot`, 312, 397, 398
 Class `svg_boxplot`, 425, 511
`y_axis`
 Class `svg_2d_plot`, 312, 398
`y_axis_color`
 Class `svg_1d_plot`, 215, 298
 Class `svg_2d_plot`, 312, 398
 Class `svg_boxplot`, 425, 511, 512
`y_axis_label_color`
 Class `svg_2d_plot`, 312, 398, 399
`y_axis_on`
 Class `svg_1d_plot`, 215, 298, 299
 Class `svg_2d_plot`, 312, 399
 Class `svg_boxplot`, 425, 512
`y_axis_position`
 Class `svg_2d_plot`, 312, 399
 Class `svg_boxplot`, 425, 512
`y_axis_value_color`
 Class `svg_2d_plot`, 312, 399
`y_axis_width`
 Class `svg_2d_plot`, 312, 399, 400
`y_decor`
 Class `svg_2d_plot`, 312, 400
`y_df_color`
 Class `svg_2d_plot`, 312, 400
`y_df_on`
 Class `svg_2d_plot`, 312, 400
`y_label`

Class `svg_1d_plot`, 215, 299
Class `svg_2d_plot`, 312, 400, 401
Class `svg_boxplot`, 425, 512
`y_labels_strip_e0s`
 Class `svg_1d_plot`, 215, 300
 Class `svg_2d_plot`, 312, 403
 Class `svg_boxplot`, 425, 514
`y_label_axis`
 Class `svg_2d_plot`, 312, 401
`y_label_color`
 Class `svg_1d_plot`, 215, 299, 300
 Class `svg_2d_plot`, 312, 401
 Class `svg_boxplot`, 425, 513
`y_label_font_family`
 Class `svg_2d_plot`, 312, 401, 402
`y_label_font_size`
 Class `svg_2d_plot`, 312, 402
 Class `svg_boxplot`, 425, 513
`y_label_on`
 Class `svg_1d_plot`, 299
 Class `svg_2d_plot`, 312, 400, 401, 402
 Class `svg_boxplot`, 425, 512, 513
`y_label_text`
 Class `svg_boxplot`, 425, 513
`y_label_units`
 Class `svg_1d_plot`, 215, 300
 Class `svg_2d_plot`, 312, 402
 Class `svg_boxplot`, 425, 513
`y_label_units_on`
 Class `svg_2d_plot`, 312, 402, 403
`y_label_weight`
 Class `svg_2d_plot`, 312, 403
`y_label_width`
 Class `svg_2d_plot`, 312, 403
`y_major_grid_color`
 Class `svg_2d_plot`, 312, 403
`y_major_grid_on`
 Class `svg_2d_plot`, 312, 403, 404
`y_major_grid_width`
 Class `svg_2d_plot`, 312, 404
`y_major_interval`
 Class `svg_2d_plot`, 312, 404
 Class `svg_boxplot`, 425, 514
`y_major_labels_side`
 Class `svg_2d_plot`, 312, 404
`y_major_label_rotation`
 Class `svg_2d_plot`, 312, 404
`y_major_tick_color`
 Class `svg_2d_plot`, 312, 405
 Class `svg_boxplot`, 425, 514
`y_major_tick_length`
 Class `svg_2d_plot`, 312, 405
 Class `svg_boxplot`, 425, 514
`y_major_tick_width`
 Class `svg_2d_plot`, 312, 405
 Class `svg_boxplot`, 425, 514
`y_max`
 Class `svg_2d_plot`, 312, 405

y_min
 Class `svg_2d_plot`, 312, 405
y_minor_grid_color
 Class `svg_2d_plot`, 312, 405
y_minor_grid_on
 Class `svg_2d_plot`, 312, 405, 406
y_minor_grid_width
 Class `svg_2d_plot`, 312, 406
y_minor_interval
 Class `svg_1d_plot`, 215, 300
 Class `svg_2d_plot`, 312, 406
 Class `svg_boxplot`, 425, 514
y_minor_tick_color
 Class `svg_2d_plot`, 312, 406
 Class `svg_boxplot`, 425, 514
y_minor_tick_length
 Class `svg_2d_plot`, 312, 406
 Class `svg_boxplot`, 425, 515
y_minor_tick_width
 Class `svg_2d_plot`, 312, 406, 407
 Class `svg_boxplot`, 425, 515
y_num_minor_ticks
 Class `svg_2d_plot`, 312, 407
 Class `svg_boxplot`, 425, 515
y_plusminus_color
 Class `svg_2d_plot`, 312, 407
y_plusminus_on
 Class `svg_2d_plot`, 312, 407
y_prefix
 Class `svg_2d_plot`, 312, 407
y_range
 Class `svg_2d_plot`, 312, 407
 Class `svg_boxplot`, 425, 515
y_separator
 Class `svg_2d_plot`, 312, 407
y_size
 1-D Data Values Examples, 43
 Class `svg_1d_plot`, 215, 239, 300, 301
 Class `svg_2d_plot`, 312, 339, 340, 408
 Class `svg_boxplot`, 425, 451, 452, 515
y_suffix
 Class `svg_2d_plot`, 312, 408
y_ticks
 Class `svg_2d_plot`, 312, 408
y_ticks_left_on
 Class `svg_2d_plot`, 312, 408
y_ticks_on_window_or_axis
 Class `svg_2d_plot`, 312, 408, 409
y_ticks_right_on
 Class `svg_2d_plot`, 312, 409
y_ticks_values_color
 Class `svg_2d_plot`, 312, 409
y_ticks_values_font_family
 Class `svg_2d_plot`, 312, 409
y_ticks_values_font_size
 Class `svg_2d_plot`, 312, 409
y_ticks_values_ioflags
 Class `svg_2d_plot`, 312, 410

y_ticks_values_precision
 Class `svg_2d_plot`, 312, 410
y_values_color
 Class `svg_2d_plot`, 312, 410, 411
y_values_font_family
 Class `svg_2d_plot`, 312, 411
y_values_font_size
 Class `svg_2d_plot`, 312, 411
y_values_ioflags
 Class `svg_2d_plot`, 312, 411
y_values_on
 Class `svg_2d_plot`, 312, 411
y_values_precision
 Class `svg_2d_plot`, 312, 411, 412
y_values_rotation
 Class `svg_2d_plot`, 312, 412
y_value_ioflags
 Class `svg_2d_plot`, 312, 410
y_value_precision
 Class `svg_2d_plot`, 312, 410

Z

z
 Class `path_element`, 173, 177
Z
 Class `path_element`, 173, 177
zero
 Class `svg_1d_plot`, 298
 Class `svg_2d_plot`, 396, 397
 Class `svg_boxplot`, 510, 511
z_path
 Struct `z_path`, 208

Macro Index

Symbols

B

`BOOST_SVG_LEGEND_DIAGNOSTICS`
 Header <`boost/svg_plot/detail/axis_plot_frame.hpp`>, 144
`BOOST_SVG_POINT_DIAGNOSTICS`
 Header <`boost/svg_plot/detail/axis_plot_frame.hpp`>, 144
`BOOST_SVG_STYLE_DIAGNOSTICS`
 Header <`boost/svg_plot/svg_style.hpp`>, 534
`BOOST_SVG_TITLE_DIAGNOSTICS`
 Header <`boost/svg_plot/detail/axis_plot_frame.hpp`>, 144

typedef Index

Symbols

Index

Symbols

1-D Auto scaling Examples

10, 36
autoscale_check_limits, 31
axis, 31
background, 31
color, 31
container, 31
data, 31
example, 31
fill, 31
intersect, 31
legend, 31
maximum, 31, 36
minimum, 31, 36
origin, 31
part, 31
RGB, 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
part, 26
scaling, 26
series, 26
SVG, 26
vector_iterator, 26
x_range, 26

1-D Data Values Examples

color, 43
colors, 43
container, 43
data, 43
example, 43
fill, 43
font, 43
ioflags, 43
label, 43
marker, 43
precision, 43
RGB, 43
series, 43
show_1d_plot_settings, 43
stroke, 43
SVG, 43
title, 43
Unicode, 43
values, 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
few, 14
legend, 14
marker, 14
optional, 14
scaling, 14
series, 14
SVG, 14
title, 14

10

1-D Auto scaling Examples, 36
Function scale_axis, 140
Function template scale_axis, 141, 142, 143

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
- colors, 67
- container, 67
- data, 67, 69
- example, 67
- fill, 67
- font, 67
- freedom, 67
- ioflags, 67
- label, 67
- marker, 67
- maximum, 67
- precision, 67
- RGB, 67
- series, 67
- show_2d_plot_settings, 67
- stroke, 67
- SVG, 67
- title, 67
- uncertainty, 67
- value, 67
- values, 67

2D

- Header < boost/svg_plot/detail/functors.hpp >, 151

50

- Definitions of the Quartiles, 86

- Header < boost/svg_plot/quartile.hpp >, 210

A**abnormal**

- Class svg_1d_plot, 215

absolute

- Struct polygon_element, 180, 181

Acknowledgements

- color, 583

- example, 583

- minimum, 583

add_g_element

- Class g_element, 163, 164

adjust_limits

- Class svg_1d_plot, 215, 301

- Class svg_2d_plot, 312, 412

- Class svg_boxplot, 425, 515, 516

alignment

- Class text_element, 196, 197

always
Function template unc_of, 129, 566

area_fill
Class bar_style, 538, 539
Class plot_line_style, 543
Class svg_2d_plot_series, 419, 420

author
Implementation and other notes, 577

autoscale
1-D Autoscaling Various Containers Examples, 40
Class svg_1d_plot, 215, 228
Class svg_2d_plot, 312, 328
Class svg_boxplot, 425, 438, 439
Demonstration of using 2D data that includes information about its uncertainty, 76
Header < boost/quan/unc.hpp >, 107
Header < boost/quan/uncs.hpp >, 132

autoscale_check_limits
1-D Auto scaling Examples, 31
Class svg_1d_plot, 215, 228, 229
Class svg_2d_plot, 312, 328, 329
Class svg_boxplot, 425, 439
Demonstration of using 2D data that includes information about its uncertainty, 76

autoscale_plusminus
Class svg_1d_plot, 215, 229
Class svg_2d_plot, 312, 329
Class svg_boxplot, 425, 439
Demonstration of using 2D data that includes information about its uncertainty, 76

axes
Class svg_1d_plot, 229, 230
Class svg_2d_plot, 329
Class svg_boxplot, 440

axes_on
Class svg_1d_plot, 215, 229, 230
Class svg_2d_plot, 312, 329, 330
Class svg_boxplot, 425, 440

axis
1-D Auto scaling Examples, 31
Class svg_boxplot, 483, 515
Tutorial: 1D Gridlines & Axes - more examples, 22

axis_color
Class svg_boxplot, 425, 440
Class svg_boxplot_series, 521, 523

axis_line_style
Class axis_line_style, 536

axis_width
Class svg_boxplot, 425, 440
Class svg_boxplot_series, 521, 523

a_path
Struct a_path, 154

B

background
1-D Auto scaling Examples, 31
1-D Vector Example, 14
Class g_element, 164, 167
Class svg_1d_plot, 215, 230, 231, 239, 240, 251, 252, 301, 302, 303
Class svg_2d_plot, 312, 328, 330, 331, 340, 341, 351, 352, 412, 413, 414

Class `svg_boxplot`, 425, 440, 441, 452, 453, 465, 516, 517
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
Type `svg_color_constant`, 530
Using Inkscape, 570
`background_border_color`
 Class `svg_1d_plot`, 215, 230
 Class `svg_2d_plot`, 312, 330
 Class `svg_boxplot`, 425, 440, 441
`background_border_width`
 Class `svg_1d_plot`, 215, 230, 231
 Class `svg_2d_plot`, 312, 330
 Class `svg_boxplot`, 425, 441
`background_color`
 Class `svg_1d_plot`, 215, 231
 Class `svg_2d_plot`, 312, 331
 Class `svg_boxplot`, 425, 441
`bar_area_fill`
 Class `svg_2d_plot_series`, 419, 420, 421
`bar_color`
 Class `svg_2d_plot_series`, 419, 421
`bar_opt`
 Class `bar_style`, 538, 539
 Class `svg_2d_plot_series`, 419, 421
`bar_style`
 Class `bar_style`, 538
`bar_width`
 Class `svg_2d_plot_series`, 419, 421
`begin`
 Class `svg_2d_plot_series`, 420
`Bezier`
 Class `path_element`, 176
`bezier_on`
 Class `plot_line_style`, 543, 544
 Class `svg_1d_plot_series`, 308, 309
 Class `svg_2d_plot_series`, 419, 421
`Boost.SVG plot C++ Reference`
 SVG, 94
`boost_license_on`
 Class `svg_1d_plot`, 215, 231
 Class `svg_2d_plot`, 312, 331
 Class `svg_boxplot`, 425, 441
`BOOST_SVG_LEGEND_DIAGNOSTICS`
 Header <`boost/svg_plot/detail/axis_plot_frame.hpp`>, 144
`BOOST_SVG_POINT_DIAGNOSTICS`
 Header <`boost/svg_plot/detail/axis_plot_frame.hpp`>, 144
`BOOST_SVG_STYLE_DIAGNOSTICS`
 Header <`boost/svg_plot/svg_style.hpp`>, 534
`BOOST_SVG_TITLE_DIAGNOSTICS`
 Header <`boost/svg_plot/detail/axis_plot_frame.hpp`>, 144
`border`
 Class `box_style`, 540
 Class `g_element`, 164

Class `svg_1d_plot`, 215, 230, 231, 237, 238, 240, 252, 303
 Class `svg_2d_plot`, 312, 328, 330, 331, 338, 339, 340, 341, 352, 414
 Class `svg_boxplot`, 425, 440, 441, 442, 446, 450, 451, 452, 453, 462, 465, 466, 479, 512, 517
 Class `svg_boxplot_series`, 523
 Class `svg_style`, 547
 Class `ticks_labels_style`, 553
 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`, 540
bottom
 Class `svg_1d_plot`, 215, 254
 Class `svg_2d_plot`, 354
 Class `svg_boxplot`, 467
box
 Class `box_style`, 540, 541
 Class `svg_1d_plot`, 215, 246, 306
 Class `svg_2d_plot`, 312, 346, 417
 Class `svg_boxplot`, 458, 462, 520
boxplot
 Class `box_style`, 539
 Class `svg_1d_plot`, 228
 Class `svg_2d_plot`, 328
 Class `svg_boxplot`, 424, 438, 439, 446, 448, 449, 464, 465, 469, 473, 475
 Class `svg_boxplot_series`, 521, 522, 524
 Definitions of the Quartiles, 86
 Function `quantile`, 212
 Global `document_ids_`, 152
 Header <`boost/svg_plot/detail/axis_plot_frame.hpp`>, 144
 Header <`boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp`>, 145
 Header <`boost/svg_plot/detail/svg_boxplot_detail.hpp`>, 151
 Header <`boost/svg_plot/svg_boxplot.hpp`>, 424
 Preface, 3
 Real-life Heat flow data, 48
 Simple Example, 84
 Tutorial: Boxplot, 84
box_border
 Class `svg_boxplot`, 425, 442
 Class `svg_boxplot_series`, 521, 523
box_fill
 Class `svg_boxplot`, 425, 442
 Class `svg_boxplot_series`, 521, 523, 524
box_style
 Class `box_style`, 540
 Class `svg_boxplot_series`, 521, 524
box_width
 Class `svg_boxplot`, 425, 442
 Class `svg_boxplot_series`, 521, 524

C

c
 Class `path_element`, 173, 174
C
 Class `path_element`, 173, 174

calculate_plot_window
 Class svg_1d_plot, 215, 232
 Class svg_2d_plot, 312, 331
 Class svg_boxplot, 425, 442
calculate_quantiles
 Class svg_boxplot_series, 521, 524
calculate_transform
 Class svg_1d_plot, 215, 232
case
 Class svg_color, 530
circle
 Class g_element, 163, 164
circle_element
 Class circle_element, 156
Class axis_line_style
 axis_line_style, 536
color, 536, 537
coordinates, 536
data, 536, 538
example, 536, 537
intersect, 537, 538
label, 536, 537
label_on, 536, 537
label_units_on, 536, 537
line, 536, 538
maximum, 536
minimum, 536
position, 536, 537
scaling, 536
stroke, 536
SVG, 536
value, 536
width, 536, 537, 538
Class bar_style
 area_fill, 538, 539
 bar_opt, 538, 539
 bar_style, 538
 color, 538, 539
 data, 538
 fill, 538, 539
 histogram, 538
 line, 538
 stroke, 538
 SVG, 538
 width, 538, 539
Class box_style
 border, 540
 border_on, 540
 box, 540, 541
 boxplot, 539
 box_style, 540
 color, 540, 541
 data, 540
 fill, 540, 541
 fill_on, 540, 541
 line, 540
 margin, 540, 541
 stroke, 540, 541

SVG, 539
width, 540, 541

Class circle_element
circle_element, 156
class_id, 156, 157
clip_id, 156, 157
color, 157, 158
data, 156
example, 156, 157
fill, 158
id, 156, 157
stroke, 158
style, 156, 157
SVG, 155, 156, 157
svg, 156
svg_element, 156
write, 156, 157
write_attributes, 156, 158

Class clip_path_element
class_id, 158, 159
clip_id, 158, 159
clip_path_element, 158
color, 159, 160
data, 158
example, 158, 159
fill, 160
id, 158, 159
stroke, 160
style, 158, 159
SVG, 158, 159
svg, 158
svg_element, 158
write, 158, 159
write_attributes, 158, 160

Class decor_printer
container, 105, 106
example, 106
suffix, 105

Class ellipse_element
class_id, 160, 161
clip_id, 160, 161
color, 162
data, 160, 161
ellipse_element, 160
example, 161, 162
fill, 162
horizontal, 160
id, 160, 161
stroke, 162
style, 160, 161, 162
SVG, 160, 161, 162
svg, 160
svg_element, 160
write, 160, 162
write_attributes, 160, 162

Class g_element
add_g_element, 163, 164
background, 164, 167

border, 164
circle, 163, 164
class_id, 163, 164
clear, 163, 164
clip_id, 163, 164, 165
color, 167
container, 162, 164, 168
data, 163, 168
ellipse, 163, 165
example, 164, 165, 167
fill, 164, 167
g, 163, 165
g_element, 163, 164
hexagon, 163, 165
id, 163, 165
legend, 167
line, 163, 165
path, 163, 166
pentagon, 163, 166
polygon, 163, 166
polyline, 163, 166
push_back, 163, 166
rect, 163, 166
rhombus, 163, 167
size, 163, 167
stroke, 167
style, 163, 167
SVG, 162, 163, 164, 165, 167
svg, 163
svg_element, 163
text, 163, 167
triangle, 163, 167
write, 163, 167
write_attributes, 163, 167

Class histogram_style
 data, 542
 fill, 542
 histogram, 541, 542
 histogram_style, 542
 SVG, 541

Class line_element
 class_id, 170, 171
 clip_id, 170, 171
 color, 171, 172
 data, 170
 example, 170, 171
 fill, 172
 from, 169, 170, 171
 id, 170, 171
 line_element, 170
 stroke, 172
 style, 170, 171
 SVG, 169, 170, 171
 svg, 170
 svg_element, 170
 write, 170, 171
 write_attributes, 170, 172

Class Meas

data, 95
deg_free, 95, 96, 97
deviation, 95, 97
Meas, 95
types, 95, 99
unc, 95
uncertainty, 94, 98, 99
value, 95, 99

Class meas2
 data, 101

Class path_element
 Bezier, 176
 c, 173, 174
 C, 173, 174
 class_id, 173, 174
 clip_id, 173, 174
 color, 176, 177
 container, 173
 data, 173
 example, 174, 175, 177
 fill, 174, 175, 177
 fill_on, 173, 174, 175
 h, 173, 175
 H, 173, 175
 horizontal, 175
 id, 173, 175
 l, 173, 175
 L, 173, 175
 M, 173, 176
 path_element, 173
 q, 173, 176
 Q, 173, 176
 s, 173, 176
 S, 173, 176
 scaling, 173
 stroke, 177
 style, 173, 176
 SVG, 172, 173, 174, 175, 177
 svg, 173
 svg_element, 173
 t, 173, 176
 T, 173, 177
 the, 173
 to, 174, 175, 176, 177
 v, 173, 177
 V, 173, 177
 vertical, 177
 write, 173, 177
 write_attributes, 173, 177
 z, 173, 177
 Z, 173, 177

Class plot_line_style
 area_fill, 543
 bezier_on, 543, 544
 color, 543, 544
 data, 542, 543, 544
 fill, 543
 line, 543, 544

line_on, 543, 544
plot_line_style, 543
series, 542, 543
stroke, 543
SVG, 542
width, 543, 544

Class plot_point_style
color, 544, 545, 546
data, 544, 545, 546, 547
example, 545, 546
fill, 545, 546
fill_color, 545, 546
font, 545, 546
marker, 544, 546
plot_point_style, 545
point, 545, 546
shape, 545, 546
size, 545, 546
stroke, 545, 546
stroke_color, 545, 546
style, 545, 546
SVG, 544
symbol, 545, 546, 547
symbols, 545, 546, 547
uncertainty, 545
Unicode, 545
value, 545

Class polyline_element
class_id, 183, 184
clip_id, 183, 184
color, 185
container, 183
data, 183
example, 184, 185
fill, 185
id, 183, 184
P, 183, 185
polyline_element, 183
stroke, 185
style, 183, 185
SVG, 183, 184, 185
svg, 183
svg_element, 183
the, 183
write, 183, 185
write_attributes, 183, 185

Class curve_element
class_id, 186, 187
clip_id, 186, 187
color, 188
data, 186
example, 187, 188
fill, 188
from, 186, 188
id, 186, 187
point, 186, 188
curve_element, 186
stroke, 188

style, 186, 188
SVG, 186, 187, 188
svg, 186
svg_element, 186
write, 186, 188
write_attributes, 186, 188

Class rect_element
 class_id, 189, 190
 clip_id, 189, 190
 color, 191
 data, 189
 example, 190, 191
 fill, 191
 height, 189, 190
 id, 189, 190
 rect_element, 189
 rect_elements, 190
 stroke, 191
 style, 189, 191
 SVG, 188, 189, 190, 191
 svg, 189
 svg_element, 189
 width, 189, 191
 write, 189, 191
 write_attributes, 189, 191
 x, 189, 191
 y, 189, 191

Class resetMaskedUncFlags
 data, 111

Class resetUncFlags
 data, 111

Class setAllUncFlags
 data, 112

Class setConfidence
 data, 112

Class setMaskedUncFlags
 data, 113

Class setRoundingLoss
 data, 113

Class setScale
 data, 114

Class setSigDigits
 data, 114

Class setUncFlags
 data, 115

Class setUncSigDigits
 data, 115
 uncertainty, 115

Class setUncWidth
 data, 116

Class showUncFlags
 data, 116

Class showUncTypes
 data, 117

Class stars
 data, 134

Class svg_1d_plot
 abnormal, 215

adjust_limits, 215, 301
autoscale, 215, 228
autoscale_check_limits, 215, 228, 229
autoscale_plusminus, 215, 229
axes, 229, 230
axes_on, 215, 229, 230
background, 215, 230, 231, 239, 240, 251, 252, 301, 302, 303
background_border_color, 215, 230
background_border_width, 215, 230, 231
background_color, 215, 231
boost_license_on, 215, 231
border, 215, 230, 231, 237, 238, 240, 252, 303
bottom, 215, 254
box, 215, 246, 306
boxplot, 228
calculate_plot_window, 215, 232
calculate_transform, 215, 232
clear_all, 215, 301
clear_background, 215, 301
clear_grids, 215, 302
clear_legend, 215, 302
clear_plot_background, 215, 302
clear_points, 215, 302
clear_title, 215, 302
clear_x_axis, 215, 303
clear_y_axis, 215, 303
color, 215, 230, 231, 236, 237, 239, 240, 241, 242, 248, 249, 251, 252, 255, 257, 261, 262, 263, 265, 266, 267, 268, 269, 270, 271, 275, 276, 278, 279, 281, 283, 284, 285, 286, 290, 291, 295, 298, 299, 300, 304
confidence, 215, 232
container, 215, 228, 236, 250, 251, 264, 265, 305
coordinate, 240, 246
coord_precision, 215, 232, 233, 262
copyright_date, 215, 233
copyright_holder, 215, 233, 234
data, 214, 215, 228, 230, 232, 234, 235, 236, 242, 243, 250, 251, 253, 254, 256, 262, 263, 265, 268, 269, 270, 280, 281, 285, 286, 287, 288, 292, 293, 295, 296, 297, 298, 301, 302, 303, 304, 305, 307
data_lines_width, 215, 234
derived, 215, 234
description, 215, 234, 235
deviation, 249, 255, 261, 304
digits, 232, 294
document, 233
document_title, 215, 235
double, 250
draw_axes, 215, 235
draw_legend, 215, 303
draw_line, 215, 236
draw_note, 215, 236
draw_plot_curve, 215, 237
draw_plot_line, 215, 237
draw_plot_point, 215, 303, 304
draw_plot_point_value, 215, 304
draw_plot_point_values, 215, 304, 305
draw_title, 215, 305
draw_x_axis, 215, 306
draw_x_axis_label, 215, 306
draw_x_major_tick, 215, 306
draw_x_minor_tick, 215, 306

example, 215, 230, 235, 236, 237, 244, 245, 250, 251, 252, 255, 256, 258, 259, 261, 262, 263, 266, 267, 268, 269, 270, 274, 284, 285, 286, 287, 293, 294, 297, 300, 304, 305, 306
fill, 239, 240, 249, 250, 251, 252, 255, 261, 304
font, 215, 232, 241, 242, 244, 245, 257, 258, 259, 260, 261, 271, 272, 291, 292, 293, 295, 296, 304
greek, 236, 256, 257, 305
grid, 262, 275, 276, 281, 282, 302, 306
header, 303
height, 239, 255, 300, 301
intersect, 266, 267, 307
interval, 278
intervals, 232
ioflags, 294
is, 232
label, 215, 232, 262, 266, 271, 272, 273, 274, 276, 277, 278, 290, 291, 292, 293, 294, 295, 299, 300, 305, 306
labels, 278
legend, 215, 232, 239, 240, 241, 242, 243, 244, 245, 246, 262, 301, 302, 303, 306, 307
legend_background_color, 215, 239
legend_border_color, 215, 240
legend_box_fill_on, 215, 240
legend_color, 215, 240, 241
legend_font_family, 215, 241
legend_font_size, 215, 241
legend_font_weight, 215, 241, 242
legend_header_font_size, 215, 242
legend_lines, 215, 242, 243
legend_on, 215, 243
legend_outside, 215, 243
legend_place, 215, 243, 244
legend_text_font_size, 215, 244
legend_text_font_weight, 215, 244
legend_title, 215, 245
legend_title_font_size, 215, 245
legend_title_font_weight, 215, 245
legend_top_left, 215, 246
legend_width, 215, 246
license, 215, 246, 247
license_attribution, 215, 247
license_commercialuse, 215, 247
license_distribution, 215, 247
license_on, 215, 247, 248
license_reproduction, 215, 248
limit_color, 215, 248, 249
limit_fill_color, 215, 249
line, 277, 278
marker, 215, 256, 263, 268, 269, 270, 285, 286, 292, 293, 295, 296, 297, 298, 303, 304, 307
markers, 307
maximum, 215, 253, 254, 263, 265, 280, 281, 286, 301
middle, 215
minimum, 215, 253, 254, 263, 265, 280, 281, 282, 283, 284, 286, 300, 301, 304, 306
my_plot, 230
NOT, 306
one_sd_color, 215, 249
origin, 215
place_legend_box, 215, 306
plot, 215, 250, 251, 256, 257
plot_background_color, 215, 251
plot_border_color, 215, 252
plot_border_width, 215, 252

plot_window_x_right, 215, 254
plot_window_y, 215, 254
plot_window_y_bottom, 215, 254
plot_window_y_top, 215, 254
point, 304
position, 266, 267
precision, 232, 233, 262, 292, 293, 294, 295, 297, 304
right, 253
rotation, 297
same, 241
scaling, 215, 232, 241, 253, 254, 262, 264, 265, 271, 272, 291
scientific, 262
series, 215, 242, 250, 251, 262, 265, 303, 307
set_ids, 215, 255
shown, 297
size, 215, 238, 239, 242, 255, 272, 287, 291, 292, 293, 301
size_legend_box, 215, 307
string, 236
stroke, 240, 248, 249, 304
suffix, 289
SVG, 214, 215, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 252, 255, 259, 261, 262, 271, 272, 277, 278, 279, 280, 283, 284, 287, 291, 292, 293, 300, 301, 302, 303, 307
svg_1d_plot, 215
text, 241, 244, 303, 306, 307
text_style, 261
ticks, 279, 280, 281, 283, 284
title, 215, 232, 235, 240, 241, 242, 244, 245, 250, 251, 256, 257, 258, 259, 260, 261, 301, 302, 303, 305, 307
title_color, 215, 257
title_font_alignment, 215, 257
title_font_decoration, 215, 257, 258
title_font_family, 215, 258
title_font_size, 215, 259
title_font_stretch, 215, 259
title_font_style, 215, 260
title_on, 215, 260, 261
title_text_length, 215, 261
transform_point, 215, 307
transform_x, 215, 307
transform_y, 215, 307
type, 251
uncertainty, 215, 304
Unicode, 236, 256, 257, 258, 261, 268, 271, 272, 287, 291, 304, 305
units, 273, 274, 306
update_image, 215, 262
wanted, 252, 253
width, 238, 239, 274, 287
window, 215, 289, 290
write, 215, 262
x_addlimits_color, 215, 262, 263
x_addlimits_on, 215, 263
x_autoscale, 215, 264, 265, 280, 281, 287, 288, 293, 298
x_auto_max_value, 215, 263
x_auto_min_value, 215, 263
x_auto_ticks, 215, 264
x_auto_tick_interval, 215, 263, 264
x_axis_color, 215, 265
x_axis_label_color, 215, 266
x_axis_on, 215, 266

x_axis_position, 215, 266
x_axis_vertical, 215, 267
x_axis_width, 215, 267
x_datetime_color, 215, 267, 268
x_datetime_on, 215, 268
x_decor, 215, 268
x_df_color, 215, 269
x_df_on, 215, 269, 270
x_id_color, 215, 270
x_id_on, 215, 270
x_label, 215, 271
x_labels_strip_e0s, 215, 274
x_label_color, 215, 271
x_label_font_size, 215, 272
x_label_on, 215, 271, 272, 273
x_label_units, 215, 273
x_major_grid_color, 215, 275
x_major_grid_on, 215, 275
x_major_grid_width, 215, 276
x_major_interval, 215, 276
x_major_labels_side, 215, 277, 278
x_major_label_rotation, 215, 276, 277
x_major_tick_color, 215, 278, 279
x_major_tick_length, 215, 279
x_major_tick_width, 215, 279, 280
x_max, 215, 280
x_min, 215, 280
x_minor_grid_color, 215, 281
x_minor_grid_on, 215, 281, 282
x_minor_grid_width, 215, 282
x_minor_tick_color, 215, 283
x_num_minor_ticks, 215, 284
x_order_color, 215, 284, 285
x_order_on, 215, 285
x_plusminus_color, 215, 285, 286
x_plusminus_on, 215, 286
x_prefix, 215, 286
x_range, 215, 286, 287
x_size, 215, 238, 239, 287
x_steps, 215, 287, 288
x_suffix, 215, 289
x_ticks_down_on, 215, 289
x_ticks_on_window_or_axis, 215, 289, 290
x_ticks_up_on, 215, 290
x_ticks_values_color, 215, 290, 291
x_ticks_values_font_family, 215, 291
x_ticks_values_font_size, 215, 291, 292
x_ticks_values_ioflags, 215, 292
x_ticks_values_precision, 215, 292, 293
x_tight, 215, 293
x_values_color, 215, 295
x_values_font_family, 215, 295
x_values_font_size, 215, 295, 296
x_values_ioflags, 215, 296
x_values_on, 215, 296, 297
x_values_precision, 215, 297
x_values_rotation, 215, 297
x_value_ioflags, 215, 293, 294

x_value_precision, 215, 294
x_with_zero, 215, 298
y_axis_color, 215, 298
y_axis_on, 215, 298, 299
y_label, 215, 299
y_labels_strip_e0s, 215, 300
y_label_color, 215, 299, 300
y_label_on, 299
y_label_units, 215, 300
y_minor_interval, 215, 300
y_size, 215, 239, 300, 301
zero, 298

Class `svg_1d_plot_series`
 bezier_on, 308, 309
 color, 308, 309, 310
 container, 308
 data, 307, 308, 310
 example, 310
 fill, 309
 fill_color, 308, 309
 legend, 308
 line_color, 308, 309
 line_on, 308, 309
 line_width, 308, 309
 marker, 308, 309, 310
 not, 308
 series, 307, 308, 310
 series_count, 308, 310
 series_limits_count, 308, 310
 shape, 308, 310
 size, 308, 310
 stroke, 309, 310
 stroke_color, 308, 310
 SVG, 307
 `svg_1d_plot_series`, 308
 symbols, 308, 310
 title, 308
 uncertainty, 308
 values, 307

Class `svg_2d_plot`
 adjust_limits, 312, 412
 autoscale, 312, 328
 autoscale_check_limits, 312, 328, 329
 autoscale_plusminus, 312, 329
 axes, 329
 axes_on, 312, 329, 330
 background, 312, 328, 330, 331, 340, 341, 351, 352, 412, 413, 414
 background_border_color, 312, 330
 background_border_width, 312, 330
 background_color, 312, 331
 boost_license_on, 312, 331
 border, 312, 328, 330, 331, 338, 339, 340, 341, 352, 414
 bottom, 354
 box, 312, 346, 417
 boxplot, 328
 calculate_plot_window, 312, 331
 clear_all, 312, 412
 clear_background, 312, 412

clear_grids, 312, 413
clear_legend, 312, 413
clear_plot_background, 312, 413
clear_points, 312, 413
clear_title, 312, 413, 414
clear_x_axis, 312, 414
clear_y_axis, 312, 414
color, 312, 328, 330, 331, 336, 337, 338, 340, 341, 343, 349, 350, 351, 352, 355, 356, 357, 361, 362, 365, 366, 367, 368, 369, 370, 371, 374, 375, 376, 378, 380, 382, 383, 384, 385, 389, 393, 394, 397, 398, 399, 400, 401, 403, 405, 406, 407, 409, 410, 411, 415
confidence, 312, 332
container, 312, 327, 336, 350, 351, 364, 365, 397, 398, 416
coordinate, 312, 341, 346, 347
coord_precision, 312, 332
copyright_date, 312, 333
copyright_holder, 312, 333
data, 311, 312, 327, 330, 331, 332, 333, 334, 335, 336, 337, 338, 343, 350, 351, 353, 354, 357, 362, 363, 365, 368, 369, 370, 379, 380, 384, 385, 387, 390, 391, 392, 393, 394, 395, 396, 397, 398, 400, 407, 411, 412, 413, 414, 415, 416, 418
data_lines_width, 312, 333, 334
derived, 312, 334
description, 312, 334, 335
deviation, 350, 355, 356, 361, 362, 415
digits, 332, 393, 410
document, 333
document_title, 312, 335
draw_bars, 312, 335
draw_bezier_lines, 312, 335
draw_histogram, 312, 335
draw_legend, 312, 414
draw_line, 312, 336
draw_note, 312, 336
draw_plot_curve, 312, 337
draw_plot_line, 312, 337
draw_plot_lines, 312, 337
draw_plot_point, 312, 414, 415
draw_plot_points, 312, 337
draw_plot_point_value, 312, 415
draw_plot_point_values, 312, 415, 416
draw_straight_lines, 312, 338
draw_title, 312, 416
draw_x_axis, 312, 417
draw_x_axis_label, 312, 417
draw_x_major_tick, 312, 417
draw_x_minor_tick, 312, 417
draw_y_axis, 312, 338
draw_y_axis_label, 312, 338
draw_y_major_tick, 312, 338
draw_y_minor_tick, 312, 338
example, 312, 328, 330, 334, 336, 337, 345, 346, 350, 351, 352, 356, 357, 358, 359, 361, 362, 365, 366, 367, 368, 369, 370, 374, 383, 384, 385, 386, 392, 395, 397, 400, 401, 403, 410, 411, 415, 416, 417, 418
file, 362
fill, 328, 338, 340, 341, 349, 350, 351, 352, 355, 356, 361, 362, 415
font, 312, 342, 343, 345, 346, 358, 359, 360, 361, 371, 372, 389, 390, 392, 394, 401, 402, 403, 409, 410, 411, 415
functor, 351
greek, 336, 356, 357, 416
grid, 338, 374, 375, 376, 380, 381, 403, 404, 405, 406, 413, 417
header, 414
height, 340, 355, 408

histogram, 335
horizontal, 404
intersect, 312, 366
interval, 312, 377
intervals, 332
ioflags, 393, 410
is_in_window, 312, 340
label, 312, 335, 338, 365, 366, 371, 372, 373, 374, 376, 377, 389, 390, 392, 393, 399, 400, 401, 402, 403, 404, 410, 412, 416, 417
labels, 312, 404
layout, 311
legend, 312, 328, 340, 341, 342, 343, 344, 345, 346, 347, 412, 413, 414, 417, 418
legend_background_color, 312, 340
legend_border_color, 312, 340, 341
legend_box_fill_on, 312, 341
legend_color, 312, 341
legend_font_family, 312, 342
legend_font_size, 312, 342
legend_font_weight, 312, 342, 343
legend_header_font_size, 312, 343
legend_lines, 312, 343
legend_on, 312, 343, 344
legend_outside, 312, 344
legend_place, 312, 344, 345
legend_text_font_size, 312, 345
legend_text_font_weight, 312, 345
legend_title, 312, 345, 346
legend_title_font_size, 312, 346
legend_title_font_weight, 312, 346
legend_top_left, 312, 346, 347
legend_width, 312, 347
license, 312, 347
license_attribution, 312, 348
license_commercialuse, 312, 348
license_distribution, 312, 348
license_on, 312, 348, 349
license_reproduction, 312, 349
limit_color, 312, 349
limit_fill_color, 312, 349, 350
line, 404
lines, 331
marker, 311, 312, 357, 362, 363, 368, 369, 370, 384, 385, 390, 391, 393, 394, 395, 396, 414, 415, 418
markers, 418
maximum, 312, 353, 354, 363, 364, 379, 380, 385, 386, 397, 398, 405, 407, 412
minimum, 312, 338, 353, 354, 363, 364, 379, 380, 381, 382, 383, 385, 386, 397, 398, 405, 406, 407, 412, 415, 417
my_plot, 330
NOT, 417
one_sd_color, 312, 350
place_legend_box, 312, 417
plot, 312, 350, 351, 356, 357
plot_background_color, 312, 351, 352
plot_border_color, 312, 352
plot_border_width, 312, 352
plot_window_x_right, 312, 354
plot_window_y, 312, 354
plot_window_y_bottom, 312, 354
plot_window_y_top, 312, 355
point, 415

points, 338
position, 366
precision, 312, 332, 333, 391, 393, 395, 396, 410, 411, 412, 415
range, 407
right, 353
rotation, 396
same, 335, 342
scale_axis, 312
scaling, 312, 331, 335, 342, 353, 354, 364, 365, 371, 372, 389, 390, 397, 398, 401, 409, 411
series, 312, 335, 337, 338, 343, 350, 351, 365, 397, 398, 414, 418
set_ids, 312, 355
shown, 312, 395, 397, 411
size, 312, 339, 340, 343, 355, 372, 386, 390, 392, 408
size_legend_box, 312, 418
string, 336
stroke, 340, 341, 349, 398, 399, 403, 405, 415
suffix, 388, 408
SVG, 311, 312, 327, 328, 331, 332, 333, 334, 335, 336, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 352, 355, 359, 361, 362, 371, 372, 378, 379, 382, 383, 386, 389, 390, 392, 401, 408, 409, 411, 412, 413, 414, 418
svg_2d_plot, 312
text, 312, 342, 345, 414, 417, 418
text_style, 312, 361
ticks, 312, 378, 379, 380, 382, 383
title, 312, 328, 335, 341, 342, 343, 345, 346, 350, 351, 356, 357, 358, 359, 360, 361, 412, 413, 414, 416, 418
title_color, 312, 357
title_font_alignment, 312, 357
title_font_decoration, 312, 358
title_font_family, 312, 358
title_font_size, 312, 359
title_font_stretch, 312, 359, 360
title_font_style, 312, 360
title_on, 312, 360, 361
title_text_length, 312, 361
transform_point, 312, 418
transform_x, 312, 418
transform_y, 312, 418
uncertainty, 312, 407, 415
Unicode, 312, 336, 356, 357, 358, 361, 368, 371, 372, 386, 389, 390, 400, 401, 407, 409, 411, 415, 416
units, 373, 374, 417
update_image, 312, 362
values, 409, 410
wanted, 353
weight, 403
width, 339, 374, 386
window, 312, 340, 409
write, 312, 362
xy_autoscale, 312, 397
xy_values_on, 312, 397
x_addlimits_color, 312, 362
x_addlimits_on, 312, 362, 363
x_autoscale, 312, 364, 365, 379, 380, 387, 391, 396
x_auto_max_value, 312, 363
x_auto_min_value, 312, 363
x_auto_ticks, 312, 363, 364
x_auto_tick_interval, 312, 363
x_axis, 312, 365
x_axis_color, 312, 365
x_axis_label_color, 312, 365, 366

x_axis_on, 312, 366
x_axis_position, 312, 366
x_axis_vertical, 312, 366, 367
x_axis_width, 312, 367
x_datetime_color, 312, 367, 368
x_datetime_on, 312, 368
x_decor, 312, 368
x_df_color, 312, 369
x_df_on, 312, 369, 370
x_id_color, 312, 370
x_id_on, 312, 370
x_label, 312, 371
x_labels_strip_e0s, 312, 374
x_label_color, 312, 371
x_label_font_size, 312, 372
x_label_on, 312, 371, 372, 373
x_label_units, 312, 373
x_major_grid_color, 312, 374, 375
x_major_grid_on, 312, 375
x_major_grid_width, 312, 375, 376
x_major_interval, 312, 376
x_major_labels_side, 312, 377
x_major_label_rotation, 312, 376, 377
x_major_tick_color, 312, 378
x_major_tick_length, 312, 378
x_major_tick_width, 312, 378, 379
x_max, 312, 379
x_min, 312, 379, 380
x_minor_grid_color, 312, 380
x_minor_grid_on, 312, 381
x_minor_grid_width, 312, 381
x_minor_tick_color, 312, 382
x_num_minor_ticks, 312, 383
x_order_color, 312, 383, 384
x_order_on, 312, 384
x_plusminus_color, 312, 384, 385
x_plusminus_on, 312, 385
x_prefix, 312, 385
x_range, 312, 385, 386
x_size, 312, 339, 386
x_steps, 312, 387, 388
x_suffix, 312, 388
x_ticks, 312, 388
x_ticks_down_on, 312, 388
x_ticks_on_window_or_axis, 312, 388
x_ticks_up_on, 312, 389
x_ticks_values_color, 312, 389
x_ticks_values_font_family, 312, 389, 390
x_ticks_values_font_size, 312, 390
x_ticks_values_ioflags, 312, 390, 391
x_ticks_values_precision, 312, 391
x_tight, 312, 391, 392
x_values_color, 312, 393
x_values_font_family, 312, 394
x_values_font_size, 312, 394
x_values_ioflags, 312, 394, 395
x_values_on, 312, 395
x_values_precision, 312, 395, 396

x_values_rotation, 312, 396
x_value_ioflags, 312, 392, 393
x_value_precision, 312, 393
x_with_zero, 312, 396
y_addlimits_color, 312, 397
y_addlimits_on, 312, 397
y_autoscale, 312, 397, 398
y_axis, 312, 398
y_axis_color, 312, 398
y_axis_label_color, 312, 398, 399
y_axis_on, 312, 399
y_axis_position, 312, 399
y_axis_value_color, 312, 399
y_axis_width, 312, 399, 400
y_decor, 312, 400
y_df_color, 312, 400
y_df_on, 312, 400
y_label, 312, 400, 401
y_labels_strip_e0s, 312, 403
y_label_axis, 312, 401
y_label_color, 312, 401
y_label_font_family, 312, 401, 402
y_label_font_size, 312, 402
y_label_on, 312, 400, 401, 402
y_label_units, 312, 402
y_label_units_on, 312, 402, 403
y_label_weight, 312, 403
y_label_width, 312, 403
y_major_grid_color, 312, 403
y_major_grid_on, 312, 403, 404
y_major_grid_width, 312, 404
y_major_interval, 312, 404
y_major_labels_side, 312, 404
y_major_label_rotation, 312, 404
y_major_tick_color, 312, 405
y_major_tick_length, 312, 405
y_major_tick_width, 312, 405
y_max, 312, 405
y_min, 312, 405
y_minor_grid_color, 312, 405
y_minor_grid_on, 312, 405, 406
y_minor_grid_width, 312, 406
y_minor_interval, 312, 406
y_minor_tick_color, 312, 406
y_minor_tick_length, 312, 406
y_minor_tick_width, 312, 406, 407
y_num_minor_ticks, 312, 407
y_plusminus_color, 312, 407
y_plusminus_on, 312, 407
y_prefix, 312, 407
y_range, 312, 407
y_separator, 312, 407
y_size, 312, 339, 340, 408
y_suffix, 312, 408
y_ticks, 312, 408
y_ticks_left_on, 312, 408
y_ticks_on_window_or_axis, 312, 408, 409
y_ticks_right_on, 312, 409

y_ticks_values_color, 312, 409
y_ticks_values_font_family, 312, 409
y_ticks_values_font_size, 312, 409
y_ticks_values_ioflags, 312, 410
y_ticks_values_precision, 312, 410
y_values_color, 312, 410, 411
y_values_font_family, 312, 411
y_values_font_size, 312, 411
y_values_ioflags, 312, 411
y_values_on, 312, 411
y_values_precision, 312, 411, 412
y_values_rotation, 312, 412
y_value_ioflags, 312, 410
y_value_precision, 312, 410
zero, 396, 397

Class `svg_2d_plot_series`
area_fill, 419, 420
bar_area_fill, 419, 420, 421
bar_color, 419, 421
bar_opt, 419, 421
bar_width, 419, 421
begin, 420
bezier_on, 419, 421
color, 419, 420, 421, 422, 424
container, 420
data, 419, 420, 421, 422, 423, 424
draw_straight_lines, 419, 420
example, 420, 422, 423
fill, 420, 421
fill_color, 419, 421
font, 422, 423
histogram, 419, 421
legend, 419, 420, 422
limits_count, 419, 422
line, 420, 421, 422
line_color, 419, 422
line_on, 419, 422
line_style, 419, 422
line_width, 419, 422
marker, 419, 420, 421, 422, 423, 424
point_font_decoration, 419, 422
point_font_family, 419, 422, 423
point_font_stretch, 419, 423
point_font_style, 419, 423
point_font_weight, 419, 423
point_style, 419, 423
series, 419, 420, 421, 422, 423, 424
shape, 419, 423
size, 419, 423
stroke, 421, 422, 424
stroke_color, 419, 424
SVG, 419
`svg_2d_plot_series`, 419
title, 419, 420
uncertainty, 420
values, 419
values_count, 419, 424

Class `svg_boxplot`

adjust_limits, 425, 515, 516
autoscale, 425, 438, 439
autoscale_check_limits, 425, 439
autoscale_plusminus, 425, 439
axes, 440
axes_on, 425, 440
axis, 483, 515
axis_color, 425, 440
axis_width, 425, 440
background, 425, 440, 441, 452, 453, 465, 516, 517
background_border_color, 425, 440, 441
background_border_width, 425, 441
background_color, 425, 441
boost_license_on, 425, 441
border, 425, 440, 441, 442, 446, 450, 451, 452, 453, 462, 465, 466, 479, 512, 517
bottom, 467
box, 458, 462, 520
boxplot, 424, 438, 439, 446, 448, 449, 464, 465, 469, 473, 475
box_border, 425, 442
box_fill, 425, 442
box_width, 425, 442
calculate_plot_window, 425, 442
clear_all, 425, 442
clear_background, 425, 516
clear_grids, 425, 516
clear_legend, 425, 516
clear_plot_background, 425, 516
clear_points, 425, 517
clear_title, 425, 517
clear_x_axis, 425, 517
clear_y_axis, 425, 517
color, 425, 438, 440, 441, 442, 446, 447, 448, 449, 452, 453, 455, 461, 462, 463, 465, 468, 469, 470, 474, 475, 478, 480, 481, 482, 483, 484, 487, 488, 491, 493, 494, 495, 497, 498, 501, 503, 507, 508, 511, 512, 513, 514, 518
confidence, 425, 442, 443
container, 438, 447, 448, 464, 465, 477, 511
coordinate, 453, 458, 459
coord_precision, 425, 443
copyright_date, 425, 443, 444
copyright_holder, 425, 444
data, 424, 425, 443, 444, 445, 446, 447, 455, 464, 465, 466, 467, 475, 476, 477, 480, 481, 482, 483, 492, 493, 497, 498, 500, 504, 505, 506, 507, 508, 509, 510, 516, 517, 518, 519, 520, 521
data_lines_width, 425, 444, 445
derived, 425, 445
description, 425, 445
deviation, 462, 463, 468, 469, 474, 518
digits, 443, 507
document, 444
document_title, 425, 446
draw_box, 425, 446
draw_boxplot, 425, 446
draw_legend, 425, 517
draw_line, 425, 446
draw_median, 425, 447
draw_note, 425, 447
draw_plot_curve, 425, 447, 448
draw_plot_line, 425, 448
draw_plot_point, 425, 518
draw_plot_point_value, 425, 518

draw_plot_point_values, 425, 519
draw_title, 425, 448
draw_whiskers, 425, 448
draw_x_axis, 425, 448
draw_x_axis_label, 425, 449
draw_x_major_tick, 425, 449, 520
draw_x_minor_tick, 425, 520
draw_y_axis, 425, 449
draw_y_axis_label, 425, 449
draw_y_major_tick, 425, 449
draw_y_minor_tick, 425, 449
example, 425, 445, 447, 448, 449, 457, 458, 464, 469, 470, 471, 472, 473, 475, 478, 480, 481, 482, 483, 485, 487, 497, 498, 499, 506, 509, 513, 514, 515, 518, 519
extreme_outlier_color, 425, 449
extreme_outlier_fill, 425, 449
extreme_outlier_shape, 425, 449, 450
extreme_outlier_size, 425, 450
extreme_outlier_values_on, 425, 450
fill, 442, 446, 449, 452, 453, 461, 462, 463, 468, 469, 474, 518
font, 425, 453, 454, 455, 457, 458, 470, 471, 472, 473, 484, 485, 503, 504, 506, 508, 513, 518
greek, 447, 469
grid, 449, 487, 488, 493, 494, 495, 516, 520
header, 517
height, 452, 468
intersect, 425, 479, 512
interval, 491
intervals, 425, 442, 443
ioflags, 507
label, 425, 449, 478, 483, 484, 485, 486, 487, 489, 490, 503, 504, 506, 507, 512, 513, 514, 519
labels, 490
left, 514
legend, 425, 442, 452, 453, 454, 455, 456, 457, 458, 459, 516, 517, 520
legend_background_color, 425, 452
legend_border_color, 425, 452, 453
legend_box_fill_on, 425, 453
legend_color, 425, 453
legend_font_family, 425, 453, 454
legend_font_size, 425, 454
legend_font_weight, 425, 454
legend_header_font_size, 425, 455
legend_lines, 425, 455
legend_on, 425, 455, 456
legend_outside, 425, 456
legend_place, 425, 456, 457
legend_text_font_size, 425, 457
legend_text_font_weight, 425, 457
legend_title, 425, 457, 458
legend_title_font_size, 425, 458
legend_title_font_weight, 425, 458
legend_top_left, 425, 458, 459
legend_width, 425, 459
length, 474
license, 425, 459
license_attribution, 425, 460
license_commercialuse, 425, 460
license_distribution, 425, 460
license_on, 425, 460, 461
license_reproduction, 425, 461

limit_color, 425, 461
limit_fill_color, 425, 461, 462
line, 479, 490, 512
marker, 425, 449, 450, 463, 464, 475, 476, 480, 481, 482, 483, 497, 498, 504, 505, 507, 508, 509, 510, 517, 518, 520
markers, 520
maximum, 425, 448, 464, 466, 467, 476, 477, 492, 493, 499, 511, 515, 516
median_color, 425, 462
median_width, 425, 462
minimum, 425, 448, 449, 464, 466, 467, 476, 477, 492, 493, 494, 495, 496, 497, 499, 511, 514, 515, 516, 518, 520
NOT, 520
one_sd_color, 425, 462, 463
origin, 425
outlier, 425, 449, 450, 463, 464
outlier_color, 425, 463
outlier_fill, 425, 463
outlier_shape, 425, 463
outlier_size, 425, 463
outlier_style, 425, 464
outlier_values_on, 425, 464
place_legend_box, 425, 520
plot, 425, 464, 465
plot_background_color, 425, 465
plot_border_color, 425, 465
plot_border_width, 425, 465
plot_window_x_right, 425, 467
plot_window_y, 425, 467
plot_window_y_bottom, 425, 467
plot_window_y_top, 425, 468
point, 518
precision, 443, 505, 507, 509, 510, 518
quartile, 425, 464, 468
quartile_definition, 425, 468
right, 466
rotation, 510
same, 454
scaling, 425, 442, 454, 466, 467, 477, 484, 503, 504, 511
series, 424, 425, 447, 455, 464, 465, 477, 517, 520
shown, 509
size, 425, 451, 452, 455, 468, 484, 485, 499, 504, 506, 515
size_legend_box, 425, 520
string, 447
stroke, 452, 453, 461, 518
style, 425
suffix, 501
SVG, 424, 425, 440, 441, 442, 443, 444, 445, 446, 447, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 465, 466, 468, 471, 472, 473, 474, 475, 484, 485, 490, 491, 492, 495, 496, 499, 500, 503, 504, 506, 515, 516, 517
svg_boxplot, 425
text, 454, 457, 517, 520
text_style, 473
ticks, 491, 492, 493, 495, 496
title, 425, 442, 445, 446, 453, 454, 455, 457, 458, 464, 465, 469, 470, 471, 472, 473, 474, 485, 517, 520
title_color, 425, 469, 470
title_font_alignment, 425, 470
title_font_decoration, 425, 470
title_font_family, 425, 470, 471
title_font_size, 425, 471, 472
title_font_stretch, 425, 472
title_font_style, 425, 472

title_on, 425, 473
title_size, 425, 473
title_text_length, 425, 473, 474
transform_point, 425, 520
transform_x, 425, 474
transform_y, 425, 474
type, 465
uncertainty, 425, 518
Unicode, 447, 448, 469, 470, 471, 473, 481, 484, 499, 503, 504, 518, 519
units, 486
update_image, 425, 474
value, 450, 462, 464
values, 511
wanted, 466
whisker_length, 425, 474
width, 451, 468, 486, 487
window, 425, 502
write, 425, 475
x_addlimits_color, 425, 475
x_addlimits_on, 425, 475, 476
x_autoscale, 425, 476, 477, 492, 493, 500, 505, 510
x_auto_max_value, 425, 476
x_auto_min_value, 425, 476
x_auto_ticks, 425, 476
x_auto_tick_interval, 425, 476
x_axis_color, 425, 478
x_axis_label_color, 425, 478
x_axis_on, 425, 479
x_axis_position, 425, 479
x_axis_vertical, 425, 479
x_axis_width, 425, 480
x_datetime_color, 425, 480
x_datetime_on, 425, 480, 481
x_decor, 425, 481
x_df_color, 425, 481, 482
x_df_on, 425, 482
x_id_color, 425, 482, 483
x_id_on, 425, 483
x_label, 425, 483
x_labels_strip_e0s, 425, 487
x_label_color, 425, 484
x_label_font_size, 425, 484, 485
x_label_on, 425, 485
x_label_size, 425, 485
x_label_text, 425, 485
x_label_units, 425, 485
x_major_grid_color, 425, 487
x_major_grid_on, 425, 487, 488
x_major_grid_width, 425, 488
x_major_interval, 425, 488, 489
x_major_labels, 425, 490
x_major_labels_side, 425, 490
x_major_label_rotation, 425, 489
x_major_tick_color, 425, 491
x_major_tick_length, 425, 491, 492
x_major_tick_width, 425, 492
x_max, 425, 492
x_min, 425, 492, 493

x_minor_grid_color, 425, 493, 494
x_minor_grid_on, 425, 494
x_minor_grid_width, 425, 494
x_minor_tick_color, 425, 495
x_num_minor_ticks, 425, 496
x_order_color, 425, 497
x_order_on, 425, 497
x_plusminus_color, 425, 497, 498
x_plusminus_on, 425, 498
x_prefix, 425, 498
x_range, 425, 499
x_size, 425, 451, 499, 500
x_steps, 425, 500, 501
x_suffix, 425, 501
x_ticks_down_on, 425, 501, 502
x_ticks_on_window_or_axis, 425, 502
x_ticks_up_on, 425, 503
x_ticks_values_color, 425, 503
x_ticks_values_font_family, 425, 503, 504
x_ticks_values_font_size, 425, 504
x_ticks_values_ioflags, 425, 504, 505
x_ticks_values_precision, 425, 505
x_tick_color, 425, 501
x_tick_length, 425, 501
x_tick_width, 425, 501
x_tight, 425, 505, 506
x_values_color, 425, 507
x_values_font_family, 425, 508
x_values_font_size, 425, 508
x_values_ioflags, 425, 508, 509
x_values_on, 425, 509
x_values_precision, 425, 509, 510
x_values_rotation, 425, 510
x_value_ioflags, 425, 506, 507
x_value_precision, 425, 507
x_with_zero, 425, 510
y_autoscale, 425, 511
y_axis_color, 425, 511, 512
y_axis_on, 425, 512
y_axis_position, 425, 512
y_label, 425, 512
y_labels_strip_e0s, 425, 514
y_label_color, 425, 513
y_label_font_size, 425, 513
y_label_on, 425, 512, 513
y_label_text, 425, 513
y_label_units, 425, 513
y_major_interval, 425, 514
y_major_tick_color, 425, 514
y_major_tick_length, 425, 514
y_major_tick_width, 425, 514
y_minor_interval, 425, 514
y_minor_tick_color, 425, 514
y_minor_tick_length, 425, 515
y_minor_tick_width, 425, 515
y_num_minor_ticks, 425, 515
y_range, 425, 515
y_size, 425, 451, 452, 515

zero, 510, 511
Class `svg_boxplot_series`
 `axis_color`, 521, 523
 `axis_width`, 521, 523
 `border`, 523
 `boxplot`, 521, 522, 524
 `box_border`, 521, 523
 `box_fill`, 521, 523, 524
 `box_style`, 521, 524
 `box_width`, 521, 524
 `calculate_quantiles`, 521, 524
 `color`, 521, 523, 524, 525, 526, 527
 `container`, 522, 523
 `data`, 521, 522, 523, 524, 528
 `extreme_outlier_color`, 521, 524
 `extreme_outlier_fill`, 521, 524, 525
 `extreme_outlier_shape`, 521, 525
 `extreme_outlier_size`, 521, 525
 `fill`, 523, 524, 525, 527
 `font`, 521
 `label`, 521
 `length`, 528
 `marker`, 521, 525, 527
 `maximum`, 521, 523, 525, 528
 `max_whisker_color`, 521, 525
 `max_whisker_width`, 521, 525
 `median_color`, 521, 525
 `median_style`, 521, 526
 `median_width`, 521, 526
 `minimum`, 521, 523, 525, 526, 528
 `min_whisker_color`, 521, 526
 `min_whisker_width`, 521, 526
 `outlier`, 524, 525, 526, 527
 `outlier_color`, 521, 526
 `outlier_fill`, 521, 527
 `outlier_shape`, 521, 527
 `outlier_size`, 521, 527
 `outlier_style`, 521, 527
 `quartile`, 521, 523, 524, 527
 `quartile_definition`, 521, 527
 `scaling`, 521, 524
 `series`, 521, 522, 524, 528
 `SVG`, 521
 `svg_boxplot_series`, 521
 `title`, 521, 523, 528
 `uncertainty`, 523
 `whisker_length`, 521, 528
Class `svg_color`
 `case`, 530
 `color`, 528, 529, 530
 `colors`, 530
 `container`, 529
 `data`, 529
 `example`, 529
 `fill`, 529
 `is_blank`, 529, 530
 `my_blank`, 529
 `red`, 529, 530

SVG, 528, 529, 530
svg_color, 529
write, 529, 530

Class svg_element
 class_id, 193, 194
 clip_id, 193, 194
 color, 194, 195
 container, 193
 example, 193, 194
 fill, 194
 id, 193, 194
 stroke, 193, 194
 style, 193, 195
 SVG, 192, 193, 194, 195
 svg_element, 193
 write, 193, 195
 write_attributes, 193, 194

Class svg_style
 border, 547
 color, 547, 548, 549
 data, 547
 example, 547, 549
 fill, 547, 548, 549
 fill_color, 547, 548
 fill_on, 547, 548
 font, 547
 stroke, 547, 548, 549
 stroke_color, 547, 548
 stroke_on, 547, 548
 stroke_width, 547, 549
 SVG, 547, 548, 549
 svg_style, 547
 svg_styles, 548
 title, 547
 width, 549
 width_on, 547, 549
 write, 547, 549

Class template abstract_printer
 container, 104
 example, 104

Class template close_to
 close_to, 148
 equal, 149
 operator, 148, 149
 size, 148, 149
 strength, 148, 149

Class template smallest
 minimum, 149
 operator, 149, 150
 size, 149, 150
 smallest, 149

Class template unc
 data, 118, 122, 123, 127
 deg_free, 118, 119, 120, 123, 125, 562, 563
 deviation, 118, 120, 123, 125, 561
 example, 118, 119, 123, 124, 564
 freedom, 564
 is, 118, 123

maximum, 122, 127, 562
minimum, 564
of, 564
precision, 118, 123
scaling, 561
SVG, 561
types, 118, 121, 123, 126, 562, 563
unc, 118, 123, 562
uncertainties, 562
uncertainty, 118, 119, 121, 122, 123, 124, 126, 127, 561, 562, 563, 564
Unicode, 564
value, 118, 122, 123, 127, 562, 563
with, 564

Class `text_element`
 alignment, 196, 197
 class_id, 196, 197
 clip_id, 196, 198
 color, 198, 200
 container, 196
 element, 199
 example, 196, 197, 198
 fill, 200
 font, 196, 199
 generate_text, 196, 197
 greek, 196
 id, 196, 198
 rotation, 196, 198
 stroke, 200
 style, 196, 198
 SVG, 196, 197, 198, 199
 svg, 196
 svg_element, 196
 text, 196, 198, 199
 textstyle, 196, 199
 text_element, 196
 tspan, 196, 199
 Unicode, 196
 write, 196, 199
 write_attributes, 196, 200
 x, 196, 199
 y, 196, 199

Class `text_element_text`
 SVG, 200
 svg, 200
 text_element_text, 200
 text_parent, 200
 write, 200

Class `text_parent`
 SVG, 200, 201
 text_parent, 201
 write, 201

Class `text_style`
 data, 550
 equality, 552
 example, 550, 551, 552
 font, 549, 550, 551, 552
 font_decoration, 550, 551
 font_family, 550, 551

font_size, 550, 551, 552
font_stretch, 550, 552
font_style, 550, 552
font_weight, 550, 552
inequality, 552
Inkscape, 552
minimum, 551
scaling, 550
size, 550, 551, 552
SVG, 549, 550, 551, 552
text_length, 550, 553
text_style, 550
Unicode, 550, 551
Class ticks_labels_style
 border, 553
 color, 553
 data, 553, 554
 example, 553
 font, 553
 grid, 553
 label, 553, 554
 label_length, 553, 554
 longest_label, 553, 554
 major_value_labels_side, 553, 554
 maximum, 553, 554
 minimum, 553, 554
 precision, 553
 right, 554
 stroke, 553
 style, 553
 SVG, 553, 554
 ticks, 553
 ticks_labels_style, 553
 units, 554
 use_down_ticks, 553, 554, 555
 use_up_ticks, 553, 555
 value, 553
Class tspan_element
 class_id, 202, 203
 clip_id, 202, 203
 color, 202, 203, 204, 205, 207
 dx, 202, 203
 dy, 202, 203
 element, 203, 204, 205
 example, 203, 204, 205
 fill, 203, 204, 207
 fill_color, 202, 203, 204
 fill_on, 202, 204
 font, 202, 204, 205, 206
 font_family, 202, 204
 font_size, 202, 204
 font_style, 202, 204
 font_weight, 202, 204, 205
 id, 202, 205
 Inkscape, 204
 length, 206
 rotation, 202, 205
 stroke, 207

style, 202, 205, 206
SVG, 201, 202, 203, 204, 205, 206
svg, 202
svg_element, 202
text, 202, 205, 206
textstyle, 202, 206
text_length, 202, 206
text_parent, 202
tspan_element, 202
write, 202, 206
write_attributes, 202, 207
x, 202, 206
y, 202, 206, 207

Class value_style
 color, 555, 556, 557
 data, 555, 556, 557
 draw_plot_point_values, 556
 example, 556
 fill, 556
 font, 556
 label, 555, 556
 line, 556
 minimum, 557
 precision, 556
 series, 555
 stroke, 556
 style, 556
 SVG, 555
 uncertainty, 555, 556, 557
 value_style, 556

class_id
 Class circle_element, 156, 157
 Class clip_path_element, 158, 159
 Class ellipse_element, 160, 161
 Class g_element, 163, 164
 Class line_element, 170, 171
 Class path_element, 173, 174
 Class polyline_element, 183, 184
 Class quurve_element, 186, 187
 Class rect_element, 189, 190
 Class svg_element, 193, 194
 Class text_element, 196, 197
 Class tspan_element, 202, 203
 Struct polygon_element, 180, 181

clear
 Class g_element, 163, 164

clear_all
 Class svg_1d_plot, 215, 301
 Class svg_2d_plot, 312, 412
 Class svg_boxplot, 425, 442

clear_background
 Class svg_1d_plot, 215, 301
 Class svg_2d_plot, 312, 412
 Class svg_boxplot, 425, 516

clear_grids
 Class svg_1d_plot, 215, 302
 Class svg_2d_plot, 312, 413
 Class svg_boxplot, 425, 516

clear_legend
 Class `svg_1d_plot`, 215, 302
 Class `svg_2d_plot`, 312, 413
 Class `svg_boxplot`, 425, 516

clear_plot_background
 Class `svg_1d_plot`, 215, 302
 Class `svg_2d_plot`, 312, 413
 Class `svg_boxplot`, 425, 516

clear_points
 Class `svg_1d_plot`, 215, 302
 Class `svg_2d_plot`, 312, 413
 Class `svg_boxplot`, 425, 517

clear_title
 Class `svg_1d_plot`, 215, 302
 Class `svg_2d_plot`, 312, 413, 414
 Class `svg_boxplot`, 425, 517

clear_x_axis
 Class `svg_1d_plot`, 215, 303
 Class `svg_2d_plot`, 312, 414
 Class `svg_boxplot`, 425, 517

clear_y_axis
 Class `svg_1d_plot`, 215, 303
 Class `svg_2d_plot`, 312, 414
 Class `svg_boxplot`, 425, 517

clip_id
 Class `circle_element`, 156, 157
 Class `clip_path_element`, 158, 159
 Class `ellipse_element`, 160, 161
 Class `g_element`, 163, 164, 165
 Class `line_element`, 170, 171
 Class `path_element`, 173, 174
 Class `polyline_element`, 183, 184
 Class `curve_element`, 186, 187
 Class `rect_element`, 189, 190
 Class `svg_element`, 193, 194
 Class `text_element`, 196, 198
 Class `tspan_element`, 202, 203
 Struct `polygon_element`, 180, 181

clip_path_element
 Class `clip_path_element`, 158

close_to
 Class template `close_to`, 148

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, 583
 Class `axis_line_style`, 536, 537
 Class `bar_style`, 538, 539
 Class `box_style`, 540, 541
 Class `circle_element`, 157, 158
 Class `clip_path_element`, 159, 160
 Class `ellipse_element`, 162
 Class `g_element`, 167
 Class `line_element`, 171, 172
 Class `path_element`, 176, 177
 Class `plot_line_style`, 543, 544

Class `plot_point_style`, 544, 545, 546
Class `polyline_element`, 185
Class `curve_element`, 188
Class `rect_element`, 191
Class `svg_1d_plot`, 215, 230, 231, 236, 237, 239, 240, 241, 242, 248, 249, 251, 252, 255, 257, 261, 262, 263, 265, 266, 267, 268, 269, 270, 271, 275, 276, 278, 279, 281, 283, 284, 285, 286, 290, 291, 295, 298, 299, 300, 304
Class `svg_1d_plot_series`, 308, 309, 310
Class `svg_2d_plot`, 312, 328, 330, 331, 336, 337, 338, 340, 341, 343, 349, 350, 351, 352, 355, 356, 357, 361, 362, 365, 366, 367, 368, 369, 370, 371, 374, 375, 376, 378, 380, 382, 383, 384, 385, 389, 393, 394, 397, 398, 399, 400, 401, 403, 405, 406, 407, 409, 410, 411, 415
Class `svg_2d_plot_series`, 419, 420, 421, 422, 424
Class `svg_boxplot`, 425, 438, 440, 441, 442, 446, 447, 448, 449, 452, 453, 455, 461, 462, 463, 465, 468, 469, 470, 474, 475, 478, 480, 481, 482, 483, 484, 487, 488, 491, 493, 494, 495, 497, 498, 501, 503, 507, 508, 511, 512, 513, 514, 518
Class `svg_boxplot_series`, 521, 523, 524, 525, 526, 527
Class `svg_color`, 528, 529, 530
Class `svg_element`, 194, 195
Class `svg_style`, 547, 548, 549
Class `text_element`, 198, 200
Class `ticks_labels_style`, 553
Class `tspan_element`, 202, 203, 204, 205, 207
Class `value_style`, 555, 556, 557
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`, 532
Function `is_blank`, 532, 533
Function `operator!=`, 533
Function `operator<<`, 533
Function `operator==`, 534
Global `color_array`, 531, 532
Header <`boost/svg_plot/svg_color.hpp`>, 528
Header <`boost/svg_plot/svg_style.hpp`>, 534
Implementation and other notes, 577
Showing data values at Numerical Limits, 92
Simple Code Example, 58, 59
Simple Example, 84
Struct `polygon_element`, 180, 182
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`, 530, 531
Using Inkscape, 570
Colors
 `background`, 6
 `border`, 6
 `color`, 6
 `colors`, 6
 `container`, 6
 `example`, 6
 `maximum`, 6
 `minimum`, 6
 `plot`, 6
 `SVG`, 6
 `colors`

1-D Data Values Examples, 43
2-D Data Values Examples, 67
Class `svg_color`, 530
Colors, 6
Demonstration of using 2D data that includes information about its uncertainty, 76
Function operator!=, 533
Function operator==, 534
Compilers and Examples
 container, 576
 example, 576
 SVG, 576
 uncertainty, 576
confidence
 Class `svg_1d_plot`, 215, 232
 Class `svg_2d_plot`, 312, 332
 Class `svg_boxplot`, 425, 442, 443
constant_to_rgb
 Function `constant_to_rgb`, 532
 Header <`boost/svg_plot/svg_color.hpp`>, 528
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 `decor_printer`, 105, 106
 Class `g_element`, 162, 164, 168
 Class `path_element`, 173
 Class `polyline_element`, 183
 Class `svg_1d_plot`, 215, 228, 236, 250, 251, 264, 265, 305
 Class `svg_1d_plot_series`, 308
 Class `svg_2d_plot`, 312, 327, 336, 350, 351, 364, 365, 397, 398, 416
 Class `svg_2d_plot_series`, 420
 Class `svg_boxplot`, 438, 447, 448, 464, 465, 477, 511
 Class `svg_boxplot_series`, 522, 523
 Class `svg_color`, 529
 Class `svg_element`, 193
 Class `template_abstract_printer`, 104
 Class `text_element`, 196
Colors, 6
Compilers and Examples, 576
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 `mnnmx`, 137
Function template `range_all`, 137
Function template `range_mx`, 137
Function template `scale_axis`, 141, 142, 143
Function template `show`, 143
Function template `show_all`, 144
Header <`boost/quan/si_units.hpp`>, 101
Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135
How To Use This Documentation, 5
Implementation and other notes, 577

Preface, 3
Simple Code Example, 58
Struct polygon_element, 180
To Do List, 582
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, 570
coordinate
 Class svg_1d_plot, 240, 246
 Class svg_2d_plot, 312, 341, 346, 347
 Class svg_boxplot, 453, 458, 459
coordinates
 Class axis_line_style, 536
 Demonstration of adding lines and curves, typically a least squares fit, 80
 Struct m_path, 172
 Struct poly_path_point, 178
coord_precision
 Class svg_1d_plot, 215, 232, 233, 262
 Class svg_2d_plot, 312, 332
 Class svg_boxplot, 425, 443
copyright_date
 Class svg_1d_plot, 215, 233
 Class svg_2d_plot, 312, 333
 Class svg_boxplot, 425, 443, 444
copyright_holder
 Class svg_1d_plot, 215, 233, 234
 Class svg_2d_plot, 312, 333
 Class svg_boxplot, 425, 444
curve
 Demonstration of adding lines and curves, typically a least squares fit, 80
c_path
 Struct c_path, 155

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, 536, 538
 Class bar_style, 538
 Class box_style, 540
 Class circle_element, 156
 Class clip_path_element, 158
 Class ellipse_element, 160, 161
 Class g_element, 163, 168
 Class histogram_style, 542
 Class line_element, 170
 Class Meas, 95
 Class meas2, 101

Class path_element, 173
Class plot_line_style, 542, 543, 544
Class plot_point_style, 544, 545, 546, 547
Class polyline_element, 183
Class curve_element, 186
Class rect_element, 189
Class resetMaskedUncFlags, 111
Class resetUncFlags, 111
Class setAllUncFlags, 112
Class setConfidence, 112
Class setMaskedUncFlags, 113
Class setRoundingLoss, 113
Class setScale, 114
Class setSigDigits, 114
Class setUncFlags, 115
Class setUncSigDigits, 115
Class setUncWidth, 116
Class showUncFlags, 116
Class showUncTypes, 117
Class stars, 134
Class svg_1d_plot, 214, 215, 228, 230, 232, 234, 235, 236, 242, 243, 250, 251, 253, 254, 256, 262, 263, 265, 268, 269, 270, 280, 281, 285, 286, 287, 288, 292, 293, 295, 296, 297, 298, 301, 302, 303, 304, 305, 307
Class svg_1d_plot_series, 307, 308, 310
Class svg_2d_plot, 311, 312, 327, 330, 331, 332, 333, 334, 335, 336, 337, 338, 343, 350, 351, 353, 354, 357, 362, 363, 365, 368, 369, 370, 379, 380, 384, 385, 387, 390, 391, 392, 393, 394, 395, 396, 397, 398, 400, 407, 411, 412, 413, 414, 415, 416, 418
Class svg_2d_plot_series, 419, 420, 421, 422, 423, 424
Class svg_boxplot, 424, 425, 443, 444, 445, 446, 447, 455, 464, 465, 466, 467, 475, 476, 477, 480, 481, 482, 483, 492, 493, 497, 498, 500, 504, 505, 506, 507, 508, 509, 510, 516, 517, 518, 519, 520, 521
Class svg_boxplot_series, 521, 522, 523, 524, 528
Class svg_color, 529
Class svg_style, 547
Class template unc, 118, 122, 123, 127
Class text_style, 550
Class ticks_labels_style, 553, 554
Class value_style, 555, 556, 557
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, 211
Function operator<<, 559
Function quantile, 211, 212
Function template range_mx, 137
Function template scale_axis, 141, 142, 143
Global spacing_factor, 147
Header < boost/quan/unc_init.hpp >, 131
Header < boost/svg_plot/detail/auto_axes.hpp >, 135
Header < boost/svg_plot/detail/axis_plot_frame.hpp >, 144
Header < boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp >, 145
Header < boost/svg_plot/detail/functors.hpp >, 151
Header < boost/svg_plot/detail/numeric_limits_handling.hpp >, 151
Header < boost/svg_plot/quantile.hpp >, 210
Header < boost/svg_plot/svg_1d_plot.hpp >, 214
Header < boost/svg_plot/svg_boxplot.hpp >, 424
Header < boost/svg_plot/svg_style.hpp >, 534
Histograms of 2D data, 82
Implementation and other notes, 577, 578

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, 154
Struct c_path, 155
Struct h_path, 168
Struct l_path, 169
Struct m_path, 172
Struct path_point, 178
Struct polygon_element, 180
Struct poly_path_point, 179
Struct q_path, 185
Struct s_path, 192
Struct t_path, 195
Struct unit, 102
Struct v_path, 207
Struct z_path, 208
To Do List, 582
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, 215, 234
 Class svg_2d_plot, 312, 333, 334
 Class svg_boxplot, 425, 444, 445
default_plot_point_style
 Header < boost/svg_plot/svg_style.hpp >, 534
definition
 Definitions of the Quartiles, 86
Definitions of the Quartiles
 50, 86
 boxplot, 86
 container, 86
 data, 86, 87
 definition, 86
 example, 86
 Hoaglin, 86
 Hoaglin4, 86
 Hoaglin5, 86
 Hoaglin6, 86
 Hoaglin7, 86
 Hoaglin8, 86
 label, 86
 maximum, 86
 minimum, 86, 87
 quantile, 86
 quartile, 86, 87
 SAS, 86
 scaling, 86, 87
 series, 86
 SVG, 86
 title, 86
deg_free
 Class Meas, 95, 96, 97

Class template unc, 118, 119, 120, 123, 125, 562, 563

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

color, 80

coordinates, 80

curve, 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

digits, 54

few, 54

fill, 54

label, 54

legend, 54

marker, 54

precision, 54

series, 54

set, 54

SVG, 54

title, 54

titles, 54

uncertainty, 54

Uncorrelated, 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

colors, 76

container, 76

data, 76

example, 76

label, 76

series, 76

set, 76

SVG, 76

u1, 76

u2, 76

uncertainty, 76

Uncorrelated, 76

uncun, 76

Unicode, 76

deque

1-D STL Containers Examples, 15

Tutorial: 1D More Layout Examples, 20

derived

Class svg_1d_plot, 215, 234

Class `svg_2d_plot`, 312, 334
Class `svg_boxplot`, 425, 445
description
Class `svg_1d_plot`, 215, 234, 235
Class `svg_2d_plot`, 312, 334, 335
Class `svg_boxplot`, 425, 445
Implementation and other notes, 577
deviation
Class `Meas`, 95, 97
Class `svg_1d_plot`, 249, 255, 261, 304
Class `svg_2d_plot`, 350, 355, 356, 361, 362, 415
Class `svg_boxplot`, 462, 463, 468, 469, 474, 518
Class `template unc`, 118, 120, 123, 125, 561
digits
Class `svg_1d_plot`, 232, 294
Class `svg_2d_plot`, 332, 393, 410
Class `svg_boxplot`, 443, 507
Demonstration of using 1D data that includes information about its Uncertainty, 54
document
Class `svg_1d_plot`, 233
Class `svg_2d_plot`, 333
Class `svg_boxplot`, 444
Using Inkscape, 570
document_title
Class `svg_1d_plot`, 215, 235
Class `svg_2d_plot`, 312, 335
Class `svg_boxplot`, 425, 446
Implementation and other notes, 577
double
Class `svg_1d_plot`, 250
draw_axes
Class `svg_1d_plot`, 215, 235
draw_bars
Class `svg_2d_plot`, 312, 335
draw_bezier_lines
Class `svg_2d_plot`, 312, 335
draw_box
Class `svg_boxplot`, 425, 446
draw_boxplot
Class `svg_boxplot`, 425, 446
draw_histogram
Class `svg_2d_plot`, 312, 335
draw_legend
Class `svg_1d_plot`, 215, 303
Class `svg_2d_plot`, 312, 414
Class `svg_boxplot`, 425, 517
draw_line
Class `svg_1d_plot`, 215, 236
Class `svg_2d_plot`, 312, 336
Class `svg_boxplot`, 425, 446
draw_median
Class `svg_boxplot`, 425, 447
draw_note
Class `svg_1d_plot`, 215, 236
Class `svg_2d_plot`, 312, 336
Class `svg_boxplot`, 425, 447
draw_plot_curve
Class `svg_1d_plot`, 215, 237

Class `svg_2d_plot`, 312, 337
Class `svg_boxplot`, 425, 447, 448
`draw_plot_line`
 Class `svg_1d_plot`, 215, 237
 Class `svg_2d_plot`, 312, 337
 Class `svg_boxplot`, 425, 448
`draw_plot_lines`
 Class `svg_2d_plot`, 312, 337
`draw_plot_point`
 Class `svg_1d_plot`, 215, 303, 304
 Class `svg_2d_plot`, 312, 414, 415
 Class `svg_boxplot`, 425, 518
`draw_plot_points`
 Class `svg_2d_plot`, 312, 337
`draw_plot_point_value`
 Class `svg_1d_plot`, 215, 304
 Class `svg_2d_plot`, 312, 415
 Class `svg_boxplot`, 425, 518
`draw_plot_point_values`
 Class `svg_1d_plot`, 215, 304, 305
 Class `svg_2d_plot`, 312, 415, 416
 Class `svg_boxplot`, 425, 519
 Class `value_style`, 556
`draw_straight_lines`
 Class `svg_2d_plot`, 312, 338
 Class `svg_2d_plot_series`, 419, 420
`draw_title`
 Class `svg_1d_plot`, 215, 305
 Class `svg_2d_plot`, 312, 416
 Class `svg_boxplot`, 425, 448
`draw_whiskers`
 Class `svg_boxplot`, 425, 448
`draw_x_axis`
 Class `svg_1d_plot`, 215, 306
 Class `svg_2d_plot`, 312, 417
 Class `svg_boxplot`, 425, 448
`draw_x_axis_label`
 Class `svg_1d_plot`, 215, 306
 Class `svg_2d_plot`, 312, 417
 Class `svg_boxplot`, 425, 449
`draw_x_major_tick`
 Class `svg_1d_plot`, 215, 306
 Class `svg_2d_plot`, 312, 417
 Class `svg_boxplot`, 425, 449, 520
`draw_x_minor_tick`
 Class `svg_1d_plot`, 215, 306
 Class `svg_2d_plot`, 312, 417
 Class `svg_boxplot`, 425, 520
`draw_y_axis`
 Class `svg_2d_plot`, 312, 338
 Class `svg_boxplot`, 425, 449
`draw_y_axis_label`
 Class `svg_2d_plot`, 312, 338
 Class `svg_boxplot`, 425, 449
`draw_y_major_tick`
 Class `svg_2d_plot`, 312, 338
 Class `svg_boxplot`, 425, 449
`draw_y_minor_tick`

Class `svg_2d_plot`, 312, 338
Class `svg_boxplot`, 425, 449
`dx`
 Class `tspan_element`, 202, 203
`dy`
 Class `tspan_element`, 202, 203

E

`element`
 Class `text_element`, 199
 Class `tspan_element`, 203, 204, 205

`ellipse`
 Class `g_element`, 163, 165

`ellipse_element`
 Class `ellipse_element`, 160

`epsilon`
 Header <`boost/svg_plot/detail/fp_compare.hpp`>, 147

`equal`
 Class `template close_to`, 149

`equality`
 Class `text_style`, 552

`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, 583
 Class `axis_line_style`, 536, 537
 Class `circle_element`, 156, 157
 Class `clip_path_element`, 158, 159
 Class `decor_printer`, 106
 Class `ellipse_element`, 161, 162
 Class `g_element`, 164, 165, 167
 Class `line_element`, 170, 171
 Class `path_element`, 174, 175, 177
 Class `plot_point_style`, 545, 546
 Class `polyline_element`, 184, 185
 Class `curve_element`, 187, 188
 Class `rect_element`, 190, 191
 Class `svg_1d_plot`, 215, 230, 235, 236, 237, 244, 245, 250, 251, 252, 255, 256, 258, 259, 261, 262, 263, 266, 267, 268, 269, 270, 274, 284, 285, 286, 287, 293, 294, 297, 300, 304, 305, 306
 Class `svg_1d_plot_series`, 310
 Class `svg_2d_plot`, 312, 328, 330, 334, 336, 337, 345, 346, 350, 351, 352, 356, 357, 358, 359, 361, 362, 365, 366, 367, 368, 369, 370, 374, 383, 384, 385, 386, 392, 395, 397, 400, 401, 403, 410, 411, 415, 416, 417, 418
 Class `svg_2d_plot_series`, 420, 422, 423
 Class `svg_boxplot`, 425, 445, 447, 448, 449, 457, 458, 464, 469, 470, 471, 472, 473, 475, 478, 480, 481, 482, 483, 485, 487, 497, 498, 499, 506, 509, 513, 514, 515, 518, 519
 Class `svg_color`, 529
 Class `svg_element`, 193, 194
 Class `svg_style`, 547, 549
 Class `template abstract_printer`, 104
 Class `template unc`, 118, 119, 123, 124, 564

Class `text_element`, 196, 197, 198
Class `text_style`, 550, 551, 552
Class `ticks_labels_style`, 553
Class `tspan_element`, 203, 204, 205
Class `value_style`, 556
Colors, 6
Compilers and Examples, 576
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`, 532
Function `operator<<`, 209, 210, 533, 559
Function `outFmtFlags`, 135, 213
Function `quantile`, 212
Function template `operator<<`, 107, 128, 565
Header < `boost/quan/pair_io.hpp` >, 101
Header < `boost/quan/unc.hpp` >, 107
Header < `boost/svg_plot/detail/functors.hpp` >, 151
Header < `boost/svg_plot/detail/pair.hpp` >, 151
Header < `boost/svg_plot/show_2d_settings.hpp` >, 213
How To Use This Documentation, 5
Implementation and other notes, 577
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`, 172
Struct `polygon_element`, 181, 182
Struct `poly_path_point`, 179
SVG tutorial, 91
To Do List, 582
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`, 530
Using Inkscape, 570
extreme_outlier_color
 Class `svg_boxplot`, 425, 449
 Class `svg_boxplot_series`, 521, 524
extreme_outlier_fill
 Class `svg_boxplot`, 425, 449
 Class `svg_boxplot_series`, 521, 524, 525
extreme_outlier_shape
 Class `svg_boxplot`, 425, 449, 450
 Class `svg_boxplot_series`, 521, 525
extreme_outlier_size
 Class `svg_boxplot`, 425, 450
 Class `svg_boxplot_series`, 521, 525
extreme_outlier_values_on
 Class `svg_boxplot`, 425, 450

F

f

Simple Example, 84
Tutorial: 1D More Layout Examples, 20
Tutorial: Fuller Layout Example, 60
few
 1-D Vector Example, 14
 Demonstration of using 1D data that includes information about its Uncertainty, 54
file
 Class `svg_2d_plot`, 362
 Using Inkscape, 570
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`, 538, 539
 Class `box_style`, 540, 541
 Class `circle_element`, 158
 Class `clip_path_element`, 160
 Class `ellipse_element`, 162
 Class `g_element`, 164, 167
 Class `histogram_style`, 542
 Class `line_element`, 172
 Class `path_element`, 174, 175, 177
 Class `plot_line_style`, 543
 Class `plot_point_style`, 545, 546
 Class `polyline_element`, 185
 Class `curve_element`, 188
 Class `rect_element`, 191
 Class `svg_1d_plot`, 239, 240, 249, 250, 251, 252, 255, 261, 304
 Class `svg_1d_plot_series`, 309
 Class `svg_2d_plot`, 328, 338, 340, 341, 349, 350, 351, 352, 355, 356, 361, 362, 415
 Class `svg_2d_plot_series`, 420, 421
 Class `svg_boxplot`, 442, 446, 449, 452, 453, 461, 462, 463, 468, 469, 474, 518
 Class `svg_boxplot_series`, 523, 524, 525, 527
 Class `svg_color`, 529
 Class `svg_element`, 194
 Class `svg_style`, 547, 548, 549
 Class `text_element`, 200
 Class `tspan_element`, 203, 204, 207
 Class `value_style`, 556
 Demonstration of using 1D data that includes information about its Uncertainty, 54
 Function operator `<<`, 558
 Header < boost/svg_plot/svg_style.hpp >, 534
 Implementation and other notes, 577
 Showing data values at Numerical Limits, 92
 Simple Example, 84
 Struct `polygon_element`, 180, 182
 Tutorial: 1D More Layout Examples, 20
 Tutorial: 2D Special Features, 62
fill_color
 Class `plot_point_style`, 545, 546
 Class `svg_1d_plot_series`, 308, 309
 Class `svg_2d_plot_series`, 419, 421
 Class `svg_style`, 547, 548
 Class `tspan_element`, 202, 203, 204
fill_on
 Class `box_style`, 540, 541

Class path_element, 173, 174, 175
Class svg_style, 547, 548
Class tspan_element, 202, 204

font
 1-D Data Values Examples, 43
 2-D Data Values Examples, 67
 Class plot_point_style, 545, 546
 Class svg_1d_plot, 215, 232, 241, 242, 244, 245, 257, 258, 259, 260, 261, 271, 272, 291, 292, 293, 295, 296, 304
 Class svg_2d_plot, 312, 342, 343, 345, 346, 358, 359, 360, 361, 371, 372, 389, 390, 392, 394, 401, 402, 403, 409, 410, 411, 415
 Class svg_2d_plot_series, 422, 423
 Class svg_boxplot, 425, 453, 454, 455, 457, 458, 470, 471, 472, 473, 484, 485, 503, 504, 506, 508, 513, 518
 Class svg_boxplot_series, 521
 Class svg_style, 547
 Class text_element, 196, 199
 Class text_style, 549, 550, 551, 552
 Class ticks_labels_style, 553
 Class tspan_element, 202, 204, 205, 206
 Class value_style, 556
Fonts, 8
Function default_font, 558
Function string_svg_length, 560
Global aspect_ratio, 146, 557
Global reducer, 144, 146
Header < boost/svg_plot/detail/axis_plot_frame.hpp >, 144
Header < boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp >, 145
Header < boost/svg_plot/detail/svg_tag.hpp >, 152
Header < boost/svg_plot/svg_style.hpp >, 534
Implementation and other notes, 577
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 2D Special Features, 62
Using Inkscape, 570

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, 550, 551

font_family
 Class text_style, 550, 551
 Class tspan_element, 202, 204

font_size
 Class text_style, 550, 551, 552
 Class tspan_element, 202, 204

font_stretch
 Class text_style, 550, 552

font_style
 Class text_style, 550, 552
 Class tspan_element, 202, 204
font_weight
 Class text_style, 550, 552
 Class tspan_element, 202, 204, 205
fpt_abs
 Function template fpt_abs, 150
 Header < boost/svg_plot/detail/fp_compare.hpp >, 147
freedom
 2-D Data Values Examples, 67
 Class template unc, 564
 Function template operator<<, 565
from
 Class line_element, 169, 170, 171
 Class curve_element, 186, 188
Function constant_to_rgb
 color, 532
 constant_to_rgb, 532
 example, 532
 SVG, 532
 svg_color, 532
 variables, 532
Function default_font
 font, 558
 legend, 558
 SVG, 557
 title, 558
 Unicode, 558
Function is_blank
 color, 532, 533
 is_blank, 532
 SVG, 532
Function median
 data, 211
 maximum, 211
 minimum, 211
 SVG, 211
Function operator!=
 color, 533
 colors, 533
 SVG, 208, 533, 558
Function operator<<
 color, 533
 data, 559
 example, 209, 210, 533, 559
 fill, 558
 marker, 559
 my_color, 533
 p, 210
 p0, 209
 points, 210
 r, 209
 RGB, 533
 string, 559
 stroke, 558
 style, 209
 SVG, 208, 209, 210, 533, 558, 559

t, 209
ts, 559
Function operator==
 color, 534
 colors, 534
 SVG, 210, 533, 559
Function outFmtFlags
 example, 135, 213
 outFmtFlags, 135, 212, 213
 SVG, 212
Function quantile
 boxplot, 212
 data, 211, 212
 example, 212
 maximum, 211, 212
 minimum, 211, 212
 quantile, 211, 212
 quartile, 212
 SVG, 211
Function rounddown10
 rounddown10, 138
 SVG, 138
Function rounddown2
 rounddown2, 138
 SVG, 138
Function rounddown5
 rounddown5, 138
 SVG, 138
Function roundup10
 roundup10, 139
 SVG, 139
Function roundup2
 roundup2, 139
 SVG, 139
Function roundup5
 roundup5, 139
 SVG, 139
Function scale_axis
 10, 140
 marker, 140
 maximum, 140, 141
 minimum, 140, 141
 origin, 140, 141
 scale_axis, 140
 SVG, 140
 uncertainty, 140
 values, 140
Function show_1d_plot_settings
 show_1d_plot_settings, 213
 SVG, 213
Function show_2d_plot_settings
 show_2d_plot_settings, 214
 SVG, 214
Function string_svg_length
 font, 560
 string_svg_length, 560
 SVG, 560
Function strip_e0s

string, 311
strip_e0s, 311
SVG, 311
Function template fpt_abs
 fpt_abs, 150
Function template isfinite
 isfinite, 136
 type, 136
Function template mnmx
 container, 137
 maximum, 136, 137
 minimum, 136, 137
 mnmx, 137
 SVG, 136
 values, 137
Function template operator<<
 example, 107, 128, 565
 freedom, 565
 minimum, 565
 SVG, 564, 565
 uncertainty, 128, 565
 Unicode, 565
 with, 128, 565
Function template range_all
 container, 137
 maximum, 137
 minimum, 137
 SVG, 137
Function template range_mx
 container, 137
 data, 137
 maximum, 137
 minimum, 137
 range_mx, 137
 series, 137
 SVG, 137
Function template safe_fpt_division
 safe_fpt_division, 150
Function template scale_axis
 10, 141, 142, 143
 container, 141, 142, 143
 data, 141, 142, 143
 maximum, 141, 142, 143
 minimum, 141, 142, 143
 origin, 141, 142, 143
 plot, 141
 scale_axis, 141, 142
 scaling, 141
 series, 141, 142, 143
 SVG, 141, 142
 uncertainty, 141, 142
Function template show
 container, 143
 show, 143
 SVG, 143
Function template show_all
 container, 144
 show_all, 144

SVG, 144
Function template uncs_of
parts, 130, 567
SVG, 566, 567
uncertainties, 129, 566, 567
uncertainty, 129, 130, 566, 567
Function template unc_of
always, 129, 566
SVG, 565, 566
uncertainty, 129, 565, 566
unc_of, 129, 565, 566
Function template values_of
SVG, 568, 569
uncertainty, 568, 569
values_of, 131, 568, 569
Function template value_of
SVG, 567, 568
uncertainty, 130, 567, 568
value_of, 130, 567, 568
Function uFlags
uncertainty, 128, 129
functor
Class svg_2d_plot, 351

G

g Class g_element, 163, 165

Simple Example, 84

Tutorial: 1D More Layout Examples, 20

Tutorial: Fuller Layout Example, 60

generate_text
Class text_element, 196, 197

Global aspect_ratio
font, 146, 557
SVG, 146, 557
title, 146, 557

Global color_array
color, 531, 532
SVG, 531

Global document_ids_
boxplot, 152
SVG, 152

Global fmtFlagWords
SVG, 212

Global no_style
SVG, 557

Global plusminus
SVG, 564
uncertainty, 564

Global reducer
font, 144, 146
SVG, 144, 146
uncertainty, 144, 146

Global sin45
label, 145, 146
scaling, 145, 146
SVG, 145, 146

Global spacing_factor
 data, 147
 legend, 147
 series, 147
 SVG, 147
Global text_plusminus
 SVG, 145, 147
Graphic
 Header < boost/svg_plot/detail/auto_axes.hpp >, 135
 Header < boost/svg_plot/svg_1d_plot.hpp >, 214
 Header < boost/svg_plot/svg_boxplot.hpp >, 424
greek
 Class svg_1d_plot, 236, 256, 257, 305
 Class svg_2d_plot, 336, 356, 357, 416
 Class svg_boxplot, 447, 469
 Class text_element, 196
Fonts, 8
 Implementation and other notes, 577
grid
 Class svg_1d_plot, 262, 275, 276, 281, 282, 302, 306
 Class svg_2d_plot, 338, 374, 375, 376, 380, 381, 403, 404, 405, 406, 413, 417
 Class svg_boxplot, 449, 487, 488, 493, 494, 495, 516, 520
 Class ticks_labels_style, 553
 Tutorial: 1D Gridlines & Axes - more examples, 22
 Tutorial: 2D Special Features, 62
g_element
 Class g_element, 163, 164

H

h
 Class path_element, 173, 175
 Tutorial: 1D More Layout Examples, 20
 Tutorial: Fuller Layout Example, 60

H
 Class path_element, 173, 175

header
 Class svg_1d_plot, 303
 Class svg_2d_plot, 414
 Class svg_boxplot, 517

Header < boost/quan/meas.hpp >
 uncertainty, 94
 uncun, 94

Header < boost/quan/meas2.hpp >
 uncertainty, 100

Header < boost/quan/pair_io.hpp >
 example, 101

Header < boost/quan/si_units.hpp >
 container, 101

Header < boost/quan/unc.hpp >
 autoscale, 107
 example, 107
 layout, 107
 uncertainty, 107
 uncun, 107
 unc_of, 107
 values_of, 107
 value_of, 107

Header < boost/quan/uncdata.hpp >
 uncertainty, 131

Header < boost/quan/uncs.hpp >
 autoscale, 132
 uncertainty, 132

Header < boost/quan/unc_init.hpp >
 data, 131
 uncertainty, 131

Header < boost/quan/xiosstream.hpp >
 outFmtFlags, 132

Header < boost/svg_plot/detail/auto_axes.hpp >
 container, 135
 data, 135
 Graphic, 135
 isfinite, 135
 maximum, 135
 minimum, 135
 mnmx, 135
 range_mx, 135
 rounddown10, 135
 rounddown2, 135
 rounddown5, 135
 roundup10, 135
 roundup2, 135
 roundup5, 135
 scale_axis, 135
 show, 135
 show_all, 135
 SVG, 135
 values, 135

Header < boost/svg_plot/detail/axis_plot_frame.hpp >
 BOOST_SVG_LEGEND_DIAGNOSTICS, 144
 BOOST_SVG_POINT_DIAGNOSTICS, 144
 BOOST_SVG_TITLE_DIAGNOSTICS, 144
 boxplot, 144
 data, 144
 font, 144
 intersect, 144
 label, 144
 legend, 144
 marker, 144
 scaling, 144
 SVG, 144
 title, 144
 uncertainty, 144

Header < boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp >
 boxplot, 145
 data, 145
 font, 145
 intersect, 145
 label, 145
 legend, 145
 maximum, 145
 maxof3, 145
 scaling, 145
 series, 145
 SVG, 145
 title, 145

uncertainty, 145
Header < boost/svg_plot/detail/fp_compare.hpp >
epsilon, 147
fpt_abs, 147
maximum, 147
minimum, 147
min_value, 147
neareq, 147
safe_fpt_division, 147
tiny, 147
Header < boost/svg_plot/detail/functors.hpp >
2D, 151
data, 151
example, 151
precision, 151
SVG, 151
uncertainty, 151
Header < boost/svg_plot/detail/numeric_limits_handling.hpp >
data, 151
maximum, 151
minimum, 151
Header < boost/svg_plot/detail/pair.hpp >
example, 151
Header < boost/svg_plot/detail/svg_boxplot_detail.hpp >
boxplot, 151
set_ids, 151
SVG, 151
Header < boost/svg_plot/detail/svg_tag.hpp >
font, 152
SVG, 152
Header < boost/svg_plot/quantile.hpp >
50, 210
data, 210
minimum, 210
quantile, 210
quartile, 210
scaling, 210
SVG, 210
Header < boost/svg_plot/show_1d_settings.hpp >
l_or_r, 212
outFmtFlags, 212
show_1d_plot_settings, 212
SVG, 212
t_or_b, 212
Header < boost/svg_plot/show_2d_settings.hpp >
example, 213
show_2d_plot_settings, 213
SVG, 213
Header < boost/svg_plot/svg_1d_plot.hpp >
data, 214
Graphic, 214
layout, 214
marker, 214
strip_e0s, 214
SVG, 214
Header < boost/svg_plot/svg_2d_plot.hpp >
SVG, 311
Header < boost/svg_plot/svg_boxplot.hpp >

boxplot, 424
data, 424
Graphic, 424
layout, 424
marker, 424
quartile, 424
SVG, 424
Header < boost/svg_plot/svg_color.hpp >
color, 528
constant_to_rgb, 528
is_blank, 528
SVG, 528
Header < boost/svg_plot/svg_style.hpp >
BOOST_SVG_STYLE_DIAGNOSTICS, 534
color, 534
data, 534
default_plot_point_style, 534
fill, 534
font, 534
label, 534
string_svg_length, 534
stroke, 534
SVG, 534
text, 534
title, 534
Unicode, 534
Header < boost/svg_plot/uncertain.hpp >
minimum, 560
SVG, 560
uncertainties, 560
uncertainty, 560
uncorr, 560
uncun, 560
unc_of, 560
values_of, 560
value_of, 560
heat_flow_data
 Real-life Heat flow data, 48
height
 Class rect_element, 189, 190
 Class svg_1d_plot, 239, 255, 300, 301
 Class svg_2d_plot, 340, 355, 408
 Class svg_boxplot, 452, 468
hexagon
 Class g_element, 163, 165
histogram
 Class bar_style, 538
 Class histogram_style, 541, 542
 Class svg_2d_plot, 335
 Class svg_2d_plot_series, 419, 421
Histograms of 2D data
 data, 82
histogram_style
 Class histogram_style, 542
History
 uncertainty, 576
Hoaglin
 Definitions of the Quartiles, 86

Hoaglin4
 Definitions of the Quartiles, 86
 Hoaglin5
 Definitions of the Quartiles, 86
 Hoaglin6
 Definitions of the Quartiles, 86
 Hoaglin7
 Definitions of the Quartiles, 86
 Hoaglin8
 Definitions of the Quartiles, 86
 horizontal
 Class ellipse_element, 160
 Class path_element, 175
 Class svg_2d_plot, 404
 How To Use This Documentation
 container, 5
 example, 5
 h_path
 Struct h_path, 168

I

id
 Class circle_element, 156, 157
 Class clip_path_element, 158, 159
 Class ellipse_element, 160, 161
 Class g_element, 163, 165
 Class line_element, 170, 171
 Class path_element, 173, 175
 Class polyline_element, 183, 184
 Class quurve_element, 186, 187
 Class rect_element, 189, 190
 Class svg_element, 193, 194
 Class text_element, 196, 198
 Class tspan_element, 202, 205
 Struct polygon_element, 180, 181, 182
 Implementation and other notes
 author, 577
 color, 577
 container, 577
 data, 577, 578
 description, 577
 document_title, 577
 example, 577
 fill, 577
 font, 577
 greek, 577
 is_license, 577
 label, 577
 legend, 577
 license_attribution, 577
 license_commercialuse, 577
 license_distribution, 577
 license_reproduction, 577
 marker, 577
 minimum, 577
 precision, 577, 578
 scaling, 577

series, 577
stroke, 577
style, 577
SVG, 577
title, 577
Unicode, 577
x_num_minor_ticks, 577

inequality
 Class text_style, 552

Inkscape
 Class text_style, 552
 Class tspan_element, 204

Fonts, 8

Preface, 3

To Do List, 582

Using Inkscape, 570

intersect
 1-D Auto scaling Examples, 31
 Class axis_line_style, 537, 538
 Class svg_1d_plot, 266, 267, 307
 Class svg_2d_plot, 312, 366
 Class svg_boxplot, 425, 479, 512
 Header < boost/svg_plot/detail/axis_plot_frame.hpp >, 144
 Header < boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp >, 145
 Tutorial: 1D Gridlines & Axes - more examples, 22

interval
 Class svg_1d_plot, 278
 Class svg_2d_plot, 312, 377
 Class svg_boxplot, 491

intervals
 Class svg_1d_plot, 232
 Class svg_2d_plot, 332
 Class svg_boxplot, 425, 442, 443

ioflags
 1-D Data Values Examples, 43
 2-D Data Values Examples, 67
 Class svg_1d_plot, 294
 Class svg_2d_plot, 393, 410
 Class svg_boxplot, 507
 Tutorial: 2D Special Features, 62

is
 Class svg_1d_plot, 232
 Class template unc, 118, 123

isfinite
 Function template isfinite, 136
 Header < boost/svg_plot/detail/auto_axes.hpp >, 135

isnan
 Showing data values at Numerical Limits, 92

is_blank
 Class svg_color, 529, 530
 Function is_blank, 532
 Header < boost/svg_plot/svg_color.hpp >, 528

is_in_window
 Class svg_2d_plot, 312, 340

is_license
 Implementation and other notes, 577

iter
 2-D Autoscaling Examples, 73

L

1

Class path_element, 173, 175

L

Class path_element, 173, 175

label

1-D Autoscaling Various Containers Examples, 40

1-D Data Values Examples, 43

2-D Data Values Examples, 67

Class axis_line_style, 536, 537

Class svg_1d_plot, 215, 232, 262, 266, 271, 272, 273, 274, 276, 277, 278, 290, 291, 292, 293, 294, 295, 299, 300, 305, 306

Class svg_2d_plot, 312, 335, 338, 365, 366, 371, 372, 373, 374, 376, 377, 389, 390, 392, 393, 399, 400, 401, 402, 403, 404, 410, 412, 416, 417

Class svg_boxplot, 425, 449, 478, 483, 484, 485, 486, 487, 489, 490, 503, 504, 506, 507, 512, 513, 514, 519

Class svg_boxplot_series, 521

Class ticks_labels_style, 553, 554

Class value_style, 555, 556

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, 145, 146

Header < boost/svg_plot/detail/axis_plot_frame.hpp >, 144

Header < boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp >, 145

Header < boost/svg_plot/svg_style.hpp >, 534

Implementation and other notes, 577

Real-life Heat flow data, 48

Simple Example, 84

To Do List, 582

Tutorial: 1D Gridlines & Axes - more examples, 22

Tutorial: 1D More Layout Examples, 20

Tutorial: 2D Special Features, 62

labels

Class svg_1d_plot, 278

Class svg_2d_plot, 312, 404

Class svg_boxplot, 490

label_length

Class ticks_labels_style, 553, 554

label_on

Class axis_line_style, 536, 537

label_units_on

Class axis_line_style, 536, 537

layout

Class svg_2d_plot, 311

Header < boost/quan/unc.hpp >, 107

Header < boost/svg_plot/svg_1d_plot.hpp >, 214

Header < boost/svg_plot/svg_boxplot.hpp >, 424

Tutorial: 1D Gridlines & Axes - more examples, 22

Tutorial: 1D More Layout Examples, 20

Tutorial: Fuller Layout Example, 60

left

Class svg_boxplot, 514

legend

1-D Auto scaling Examples, 31

1-D Vector Example, 14

Class g_element, 167

Class svg_1d_plot, 215, 232, 239, 240, 241, 242, 243, 244, 245, 246, 262, 301, 302, 303, 306, 307

Class `svg_1d_plot_series`, 308
Class `svg_2d_plot`, 312, 328, 340, 341, 342, 343, 344, 345, 346, 347, 412, 413, 414, 417, 418
Class `svg_2d_plot_series`, 419, 420, 422
Class `svg_boxplot`, 425, 442, 452, 453, 454, 455, 456, 457, 458, 459, 516, 517, 520
Demonstration of using 1D data that includes information about its Uncertainty, 54
Fonts, 8
Function `default_font`, 558
Global `spacing_factor`, 147
Header <boost/svg_plot/detail/axis_plot_frame.hpp>, 144
Header <boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp>, 145
Implementation and other notes, 577
Simple Code Example, 58, 59
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 1D More Layout Examples, 20
Tutorial: Fuller Layout Example, 60
Type `svg_color_constant`, 530
`legend_background_color`
 Class `svg_1d_plot`, 215, 239
 Class `svg_2d_plot`, 312, 340
 Class `svg_boxplot`, 425, 452
`legend_border_color`
 Class `svg_1d_plot`, 215, 240
 Class `svg_2d_plot`, 312, 340, 341
 Class `svg_boxplot`, 425, 452, 453
`legend_box_fill_on`
 Class `svg_1d_plot`, 215, 240
 Class `svg_2d_plot`, 312, 341
 Class `svg_boxplot`, 425, 453
`legend_color`
 Class `svg_1d_plot`, 215, 240, 241
 Class `svg_2d_plot`, 312, 341
 Class `svg_boxplot`, 425, 453
`legend_font_family`
 Class `svg_1d_plot`, 215, 241
 Class `svg_2d_plot`, 312, 342
 Class `svg_boxplot`, 425, 453, 454
`legend_font_size`
 Class `svg_1d_plot`, 215, 241
 Class `svg_2d_plot`, 312, 342
 Class `svg_boxplot`, 425, 454
`legend_font_weight`
 Class `svg_1d_plot`, 215, 241, 242
 Class `svg_2d_plot`, 312, 342, 343
 Class `svg_boxplot`, 425, 454
`legend_header_font_size`
 Class `svg_1d_plot`, 215, 242
 Class `svg_2d_plot`, 312, 343
 Class `svg_boxplot`, 425, 455
`legend_lines`
 Class `svg_1d_plot`, 215, 242, 243
 Class `svg_2d_plot`, 312, 343
 Class `svg_boxplot`, 425, 455
`legend_on`
 Class `svg_1d_plot`, 215, 243
 Class `svg_2d_plot`, 312, 343, 344
 Class `svg_boxplot`, 425, 455, 456
`legend_outside`
 Class `svg_1d_plot`, 215, 243

Class `svg_2d_plot`, 312, 344
Class `svg_boxplot`, 425, 456
`legend_place`
 Class `svg_1d_plot`, 215, 243, 244
 Class `svg_2d_plot`, 312, 344, 345
 Class `svg_boxplot`, 425, 456, 457
`legend_text_font_size`
 Class `svg_1d_plot`, 215, 244
 Class `svg_2d_plot`, 312, 345
 Class `svg_boxplot`, 425, 457
`legend_text_font_weight`
 Class `svg_1d_plot`, 215, 244
 Class `svg_2d_plot`, 312, 345
 Class `svg_boxplot`, 425, 457
`legend_title`
 Class `svg_1d_plot`, 215, 245
 Class `svg_2d_plot`, 312, 345, 346
 Class `svg_boxplot`, 425, 457, 458
`legend_title_font_size`
 Class `svg_1d_plot`, 215, 245
 Class `svg_2d_plot`, 312, 346
 Class `svg_boxplot`, 425, 458
`legend_title_font_weight`
 Class `svg_1d_plot`, 215, 245
 Class `svg_2d_plot`, 312, 346
 Class `svg_boxplot`, 425, 458
`legend_top_left`
 Class `svg_1d_plot`, 215, 246
 Class `svg_2d_plot`, 312, 346, 347
 Class `svg_boxplot`, 425, 458, 459
`legend_width`
 Class `svg_boxplot`, 474
 Class `svg_boxplot_series`, 528
 Class `tspan_element`, 206
`license`
 Class `svg_1d_plot`, 215, 246, 247
 Class `svg_2d_plot`, 312, 347
 Class `svg_boxplot`, 425, 459
`license_attribution`
 Class `svg_1d_plot`, 215, 247
 Class `svg_2d_plot`, 312, 348
 Class `svg_boxplot`, 425, 460
 Implementation and other notes, 577
`license_commercialuse`
 Class `svg_1d_plot`, 215, 247
 Class `svg_2d_plot`, 312, 348
 Class `svg_boxplot`, 425, 460
 Implementation and other notes, 577
`license_distribution`
 Class `svg_1d_plot`, 215, 247
 Class `svg_2d_plot`, 312, 348
 Class `svg_boxplot`, 425, 460
 Implementation and other notes, 577
`license_on`

Class `svg_1d_plot`, 215, 247, 248
 Class `svg_2d_plot`, 312, 348, 349
 Class `svg_boxplot`, 425, 460, 461
license_reproduction
 Class `svg_1d_plot`, 215, 248
 Class `svg_2d_plot`, 312, 349
 Class `svg_boxplot`, 425, 461
 Implementation and other notes, 577
limits_count
 Class `svg_2d_plot_series`, 419, 422
limit_color
 Class `svg_1d_plot`, 215, 248, 249
 Class `svg_2d_plot`, 312, 349
 Class `svg_boxplot`, 425, 461
limit_fill_color
 Class `svg_1d_plot`, 215, 249
 Class `svg_2d_plot`, 312, 349, 350
 Class `svg_boxplot`, 425, 461, 462
line
 Class `axis_line_style`, 536, 538
 Class `bar_style`, 538
 Class `box_style`, 540
 Class `g_element`, 163, 165
 Class `plot_line_style`, 543, 544
 Class `svg_1d_plot`, 277, 278
 Class `svg_2d_plot`, 404
 Class `svg_2d_plot_series`, 420, 421, 422
 Class `svg_boxplot`, 479, 490, 512
 Class `value_style`, 556
 Demonstration of adding lines and curves, typically a least squares fit, 80
lines
 Class `svg_2d_plot`, 331
line_color
 Class `svg_1d_plot_series`, 308, 309
 Class `svg_2d_plot_series`, 419, 422
line_element
 Class `line_element`, 170
line_on
 Class `plot_line_style`, 543, 544
 Class `svg_1d_plot_series`, 308, 309
 Class `svg_2d_plot_series`, 419, 422
line_style
 Class `svg_2d_plot_series`, 419, 422
line_width
 Class `svg_1d_plot_series`, 308, 309
 Class `svg_2d_plot_series`, 419, 422
longest_label
 Class `ticks_labels_style`, 553, 554
l_or_r
 Header <`boost/svg_plot/show_1d_settings.hpp`>, 212
l_path
 Struct `l_path`, 169

M

M
 Class `path_element`, 173, 176
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, 553, 554
margin
 Class box_style, 540, 541
marker
 1-D Data Values Examples, 43
 1-D Vector Example, 14
 2-D Data Values Examples, 67
 Class plot_point_style, 544, 546
 Class svg_1d_plot, 215, 256, 263, 268, 269, 270, 285, 286, 292, 293, 295, 296, 297, 298, 303, 304, 307
 Class svg_1d_plot_series, 308, 309, 310
 Class svg_2d_plot, 311, 312, 357, 362, 363, 368, 369, 370, 384, 385, 390, 391, 393, 394, 395, 396, 414, 415, 418
 Class svg_2d_plot_series, 419, 420, 421, 422, 423, 424
 Class svg_boxplot, 425, 449, 450, 463, 464, 475, 476, 480, 481, 482, 483, 497, 498, 504, 505, 507, 508, 509, 510, 517, 518, 520
 Class svg_boxplot_series, 521, 525, 527
 Demonstration of using 1D data that includes information about its Uncertainty, 54
Function operator<<, 559
Function scale_axis, 140
Header < boost/svg_plot/detail/axis_plot_frame.hpp >, 144
Header < boost/svg_plot/svg_1d_plot.hpp >, 214
Header < boost/svg_plot/svg_boxplot.hpp >, 424
Implementation and other notes, 577
Simple Code Example, 58
Tutorial: Fuller Layout Example, 60
markers
 Class svg_1d_plot, 307
 Class svg_2d_plot, 418
 Class svg_boxplot, 520
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, 536
 Class svg_1d_plot, 215, 253, 254, 263, 265, 280, 281, 286, 301
 Class svg_2d_plot, 312, 353, 354, 363, 364, 379, 380, 385, 386, 397, 398, 405, 407, 412
 Class svg_boxplot, 425, 448, 464, 466, 467, 476, 477, 492, 493, 499, 511, 515, 516
 Class svg_boxplot_series, 521, 523, 525, 528
 Class template unc, 122, 127, 562
 Class ticks_labels_style, 553, 554
Colors, 6
Definitions of the Quartiles, 86
Function median, 211
Function quantile, 211, 212
Function scale_axis, 140, 141
Function template mnmx, 136, 137
Function template range_all, 137
Function template range_mx, 137
Function template scale_axis, 141, 142, 143
Header < boost/svg_plot/detail/auto_axes.hpp >, 135

Header < boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp >, 145
Header < boost/svg_plot/detail/fp_compare.hpp >, 147
Header < boost/svg_plot/detail/numeric_limits_handling.hpp >, 151
Showing data values at Numerical Limits, 92
Tutorial: 1D Autoscale with Multiple Containers, 18
maxof3
 Header < boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp >, 145
max_whisker_color
 Class svg_boxplot_series, 521, 525
max_whisker_width
 Class svg_boxplot_series, 521, 525
Meas
 Class Meas, 95
median_color
 Class svg_boxplot, 425, 462
 Class svg_boxplot_series, 521, 525
median_style
 Class svg_boxplot_series, 521, 526
median_width
 Class svg_boxplot, 425, 462
 Class svg_boxplot_series, 521, 526
middle
 Class svg_1d_plot, 215
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, 583
 Class axis_line_style, 536
 Class svg_1d_plot, 215, 253, 254, 263, 265, 280, 281, 282, 283, 284, 286, 300, 301, 304, 306
 Class svg_2d_plot, 312, 338, 353, 354, 363, 364, 379, 380, 381, 382, 383, 385, 386, 397, 398, 405, 406, 407, 412, 415, 417
 Class svg_boxplot, 425, 448, 449, 464, 466, 467, 476, 477, 492, 493, 494, 495, 496, 497, 499, 511, 514, 515, 516, 518, 520
 Class svg_boxplot_series, 521, 523, 525, 526, 528
 Class template smallest, 149
 Class template unc, 564
 Class text_style, 551
 Class ticks_labels_style, 553, 554
 Class value_style, 557
Colors, 6
Definitions of the Quartiles, 86, 87
Fonts, 8
Function median, 211
Function quantile, 211, 212
Function scale_axis, 140, 141
Function template mnmx, 136, 137
Function template operator<<, 565
Function template range_all, 137
Function template range_mx, 137
Function template scale_axis, 141, 142, 143
Header < boost/svg_plot/detail/auto_axes.hpp >, 135
Header < boost/svg_plot/detail/fp_compare.hpp >, 147
Header < boost/svg_plot/detail/numeric_limits_handling.hpp >, 151
Header < boost/svg_plot/quantile.hpp >, 210
Header < boost/svg_plot/uncertain.hpp >, 560
Implementation and other notes, 577
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`, 531
 Using Inkscape, 570
`min_value`
 Header <`boost/svg_plot/detail/fp_compare.hpp`>, 147
`min_whisker_color`
 Class `svg_boxplot_series`, 521, 526
`min_whisker_width`
 Class `svg_boxplot_series`, 521, 526
`mnnmx`
 Function template `mnnmx`, 137
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135
`my_blank`
 Class `svg_color`, 529
`my_color`
 Function operator`<<`, 533
`my_plot`
 Class `svg_1d_plot`, 230
 Class `svg_2d_plot`, 330
`m_path`
 Struct `m_path`, 172

N

`neareq`
 Header <`boost/svg_plot/detail/fp_compare.hpp`>, 147
`not`
 Class `svg_1d_plot_series`, 308
 Plotting Graphs in SVG format., 2
`NOT`
 Class `svg_1d_plot`, 306
 Class `svg_2d_plot`, 417
 Class `svg_boxplot`, 520

O

`of`
 Class template `unc`, 564
`one_sd_color`
 Class `svg_1d_plot`, 215, 249
 Class `svg_2d_plot`, 312, 350
 Class `svg_boxplot`, 425, 462, 463
`operator`
 Class template `close_to`, 148, 149
 Class template `smallest`, 149, 150
 Struct template `lessAbs`, 110
`optional`
 1-D Vector Example, 14
`origin`
 1-D Auto scaling Examples, 31
 1-D Axis Scaling, 26
 Class `svg_1d_plot`, 215
 Class `svg_boxplot`, 425
 Demonstration of adding lines and curves, typically a least squares fit, 80
 Function `scale_axis`, 140, 141
 Function template `scale_axis`, 141, 142, 143
`outFmtFlags`

Function outFmtFlags, 135, 212, 213
 Header < boost/quan/xiostream.hpp >, 132
 Header < boost/svg_plot/show_1d_settings.hpp >, 212
 outlier
 Class svg_boxplot, 425, 449, 450, 463, 464
 Class svg_boxplot_series, 524, 525, 526, 527
 outlier_color
 Class svg_boxplot, 425, 463
 Class svg_boxplot_series, 521, 526
 outlier_fill
 Class svg_boxplot, 425, 463
 Class svg_boxplot_series, 521, 527
 outlier_shape
 Class svg_boxplot, 425, 463
 Class svg_boxplot_series, 521, 527
 outlier_size
 Class svg_boxplot, 425, 463
 Class svg_boxplot_series, 521, 527
 outlier_style
 Class svg_boxplot, 425, 464
 Class svg_boxplot_series, 521, 527
 outlier_values_on
 Class svg_boxplot, 425, 464

P

P
 Class polyline_element, 183, 185
 Struct polygon_element, 180, 182
 p
 Function operator<<, 210
 Struct polygon_element, 182
 p0
 Function operator<<, 209
 part
 1-D Auto scaling Examples, 31
 1-D Axis Scaling, 26
 parts
 Function template uncs_of, 130, 567
 path
 Class g_element, 163, 166
 path_element
 Class path_element, 173
 path_point
 Struct a_path, 154
 Struct c_path, 155
 Struct h_path, 168
 Struct l_path, 169
 Struct m_path, 172
 Struct path_point, 178
 Struct q_path, 185
 Struct s_path, 192
 Struct t_path, 195
 Struct v_path, 207
 Struct z_path, 208
 pentagon
 Class g_element, 163, 166
 place_legend_box

Class `svg_1d_plot`, 215, 306
 Class `svg_2d_plot`, 312, 417
 Class `svg_boxplot`, 425, 520
plot
 Class `svg_1d_plot`, 215, 250, 251, 256, 257
 Class `svg_2d_plot`, 312, 350, 351, 356, 357
 Class `svg_boxplot`, 425, 464, 465
 Colors, 6
 Function template `scale_axis`, 141
 Showing data values at Numerical Limits, 92
 Simple Code Example, 58
 Tutorial: 2D Special Features, 62
Plotting Graphs in SVG format.
 not, 2
 SVG, 2
plot_background_color
 Class `svg_1d_plot`, 215, 251
 Class `svg_2d_plot`, 312, 351, 352
 Class `svg_boxplot`, 425, 465
plot_border_color
 Class `svg_1d_plot`, 215, 252
 Class `svg_2d_plot`, 312, 352
 Class `svg_boxplot`, 425, 465
plot_border_width
 Class `svg_1d_plot`, 215, 252
 Class `svg_2d_plot`, 312, 352
 Class `svg_boxplot`, 425, 465
plot_line_style
 Class `plot_line_style`, 543
plot_point_style
 Class `plot_point_style`, 545
plot_window_x_right
 Class `svg_1d_plot`, 215, 254
 Class `svg_2d_plot`, 312, 354
 Class `svg_boxplot`, 425, 467
plot_window_y
 Class `svg_1d_plot`, 215, 254
 Class `svg_2d_plot`, 312, 354
 Class `svg_boxplot`, 425, 467
plot_window_y_bottom
 Class `svg_1d_plot`, 215, 254
 Class `svg_2d_plot`, 312, 354
 Class `svg_boxplot`, 425, 467
plot_window_y_top
 Class `svg_1d_plot`, 215, 254
 Class `svg_2d_plot`, 312, 355
 Class `svg_boxplot`, 425, 468
point
 Class `plot_point_style`, 545, 546
 Class `curve_element`, 186, 188
 Class `svg_1d_plot`, 304
 Class `svg_2d_plot`, 415
 Class `svg_boxplot`, 518
 Struct `h_path`, 168
 Struct `polygon_element`, 182
 Struct `v_path`, 207
points
 Class `svg_2d_plot`, 338

Function operator<<, 210
point_font_decoration
 Class svg_2d_plot_series, 419, 422
point_font_family
 Class svg_2d_plot_series, 419, 422, 423
point_font_stretch
 Class svg_2d_plot_series, 419, 423
point_font_style
 Class svg_2d_plot_series, 419, 423
point_font_weight
 Class svg_2d_plot_series, 419, 423
point_style
 Class svg_2d_plot_series, 419, 423
polygon
 Class g_element, 163, 166
 Struct polygon_element, 180
polygon_element
 Struct polygon_element, 180
polyline
 Class g_element, 163, 166
polyline_element
 Class polyline_element, 183
poly_path_point
 Struct poly_path_point, 179
position
 Class axis_line_style, 536, 537
 Class svg_1d_plot, 266, 267
 Class svg_2d_plot, 366
precision
 1-D Data Values Examples, 43
 2-D Data Values Examples, 67
 Class svg_1d_plot, 232, 233, 262, 292, 293, 294, 295, 297, 304
 Class svg_2d_plot, 312, 332, 333, 391, 393, 395, 396, 410, 411, 412, 415
 Class svg_boxplot, 443, 505, 507, 509, 510, 518
 Class template unc, 118, 123
 Class ticks_labels_style, 553
 Class value_style, 556
 Demonstration of using 1D data that includes information about its Uncertainty, 54
Fonts, 8
Header < boost/svg_plot/detail/functors.hpp >, 151
Implementation and other notes, 577, 578
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
push_back
 Class g_element, 163, 166

Q

q
 Class path_element, 173, 176
Q
 Class path_element, 173, 176
quantile
 Definitions of the Quartiles, 86
 Function quantile, 211, 212
 Header < boost/svg_plot/quantile.hpp >, 210
quartile
 Class svg_boxplot, 425, 464, 468
 Class svg_boxplot_series, 521, 523, 524, 527
 Definitions of the Quartiles, 86, 87
 Function quantile, 212
 Header < boost/svg_plot/quantile.hpp >, 210
 Header < boost/svg_plot/svg_boxplot.hpp >, 424
 Real-life Heat flow data, 48
quartile_definition
 Class svg_boxplot, 425, 468
 Class svg_boxplot_series, 521, 527
curve_element
 Class curve_element, 186
q_path
 Struct q_path, 185

R

r
 Function operator<<, 209
range
 Class svg_2d_plot, 407
range_mx
 Function template range_mx, 137
 Header < boost/svg_plot/detail/auto_axes.hpp >, 135
Real-life Heat flow data
 boxplot, 48
 data, 48
 example, 48
 heat_flow_data, 48
 label, 48
 main, 48
 precision, 48
 quartile, 48
 scaling, 48
 SVG, 48
 title, 48
rect
 Class g_element, 163, 166
rect_element
 Class rect_element, 189
rect_elements
 Class rect_element, 190
red
 Class svg_color, 529, 530
RGB
 1-D Auto scaling Examples, 31
 1-D Data Values Examples, 43
 2-D Data Values Examples, 67

- Function operator<<, 533
- rhombus
 - Class g_element, 163, 167
- right
 - Class svg_1d_plot, 253
 - Class svg_2d_plot, 353
 - Class svg_boxplot, 466
 - Class ticks_labels_style, 554
- rotation
 - Class svg_1d_plot, 297
 - Class svg_2d_plot, 396
 - Class svg_boxplot, 510
 - Class text_element, 196, 198
 - Class tspan_element, 202, 205
- rounddown10
 - Function rounddown10, 138
 - Header < boost/svg_plot/detail/auto_axes.hpp >, 135
- rounddown2
 - Function rounddown2, 138
 - Header < boost/svg_plot/detail/auto_axes.hpp >, 135
- rounddown5
 - Function rounddown5, 138
 - Header < boost/svg_plot/detail/auto_axes.hpp >, 135
- roundup10
 - Function roundup10, 139
 - Header < boost/svg_plot/detail/auto_axes.hpp >, 135
- roundup2
 - Function roundup2, 139
 - Header < boost/svg_plot/detail/auto_axes.hpp >, 135
- roundup5
 - Function roundup5, 139
 - Header < boost/svg_plot/detail/auto_axes.hpp >, 135

S

- s
 - 2-D Autoscaling Examples, 73
 - Class path_element, 173, 176
- S
 - Class path_element, 173, 176
- safe_fpt_division
 - Function template safe_fpt_division, 150
 - Header < boost/svg_plot/detail/fp_compare.hpp >, 147
- same
 - Class svg_1d_plot, 241
 - Class svg_2d_plot, 335, 342
 - Class svg_boxplot, 454
- SAS
 - Definitions of the Quartiles, 86
- scale_axis
 - Class svg_2d_plot, 312
 - Function scale_axis, 140
 - Function template scale_axis, 141, 142
 - Header < boost/svg_plot/detail/auto_axes.hpp >, 135
- scaling
 - 1-D Auto scaling Examples, 31
 - 1-D Axis Scaling, 26
 - 1-D Vector Example, 14

Class `axis_line_style`, 536
Class `path_element`, 173
Class `svg_1d_plot`, 215, 232, 241, 253, 254, 262, 264, 265, 271, 272, 291
Class `svg_2d_plot`, 312, 331, 335, 342, 353, 354, 364, 365, 371, 372, 389, 390, 397, 398, 401, 409, 411
Class `svg_boxplot`, 425, 442, 454, 466, 467, 477, 484, 503, 504, 511
Class `svg_boxplot_series`, 521, 524
Class `template unc`, 561
Class `text_style`, 550
Definitions of the Quartiles, 86, 87
Demonstration of adding lines and curves, typically a least squares fit, 80
Fonts, 8
Function template `scale_axis`, 141
Global `sin45`, 145, 146
Header <`boost/svg_plot/detail/axis_plot_frame.hpp`>, 144
Header <`boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp`>, 145
Header <`boost/svg_plot/quantile.hpp`>, 210
Implementation and other notes, 577
Real-life Heat flow data, 48
Showing data values at Numerical Limits, 92
Struct `polygon_element`, 180
To Do List, 582
Tutorial: 1D Gridlines & Axes - more examples, 22
Tutorial: 2D Special Features, 62
Using Inkscape, 570
scientific
 Class `svg_1d_plot`, 262
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`, 542, 543
 Class `svg_1d_plot`, 215, 242, 250, 251, 262, 265, 303, 307
 Class `svg_1d_plot_series`, 307, 308, 310
 Class `svg_2d_plot`, 312, 335, 337, 338, 343, 350, 351, 365, 397, 398, 414, 418
 Class `svg_2d_plot_series`, 419, 420, 421, 422, 423, 424
 Class `svg_boxplot`, 424, 425, 447, 455, 464, 465, 477, 517, 520
 Class `svg_boxplot_series`, 521, 522, 524, 528
 Class `value_style`, 555
 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`, 137
 Function template `scale_axis`, 141, 142, 143
 Global `spacing_factor`, 147
 Header <`boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp`>, 145
 Implementation and other notes, 577
 Simple Code Example, 58, 59
 Simple Example, 84
 To Do List, 582
 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`, 308, 310

series_limits_count
 Class `svg_1d_plot_series`, 308, 310

set
 Demonstration of using 1D data that includes information about its Uncertainty, 54
 Demonstration of using 2D data that includes information about its uncertainty, 76

set_ids
 Class `svg_1d_plot`, 215, 255
 Class `svg_2d_plot`, 312, 355
 Header <`boost/svg_plot/detail/svg_boxplot_detail.hpp`>, 151

shape
 Class `plot_point_style`, 545, 546
 Class `svg_1d_plot_series`, 308, 310
 Class `svg_2d_plot_series`, 419, 423

show
 Function template `show`, 143
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135

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

shown
 Class `svg_1d_plot`, 297
 Class `svg_2d_plot`, 312, 395, 397, 411
 Class `svg_boxplot`, 509

show_1d_plot_settings
 1-D Auto scaling Examples, 31
 1-D Data Values Examples, 43
 Function `show_1d_plot_settings`, 213
 Header <`boost/svg_plot/show_1d_settings.hpp`>, 212
 Tutorial: 2D Special Features, 62

show_2d_plot_settings
 2-D Data Values Examples, 67
 Function `show_2d_plot_settings`, 214
 Header <`boost/svg_plot/show_2d_settings.hpp`>, 213

show_all
 Function template `show_all`, 144
 Header <`boost/svg_plot/detail/auto_axes.hpp`>, 135

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, 582
size
 Class g_element, 163, 167
 Class plot_point_style, 545, 546
 Class svg_1d_plot, 215, 238, 239, 242, 255, 272, 287, 291, 292, 293, 301
 Class svg_1d_plot_series, 308, 310
 Class svg_2d_plot, 312, 339, 340, 343, 355, 372, 386, 390, 392, 408
 Class svg_2d_plot_series, 419, 423
 Class svg_boxplot, 425, 451, 452, 455, 468, 484, 485, 499, 504, 506, 515
 Class template close_to, 148, 149
 Class template smallest, 149, 150
 Class text_style, 550, 551, 552
 Using Inkscape, 570
size_legend_box
 Class svg_1d_plot, 215, 307
 Class svg_2d_plot, 312, 418
 Class svg_boxplot, 425, 520
smallest
 Class template smallest, 149
sqrt
 Tutorial: 1D More Layout Examples, 20
 Tutorial: Fuller Layout Example, 60
strength
 Class template close_to, 148, 149
string
 Class svg_1d_plot, 236
 Class svg_2d_plot, 336
 Class svg_boxplot, 447
 Function operator<<, 559
 Function strip_e0s, 311
string_svg_length
 Function string_svg_length, 560
 Header < boost/svg_plot/svg_style.hpp >, 534
strip_e0s
 Function strip_e0s, 311
 Header < boost/svg_plot/svg_1d_plot.hpp >, 214
stroke
 1-D Auto scaling Examples, 31

1-D Data Values Examples, 43
2-D Data Values Examples, 67
Class axis_line_style, 536
Class bar_style, 538
Class box_style, 540, 541
Class circle_element, 158
Class clip_path_element, 160
Class ellipse_element, 162
Class g_element, 167
Class line_element, 172
Class path_element, 177
Class plot_line_style, 543
Class plot_point_style, 545, 546
Class polyline_element, 185
Class curve_element, 188
Class rect_element, 191
Class svg_1d_plot, 240, 248, 249, 304
Class svg_1d_plot_series, 309, 310
Class svg_2d_plot, 340, 341, 349, 398, 399, 403, 405, 415
Class svg_2d_plot_series, 421, 422, 424
Class svg_boxplot, 452, 453, 461, 518
Class svg_element, 193, 194
Class svg_style, 547, 548, 549
Class text_element, 200
Class ticks_labels_style, 553
Class tspan_element, 207
Class value_style, 556
Function operator<<, 558
Header < boost/svg_plot/svg_style.hpp >, 534
Implementation and other notes, 577
Struct polygon_element, 182
stroke_color
 Class plot_point_style, 545, 546
 Class svg_1d_plot_series, 308, 310
 Class svg_2d_plot_series, 419, 424
 Class svg_style, 547, 548
stroke_on
 Class svg_style, 547, 548
stroke_width
 Class svg_style, 547, 549
Struct a_path
 a_path, 154
 data, 154
 path_point, 154
 SVG, 153, 154
 svg, 154
 to, 153
 write, 154
Struct c_path
 c_path, 155
 data, 155
 path_point, 155
 SVG, 154, 155
 svg, 155
 to, 154
 write, 155
Struct h_path
 data, 168

h_path, 168
path_point, 168
point, 168
SVG, 168
svg, 168
write, 168
Struct l_path
 data, 169
 l_path, 169
 path_point, 169
 SVG, 168, 169
 svg, 169
 write, 169
Struct m_path
 coordinates, 172
 data, 172
 example, 172
 m_path, 172
 path_point, 172
 SVG, 172
 svg, 172
 write, 172
Struct path_point
 data, 178
 path_point, 178
 SVG, 177, 178
 write, 178
Struct polygon_element
 absolute, 180, 181
 class_id, 180, 181
 clip_id, 180, 181
 color, 180, 182
 container, 180
 data, 180
 example, 181, 182
 fill, 180, 182
 id, 180, 181, 182
 P, 180, 182
 p, 182
 point, 182
 polygon, 180
 polygon_element, 180
 scaling, 180
 stroke, 182
 style, 180, 182
 SVG, 179, 180, 181, 182
 svg, 180
 svg_element, 180
 write, 180, 182
 write_attributes, 180, 182
Struct poly_path_point
 coordinates, 178
 data, 179
 example, 179
 poly_path_point, 179
 SVG, 178, 179
 write, 179
Struct q_path

data, 185
path_point, 185
q_path, 185
SVG, 185, 186
svg, 185
to, 185
write, 185, 186

Struct s_path
 data, 192
 path_point, 192
 SVG, 191, 192
 svg, 192
 s_path, 192
 to, 191
 write, 192

Struct template lessAbs
 operator, 110

Struct t_path
 data, 195
 path_point, 195
 SVG, 195
 svg, 195
 to, 195
 t_path, 195
 write, 195

Struct unit
 data, 102

Struct v_path
 data, 207
 path_point, 207
 point, 207
 SVG, 207
 svg, 207
 v_path, 207
 write, 207

Struct z_path
 data, 208
 path_point, 208
 SVG, 207, 208
 svg, 208
 write, 208
 z_path, 208

style
 Class circle_element, 156, 157
 Class clip_path_element, 158, 159
 Class ellipse_element, 160, 161, 162
 Class g_element, 163, 167
 Class line_element, 170, 171
 Class path_element, 173, 176
 Class plot_point_style, 545, 546
 Class polyline_element, 183, 185
 Class curve_element, 186, 188
 Class rect_element, 189, 191
 Class svg_boxplot, 425
 Class svg_element, 193, 195
 Class text_element, 196, 198
 Class ticks_labels_style, 553
 Class tspan_element, 202, 205, 206

Class value_style, 556
Function operator<<, 209
Implementation and other notes, 577
Struct polygon_element, 180, 182
suffix
 Class decor_printer, 105
 Class svg_1d_plot, 289
 Class svg_2d_plot, 388, 408
 Class svg_boxplot, 501
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, 536
 Class bar_style, 538
 Class box_style, 539
 Class circle_element, 155, 156, 157
 Class clip_path_element, 158, 159
 Class ellipse_element, 160, 161, 162
 Class g_element, 162, 163, 164, 165, 167
 Class histogram_style, 541
 Class line_element, 169, 170, 171
 Class path_element, 172, 173, 174, 175, 177
 Class plot_line_style, 542
 Class plot_point_style, 544
 Class polyline_element, 183, 184, 185
 Class quurve_element, 186, 187, 188
 Class rect_element, 188, 189, 190, 191
 Class svg_1d_plot, 214, 215, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 252, 255, 259, 261, 262, 271, 272, 277, 278, 279, 280, 283, 284, 287, 291, 292, 293, 300, 301, 302, 303, 307
 Class svg_1d_plot_series, 307
 Class svg_2d_plot, 311, 312, 327, 328, 331, 332, 333, 334, 335, 336, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 352, 355, 359, 361, 362, 371, 372, 378, 379, 382, 383, 386, 389, 390, 392, 401, 408, 409, 411, 412, 413, 414, 418
 Class svg_2d_plot_series, 419
 Class svg_boxplot, 424, 425, 440, 441, 442, 443, 444, 445, 446, 447, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 465, 466, 468, 471, 472, 473, 474, 475, 484, 485, 490, 491, 492, 495, 496, 499, 500, 503, 504, 506, 515, 516, 517
 Class svg_boxplot_series, 521
 Class svg_color, 528, 529, 530
 Class svg_element, 192, 193, 194, 195
 Class svg_style, 547, 548, 549
 Class template unc, 561
 Class text_element, 196, 197, 198, 199
 Class text_element_text, 200
 Class text_parent, 200, 201
 Class text_style, 549, 550, 551, 552
 Class ticks_labels_style, 553, 554
 Class tspan_element, 201, 202, 203, 204, 205, 206
 Class value_style, 555
Colors, 6
Compilers and Examples, 576
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, 532
Function default_font, 557
Function is_blank, 532
Function median, 211
Function operator!=, 208, 533, 558
Function operator<<, 208, 209, 210, 533, 558, 559
Function operator==, 210, 533, 559
Function outFmtFlags, 212
Function quantile, 211
Function rounddown10, 138
Function rounddown2, 138
Function rounddown5, 138
Function roundup10, 139
Function roundup2, 139
Function roundup5, 139
Function scale_axis, 140
Function show_1d_plot_settings, 213
Function show_2d_plot_settings, 214
Function string_svg_length, 560
Function strip_e0s, 311
Function template mnmx, 136
Function template operator<<, 564, 565
Function template range_all, 137
Function template range_mx, 137
Function template scale_axis, 141, 142
Function template show, 143
Function template show_all, 144
Function template uncs_of, 566, 567
Function template unc_of, 565, 566
Function template values_of, 568, 569
Function template value_of, 567, 568
Global aspect_ratio, 146, 557
Global color_array, 531
Global document_ids_, 152
Global fmtFlagWords, 212
Global no_style, 557
Global plusminus, 564
Global reducer, 144, 146
Global sin45, 145, 146
Global spacing_factor, 147
Global text_plusminus, 145, 147
Header < boost/svg_plot/detail/auto_axes.hpp >, 135
Header < boost/svg_plot/detail/axis_plot_frame.hpp >, 144
Header < boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp >, 145
Header < boost/svg_plot/detail/functors.hpp >, 151
Header < boost/svg_plot/detail/svg_boxplot_detail.hpp >, 151
Header < boost/svg_plot/detail/svg_tag.hpp >, 152
Header < boost/svg_plot/quantile.hpp >, 210
Header < boost/svg_plot/show_1d_settings.hpp >, 212
Header < boost/svg_plot/show_2d_settings.hpp >, 213
Header < boost/svg_plot/svg_1d_plot.hpp >, 214
Header < boost/svg_plot/svg_2d_plot.hpp >, 311
Header < boost/svg_plot/svg_boxplot.hpp >, 424
Header < boost/svg_plot/svg_color.hpp >, 528
Header < boost/svg_plot/svg_style.hpp >, 534

Header < boost/svg_plot/uncertain.hpp >, 560
Implementation and other notes, 577
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, 153, 154
Struct c_path, 154, 155
Struct h_path, 168
Struct l_path, 168, 169
Struct m_path, 172
Struct path_point, 177, 178
Struct polygon_element, 179, 180, 181, 182
Struct poly_path_point, 178, 179
Struct q_path, 185, 186
Struct s_path, 191, 192
Struct t_path, 195
Struct v_path, 207
Struct z_path, 207, 208
SVG tutorial, 91
To Do List, 582
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, 530, 531
Using Inkscape, 570

svg

Class circle_element, 156
Class clip_path_element, 158
Class ellipse_element, 160
Class g_element, 163
Class line_element, 170
Class path_element, 173
Class polyline_element, 183
Class curve_element, 186
Class rect_element, 189
Class text_element, 196
Class text_element_text, 200
Class tspan_element, 202
Struct a_path, 154
Struct c_path, 155
Struct h_path, 168
Struct l_path, 169
Struct m_path, 172
Struct polygon_element, 180
Struct q_path, 185
Struct s_path, 192
Struct t_path, 195
Struct v_path, 207
Struct z_path, 208

SVG tutorial

example, 91
SVG, 91

svg_1d_plot

Class svg_1d_plot, 215

svg_1d_plot_series
 Class `svg_1d_plot_series`, 308
svg_2d_plot
 Class `svg_2d_plot`, 312
svg_2d_plot_series
 Class `svg_2d_plot_series`, 419
svg_boxplot
 Class `svg_boxplot`, 425
svg_boxplot_series
 Class `svg_boxplot_series`, 521
svg_color
 Class `svg_color`, 529
 Function `constant_to_rgb`, 532
svg_element
 Class `circle_element`, 156
 Class `clip_path_element`, 158
 Class `ellipse_element`, 160
 Class `g_element`, 163
 Class `line_element`, 170
 Class `path_element`, 173
 Class `polyline_element`, 183
 Class `curve_element`, 186
 Class `rect_element`, 189
 Class `svg_element`, 193
 Class `text_element`, 196
 Class `tspan_element`, 202
 Struct `polygon_element`, 180
svg_style
 Class `svg_style`, 547
svg_styles
 Class `svg_style`, 548
symbol
 Class `plot_point_style`, 545, 546, 547
symbols
 Class `plot_point_style`, 545, 546, 547
 Class `svg_1d_plot_series`, 308, 310
s_path
 Struct `s_path`, 192

T

t
 Class `path_element`, 173, 176
 Function `operator<<`, 209
T
 Class `path_element`, 173, 177
text
 Class `g_element`, 163, 167
 Class `svg_1d_plot`, 241, 244, 303, 306, 307
 Class `svg_2d_plot`, 312, 342, 345, 414, 417, 418
 Class `svg_boxplot`, 454, 457, 517, 520
 Class `text_element`, 196, 198, 199
 Class `tspan_element`, 202, 205, 206
 Header <`boost/svg_plot/svg_style.hpp`>, 534
textstyle
 Class `text_element`, 196, 199
 Class `tspan_element`, 202, 206
text_element

Class text_element, 196
text_element_text
 Class text_element_text, 200
text_length
 Class text_style, 550, 553
 Class tspan_element, 202, 206
text_parent
 Class text_element_text, 200
 Class text_parent, 201
 Class tspan_element, 202
text_style
 Class svg_1d_plot, 261
 Class svg_2d_plot, 312, 361
 Class svg_boxplot, 473
 Class text_style, 550
the
 Class path_element, 173
 Class polyline_element, 183
Tutorial: 1D Autoscale with Multiple Containers, 18
ticks
 Class svg_1d_plot, 279, 280, 281, 283, 284
 Class svg_2d_plot, 312, 378, 379, 380, 382, 383
 Class svg_boxplot, 491, 492, 493, 495, 496
 Class ticks_labels_style, 553
ticks_labels_style
 Class ticks_labels_style, 553
tiny
 Header < boost/svg_plot/detail/fp_compare.hpp >, 147
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_1d_plot, 215, 232, 235, 240, 241, 242, 244, 245, 250, 251, 256, 257, 258, 259, 260, 261, 301, 302, 303, 305, 307
 Class svg_1d_plot_series, 308
 Class svg_2d_plot, 312, 328, 335, 341, 342, 343, 345, 346, 350, 351, 356, 357, 358, 359, 360, 361, 412, 413, 414, 416, 418
 Class svg_2d_plot_series, 419, 420
 Class svg_boxplot, 425, 442, 445, 446, 453, 454, 455, 457, 458, 464, 465, 469, 470, 471, 472, 473, 474, 485, 517, 520
 Class svg_boxplot_series, 521, 523, 528
 Class svg_style, 547
 Definitions of the Quartiles, 86
 Demonstration of using 1D data that includes information about its Uncertainty, 54
Fonts, 8
 Function default_font, 558
 Global aspect_ratio, 146, 557
 Header < boost/svg_plot/detail/axis_plot_frame.hpp >, 144
 Header < boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp >, 145
 Header < boost/svg_plot/svg_style.hpp >, 534
Implementation and other notes, 577
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

titles
 Demonstration of using 1D data that includes information about its Uncertainty, 54

title_color
 Class `svg_1d_plot`, 215, 257
 Class `svg_2d_plot`, 312, 357
 Class `svg_boxplot`, 425, 469, 470

title_font_alignment
 Class `svg_1d_plot`, 215, 257
 Class `svg_2d_plot`, 312, 357
 Class `svg_boxplot`, 425, 470

title_font_decoration
 Class `svg_1d_plot`, 215, 257, 258
 Class `svg_2d_plot`, 312, 358
 Class `svg_boxplot`, 425, 470

title_font_family
 Class `svg_1d_plot`, 215, 258
 Class `svg_2d_plot`, 312, 358
 Class `svg_boxplot`, 425, 470, 471

title_font_size
 Class `svg_1d_plot`, 215, 259
 Class `svg_2d_plot`, 312, 359
 Class `svg_boxplot`, 425, 471, 472

title_font_stretch
 Class `svg_1d_plot`, 215, 259
 Class `svg_2d_plot`, 312, 359, 360
 Class `svg_boxplot`, 425, 472

title_font_style
 Class `svg_1d_plot`, 215, 260
 Class `svg_2d_plot`, 312, 360
 Class `svg_boxplot`, 425, 472

title_on
 Class `svg_1d_plot`, 215, 260, 261
 Class `svg_2d_plot`, 312, 360, 361
 Class `svg_boxplot`, 425, 473

title_size
 Class `svg_boxplot`, 425, 473

title_text_length
 Class `svg_1d_plot`, 215, 261
 Class `svg_2d_plot`, 312, 361
 Class `svg_boxplot`, 425, 473, 474

to
 Class `path_element`, 174, 175, 176, 177
 Struct `a_path`, 153
 Struct `c_path`, 154
 Struct `q_path`, 185
 Struct `s_path`, 191
 Struct `t_path`, 195

To Do List
 container, 582
 data, 582
 example, 582
 Inkscape, 582
 label, 582
 scaling, 582
 series, 582
 sin, 582
 SVG, 582

transform_point
 Class svg_1d_plot, 215, 307
 Class svg_2d_plot, 312, 418
 Class svg_boxplot, 425, 520

transform_x
 Class svg_1d_plot, 215, 307
 Class svg_2d_plot, 312, 418
 Class svg_boxplot, 425, 474

transform_y
 Class svg_1d_plot, 215, 307
 Class svg_2d_plot, 312, 418
 Class svg_boxplot, 425, 474

triangle
 Class g_element, 163, 167

ts
 Function operator<<, 559

tspan
 Class text_element, 196, 199

tspan_element
 Class tspan_element, 202

Tutorial: 1D Autoscale with Multiple Containers
 color, 18
 container, 18
 data, 18
 example, 18
 maximum, 18
 minimum, 18
 series, 18
 SVG, 18
 the, 18
 way, 18
 x_autoscale, 18
 x_range, 18

Tutorial: 1D Gridlines & Axes - more examples
 axis, 22
 background, 22
 color, 22
 container, 22
 data, 22
 example, 22
 font, 22
 grid, 22
 intersect, 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
X, 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

type

Class `svg_1d_plot`, 251
Class `svg_boxplot`, 465
Function template `isfinite`, 136

Type `svg_color_constant`

background, 530
 color, 530, 531
 example, 530
 legend, 530
 minimum, 531
 SVG, 530, 531
types
 Class Meas, 95, 99
 Class template unc, 118, 121, 123, 126, 562, 563
t_or_b
 Header < boost/svg_plot/show_1d_settings.hpp >, 212
t_path
 Struct t_path, 195

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 Meas, 95
 Class template unc, 118, 123, 562
uncertainties
 Class template unc, 562
 Function template uncs_of, 129, 566, 567
 Header < boost/svg_plot/uncertain.hpp >, 560
uncertainty
 2-D Data Values Examples, 67
 Class Meas, 94, 98, 99
 Class plot_point_style, 545
 Class setUncSigDigits, 115
 Class svg_1d_plot, 215, 304
 Class svg_1d_plot_series, 308
 Class svg_2d_plot, 312, 407, 415
 Class svg_2d_plot_series, 420
 Class svg_boxplot, 425, 518
 Class svg_boxplot_series, 523
 Class template unc, 118, 119, 121, 122, 123, 124, 126, 127, 561, 562, 563, 564
 Class value_style, 555, 556, 557
 Compilers and Examples, 576
 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, 140
 Function template operator<<, 128, 565
 Function template scale_axis, 141, 142
 Function template uncs_of, 129, 130, 566, 567
 Function template unc_of, 129, 565, 566
 Function template values_of, 568, 569
 Function template value_of, 130, 567, 568
 Function uFlags, 128, 129
 Global plusminus, 564
 Global reducer, 144, 146
 Header < boost/quan/meas.hpp >, 94
 Header < boost/quan/meas2.hpp >, 100
 Header < boost/quan/unc.hpp >, 107
 Header < boost/quan/uncdata.hpp >, 131
 Header < boost/quan/uncs.hpp >, 132

Header < boost/quan/unc_init.hpp >, 131
 Header < boost/svg_plot/detail/axis_plot_frame.hpp >, 144
 Header < boost/svg_plot/detail/axis_plot_frame_14Aug18_1644_dud.hpp >, 145
 Header < boost/svg_plot/detail/functors.hpp >, 151
 Header < boost/svg_plot/uncertain.hpp >, 560
 History, 576
 Preface, 3
uncorr
 Header < boost/svg_plot/uncertain.hpp >, 560
Uncorrelated
 Demonstration of using 1D data that includes information about its Uncertainty, 54
 Demonstration of using 2D data that includes information about its uncertainty, 76
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
 Header < boost/quan/meas.hpp >, 94
 Header < boost/quan/unc.hpp >, 107
 Header < boost/svg_plot/uncertain.hpp >, 560
unc_of
 Function template unc_of, 129, 565, 566
 Header < boost/quan/unc.hpp >, 107
 Header < boost/svg_plot/uncertain.hpp >, 560
Unicode
 1-D Data Values Examples, 43
 Class plot_point_style, 545
 Class svg_1d_plot, 236, 256, 257, 258, 261, 268, 271, 272, 287, 291, 304, 305
 Class svg_2d_plot, 312, 336, 356, 357, 358, 361, 368, 371, 372, 386, 389, 390, 400, 401, 407, 409, 411, 415, 416
 Class svg_boxplot, 447, 448, 469, 470, 471, 473, 481, 484, 499, 503, 504, 518, 519
 Class template unc, 564
 Class text_element, 196
 Class text_style, 550, 551
 Demonstration of using 2D data that includes information about its uncertainty, 76
Fonts, 8
 Function default_font, 558
 Function template operator<<, 565
 Header < boost/svg_plot/svg_style.hpp >, 534
 Implementation and other notes, 577
 Tutorial: 1D Gridlines & Axes - more examples, 22
 Tutorial: 2D Special Features, 62
units
 Class svg_1d_plot, 273, 274, 306
 Class svg_2d_plot, 373, 374, 417
 Class svg_boxplot, 486
 Class ticks_labels_style, 554
update_image
 Class svg_1d_plot, 215, 262
 Class svg_2d_plot, 312, 362
 Class svg_boxplot, 425, 474
use_down_ticks
 Class ticks_labels_style, 553, 554, 555
use_up_ticks
 Class ticks_labels_style, 553, 555
Using Inkscape
 background, 570
 color, 570
 container, 570
 document, 570
 example, 570

file, 570
font, 570
Inkscape, 570
minimum, 570
scaling, 570
size, 570
SVG, 570
values, 570

V

v Class path_element, 173, 177
V Class path_element, 173, 177
value
 2-D Data Values Examples, 67
 Class axis_line_style, 536
 Class Meas, 95, 99
 Class plot_point_style, 545
 Class svg_boxplot, 450, 462, 464
 Class template unc, 118, 122, 123, 127, 562, 563
 Class ticks_labels_style, 553
 Tutorial: 2D Special Features, 62
values
 1-D Data Values Examples, 43
 2-D Data Values Examples, 67
 Class svg_1d_plot_series, 307
 Class svg_2d_plot, 409, 410
 Class svg_2d_plot_series, 419
 Class svg_boxplot, 511
 Function scale_axis, 140
 Function template mnnmx, 137
 Header < boost/svg_plot/detail/auto_axes.hpp >, 135
 Using Inkscape, 570
values_count
 Class svg_2d_plot_series, 419, 424
values_of
 Function template values_of, 131, 568, 569
 Header < boost/quan/unc.hpp >, 107
 Header < boost/svg_plot/uncertain.hpp >, 560
value_of
 Function template value_of, 130, 567, 568
 Header < boost/quan/unc.hpp >, 107
 Header < boost/svg_plot/uncertain.hpp >, 560
value_style
 Class value_style, 556
variables
 Function constant_to_rgb, 532
vector_iterator
 1-D Auto scaling Examples, 31
 1-D Axis Scaling, 26
vertical
 Class path_element, 177
v_path
 Struct v_path, 207

W

wanted

Class `svg_1d_plot`, 252, 253
Class `svg_2d_plot`, 353
Class `svg_boxplot`, 466

way

Tutorial: 1D Autoscale with Multiple Containers, 18

weight

Class `svg_2d_plot`, 403

whisker_length

Class `svg_boxplot`, 425, 474
Class `svg_boxplot_series`, 521, 528

width

Class `axis_line_style`, 536, 537, 538
Class `bar_style`, 538, 539
Class `box_style`, 540, 541
Class `plot_line_style`, 543, 544
Class `rect_element`, 189, 191
Class `svg_1d_plot`, 238, 239, 274, 287
Class `svg_2d_plot`, 339, 374, 386
Class `svg_boxplot`, 451, 468, 486, 487
Class `svg_style`, 549

Demonstration of using 1D data that includes information about its Uncertainty, 54

width_on

Class `svg_style`, 547, 549

window

Class `svg_1d_plot`, 215, 289, 290
Class `svg_2d_plot`, 312, 340, 409
Class `svg_boxplot`, 425, 502

with

Class `template unc`, 564
Function template `operator<<`, 128, 565

write

Class `circle_element`, 156, 157
Class `clip_path_element`, 158, 159
Class `ellipse_element`, 160, 162
Class `g_element`, 163, 167
Class `line_element`, 170, 171
Class `path_element`, 173, 177
Class `polyline_element`, 183, 185
Class `curve_element`, 186, 188
Class `rect_element`, 189, 191
Class `svg_1d_plot`, 215, 262
Class `svg_2d_plot`, 312, 362
Class `svg_boxplot`, 425, 475
Class `svg_color`, 529, 530
Class `svg_element`, 193, 195
Class `svg_style`, 547, 549
Class `text_element`, 196, 199
Class `text_element_text`, 200
Class `text_parent`, 201
Class `tspan_element`, 202, 206
Struct `a_path`, 154
Struct `c_path`, 155
Struct `h_path`, 168
Struct `l_path`, 169
Struct `m_path`, 172

Struct path_point, 178
 Struct polygon_element, 180, 182
 Struct poly_path_point, 179
 Struct q_path, 185, 186
 Struct s_path, 192
 Struct t_path, 195
 Struct v_path, 207
 Struct z_path, 208
write_attributes
 Class circle_element, 156, 158
 Class clip_path_element, 158, 160
 Class ellipse_element, 160, 162
 Class g_element, 163, 167
 Class line_element, 170, 172
 Class path_element, 173, 177
 Class polyline_element, 183, 185
 Class quurve_element, 186, 188
 Class rect_element, 189, 191
 Class svg_element, 193, 194
 Class text_element, 196, 200
 Class tspan_element, 202, 207
 Struct polygon_element, 180, 182

X**X**

Tutorial: 2D Special Features, 62

x

Class rect_element, 189, 191
 Class text_element, 196, 199
 Class tspan_element, 202, 206

xy_autoscale

Class svg_2d_plot, 312, 397

xy_values_on

Class svg_2d_plot, 312, 397

x_adlimits_color

Class svg_1d_plot, 215, 262, 263
 Class svg_2d_plot, 312, 362
 Class svg_boxplot, 425, 475

x_adlimits_on

Class svg_1d_plot, 215, 263
 Class svg_2d_plot, 312, 362, 363
 Class svg_boxplot, 425, 475, 476

x_autoscale

1-D Auto scaling Examples, 31
 Class svg_1d_plot, 215, 264, 265, 280, 281, 287, 288, 293, 298
 Class svg_2d_plot, 312, 364, 365, 379, 380, 387, 391, 396
 Class svg_boxplot, 425, 476, 477, 492, 493, 500, 505, 510

Tutorial: 1D Autoscale with Multiple Containers, 18

x_auto_max_value

Class svg_1d_plot, 215, 263
 Class svg_2d_plot, 312, 363
 Class svg_boxplot, 425, 476

x_auto_min_value

Class svg_1d_plot, 215, 263
 Class svg_2d_plot, 312, 363
 Class svg_boxplot, 425, 476

x_auto_ticks

Class `svg_1d_plot`, 215, 264
Class `svg_2d_plot`, 312, 363, 364
Class `svg_boxplot`, 425, 476
`x_auto_tick_interval`
 Class `svg_1d_plot`, 215, 263, 264
 Class `svg_2d_plot`, 312, 363
 Class `svg_boxplot`, 425, 476
`x_axis`
 Class `svg_2d_plot`, 312, 365
`x_axis_color`
 1-D Auto scaling Examples, 31
 Class `svg_1d_plot`, 215, 265
 Class `svg_2d_plot`, 312, 365
 Class `svg_boxplot`, 425, 478
`x_axis_label_color`
 Class `svg_1d_plot`, 215, 266
 Class `svg_2d_plot`, 312, 365, 366
 Class `svg_boxplot`, 425, 478
`x_axis_on`
 Class `svg_1d_plot`, 215, 266
 Class `svg_2d_plot`, 312, 366
 Class `svg_boxplot`, 425, 479
`x_axis_position`
 Class `svg_1d_plot`, 215, 266
 Class `svg_2d_plot`, 312, 366
 Class `svg_boxplot`, 425, 479
`x_axis_vertical`
 Class `svg_1d_plot`, 215, 267
 Class `svg_2d_plot`, 312, 366, 367
 Class `svg_boxplot`, 425, 479
`x_axis_width`
 Class `svg_1d_plot`, 215, 267
 Class `svg_2d_plot`, 312, 367
 Class `svg_boxplot`, 425, 480
`x_datetime_color`
 Class `svg_1d_plot`, 215, 267, 268
 Class `svg_2d_plot`, 312, 367, 368
 Class `svg_boxplot`, 425, 480
`x_datetime_on`
 Class `svg_1d_plot`, 215, 268
 Class `svg_2d_plot`, 312, 368
 Class `svg_boxplot`, 425, 480, 481
`x_decor`
 Class `svg_1d_plot`, 215, 268
 Class `svg_2d_plot`, 312, 368
 Class `svg_boxplot`, 425, 481
`x_df_color`
 Class `svg_1d_plot`, 215, 269
 Class `svg_2d_plot`, 312, 369
 Class `svg_boxplot`, 425, 481, 482
`x_df_on`
 Class `svg_1d_plot`, 215, 269, 270
 Class `svg_2d_plot`, 312, 369, 370
 Class `svg_boxplot`, 425, 482
`x_id_color`
 Class `svg_1d_plot`, 215, 270
 Class `svg_2d_plot`, 312, 370
 Class `svg_boxplot`, 425, 482, 483

x_id_on
Class svg_1d_plot, 215, 270
Class svg_2d_plot, 312, 370
Class svg_boxplot, 425, 483

x_label
Class svg_1d_plot, 215, 271
Class svg_2d_plot, 312, 371
Class svg_boxplot, 425, 483

x_labels_strip_e0s
Class svg_1d_plot, 215, 274
Class svg_2d_plot, 312, 374
Class svg_boxplot, 425, 487

x_label_color
Class svg_1d_plot, 215, 271
Class svg_2d_plot, 312, 371
Class svg_boxplot, 425, 484

x_label_font_size
Class svg_1d_plot, 215, 272
Class svg_2d_plot, 312, 372
Class svg_boxplot, 425, 484, 485

x_label_on
Class svg_1d_plot, 215, 271, 272, 273
Class svg_2d_plot, 312, 371, 372, 373
Class svg_boxplot, 425, 485

x_label_size
Class svg_boxplot, 425, 485

x_label_text
Class svg_boxplot, 425, 485

x_label_units
Class svg_1d_plot, 215, 273
Class svg_2d_plot, 312, 373
Class svg_boxplot, 425, 485

x_major_grid_color
Class svg_1d_plot, 215, 275
Class svg_2d_plot, 312, 374, 375
Class svg_boxplot, 425, 487

x_major_grid_on
Class svg_1d_plot, 215, 275
Class svg_2d_plot, 312, 375
Class svg_boxplot, 425, 487, 488

x_major_grid_width
Class svg_1d_plot, 215, 276
Class svg_2d_plot, 312, 375, 376
Class svg_boxplot, 425, 488

x_major_interval
Class svg_1d_plot, 215, 276
Class svg_2d_plot, 312, 376
Class svg_boxplot, 425, 488, 489

x_major_labels
Class svg_boxplot, 425, 490

x_major_labels_side
Class svg_1d_plot, 215, 277, 278
Class svg_2d_plot, 312, 377
Class svg_boxplot, 425, 490

x_major_label_rotation
Class svg_1d_plot, 215, 276, 277
Class svg_2d_plot, 312, 376, 377
Class svg_boxplot, 425, 489

x_major_tick_color
Class svg_1d_plot, 215, 278, 279
Class svg_2d_plot, 312, 378
Class svg_boxplot, 425, 491

x_major_tick_length
Class svg_1d_plot, 215, 279
Class svg_2d_plot, 312, 378
Class svg_boxplot, 425, 491, 492

x_major_tick_width
Class svg_1d_plot, 215, 279, 280
Class svg_2d_plot, 312, 378, 379
Class svg_boxplot, 425, 492

x_max
Class svg_1d_plot, 215, 280
Class svg_2d_plot, 312, 379
Class svg_boxplot, 425, 492

x_min
Class svg_1d_plot, 215, 280
Class svg_2d_plot, 312, 379, 380
Class svg_boxplot, 425, 492, 493

x_minor_grid_color
Class svg_1d_plot, 215, 281
Class svg_2d_plot, 312, 380
Class svg_boxplot, 425, 493, 494

x_minor_grid_on
Class svg_1d_plot, 215, 281, 282
Class svg_2d_plot, 312, 381
Class svg_boxplot, 425, 494

x_minor_grid_width
Class svg_1d_plot, 215, 282
Class svg_2d_plot, 312, 381
Class svg_boxplot, 425, 494

x_minor_tick_color
Class svg_1d_plot, 215, 283
Class svg_2d_plot, 312, 382
Class svg_boxplot, 425, 495

x_num_minor_ticks
Class svg_1d_plot, 215, 284
Class svg_2d_plot, 312, 383
Class svg_boxplot, 425, 496
Implementation and other notes, 577

x_order_color
Class svg_1d_plot, 215, 284, 285
Class svg_2d_plot, 312, 383, 384
Class svg_boxplot, 425, 497

x_order_on
Class svg_1d_plot, 215, 285
Class svg_2d_plot, 312, 384
Class svg_boxplot, 425, 497

x_plusminus_color
Class svg_1d_plot, 215, 285, 286
Class svg_2d_plot, 312, 384, 385
Class svg_boxplot, 425, 497, 498

x_plusminus_on
Class svg_1d_plot, 215, 286
Class svg_2d_plot, 312, 385
Class svg_boxplot, 425, 498

x_prefix

Class `svg_1d_plot`, 215, 286
Class `svg_2d_plot`, 312, 385
Class `svg_boxplot`, 425, 498
x_range
 1-D Auto scaling Examples, 31
 1-D Autoscaling Various Containers Examples, 40
 1-D Axis Scaling, 26
 Class `svg_1d_plot`, 215, 286, 287
 Class `svg_2d_plot`, 312, 385, 386
 Class `svg_boxplot`, 425, 499
 Tutorial: 1D Autoscale with Multiple Containers, 18
x_size
 1-D Data Values Examples, 43
 Class `svg_1d_plot`, 215, 238, 239, 287
 Class `svg_2d_plot`, 312, 339, 386
 Class `svg_boxplot`, 425, 451, 499, 500
x_steps
 Class `svg_1d_plot`, 215, 287, 288
 Class `svg_2d_plot`, 312, 387, 388
 Class `svg_boxplot`, 425, 500, 501
x_suffix
 Class `svg_1d_plot`, 215, 289
 Class `svg_2d_plot`, 312, 388
 Class `svg_boxplot`, 425, 501
x_ticks
 Class `svg_2d_plot`, 312, 388
x_ticks_down_on
 Class `svg_1d_plot`, 215, 289
 Class `svg_2d_plot`, 312, 388
 Class `svg_boxplot`, 425, 501, 502
x_ticks_on_window_or_axis
 Class `svg_1d_plot`, 215, 289, 290
 Class `svg_2d_plot`, 312, 388
 Class `svg_boxplot`, 425, 502
x_ticks_up_on
 Class `svg_1d_plot`, 215, 290
 Class `svg_2d_plot`, 312, 389
 Class `svg_boxplot`, 425, 503
x_ticks_values_color
 Class `svg_1d_plot`, 215, 290, 291
 Class `svg_2d_plot`, 312, 389
 Class `svg_boxplot`, 425, 503
x_ticks_values_font_family
 Class `svg_1d_plot`, 215, 291
 Class `svg_2d_plot`, 312, 389, 390
 Class `svg_boxplot`, 425, 503, 504
x_ticks_values_font_size
 Class `svg_1d_plot`, 215, 291, 292
 Class `svg_2d_plot`, 312, 390
 Class `svg_boxplot`, 425, 504
x_ticks_values_ioflags
 Class `svg_1d_plot`, 215, 292
 Class `svg_2d_plot`, 312, 390, 391
 Class `svg_boxplot`, 425, 504, 505
x_ticks_values_precision
 Class `svg_1d_plot`, 215, 292, 293
 Class `svg_2d_plot`, 312, 391
 Class `svg_boxplot`, 425, 505

x_tick_color
 Class svg_boxplot, 425, 501
x_tick_length
 Class svg_boxplot, 425, 501
x_tick_width
 Class svg_boxplot, 425, 501
x_tight
 Class svg_1d_plot, 215, 293
 Class svg_2d_plot, 312, 391, 392
 Class svg_boxplot, 425, 505, 506
x_values_color
 Class svg_1d_plot, 215, 295
 Class svg_2d_plot, 312, 393
 Class svg_boxplot, 425, 507
x_values_font_family
 Class svg_1d_plot, 215, 295
 Class svg_2d_plot, 312, 394
 Class svg_boxplot, 425, 508
x_values_font_size
 Class svg_1d_plot, 215, 295, 296
 Class svg_2d_plot, 312, 394
 Class svg_boxplot, 425, 508
x_values_ioflags
 Class svg_1d_plot, 215, 296
 Class svg_2d_plot, 312, 394, 395
 Class svg_boxplot, 425, 508, 509
x_values_on
 Class svg_1d_plot, 215, 296, 297
 Class svg_2d_plot, 312, 395
 Class svg_boxplot, 425, 509
x_values_precision
 Class svg_1d_plot, 215, 297
 Class svg_2d_plot, 312, 395, 396
 Class svg_boxplot, 425, 509, 510
x_values_rotation
 Class svg_1d_plot, 215, 297
 Class svg_2d_plot, 312, 396
 Class svg_boxplot, 425, 510
x_value_ioflags
 Class svg_1d_plot, 215, 293, 294
 Class svg_2d_plot, 312, 392, 393
 Class svg_boxplot, 425, 506, 507
x_value_precision
 Class svg_1d_plot, 215, 294
 Class svg_2d_plot, 312, 393
 Class svg_boxplot, 425, 507
x_with_zero
 Class svg_1d_plot, 215, 298
 Class svg_2d_plot, 312, 396
 Class svg_boxplot, 425, 510

Y

y
 Class rect_element, 189, 191
 Class text_element, 196, 199
 Class tspan_element, 202, 206, 207
y_addlimits_color

Class `svg_2d_plot`, 312, 397
`y_addlimits_on`
 Class `svg_2d_plot`, 312, 397
`y_autoscale`
 Class `svg_2d_plot`, 312, 397, 398
 Class `svg_boxplot`, 425, 511
`y_axis`
 Class `svg_2d_plot`, 312, 398
`y_axis_color`
 Class `svg_1d_plot`, 215, 298
 Class `svg_2d_plot`, 312, 398
 Class `svg_boxplot`, 425, 511, 512
`y_axis_label_color`
 Class `svg_2d_plot`, 312, 398, 399
`y_axis_on`
 Class `svg_1d_plot`, 215, 298, 299
 Class `svg_2d_plot`, 312, 399
 Class `svg_boxplot`, 425, 512
`y_axis_position`
 Class `svg_2d_plot`, 312, 399
 Class `svg_boxplot`, 425, 512
`y_axis_value_color`
 Class `svg_2d_plot`, 312, 399
`y_axis_width`
 Class `svg_2d_plot`, 312, 399, 400
`y_decor`
 Class `svg_2d_plot`, 312, 400
`y_df_color`
 Class `svg_2d_plot`, 312, 400
`y_df_on`
 Class `svg_2d_plot`, 312, 400
`y_label`
 Class `svg_1d_plot`, 215, 299
 Class `svg_2d_plot`, 312, 400, 401
 Class `svg_boxplot`, 425, 512
`y_labels_strip_e0s`
 Class `svg_1d_plot`, 215, 300
 Class `svg_2d_plot`, 312, 403
 Class `svg_boxplot`, 425, 514
`y_label_axis`
 Class `svg_2d_plot`, 312, 401
`y_label_color`
 Class `svg_1d_plot`, 215, 299, 300
 Class `svg_2d_plot`, 312, 401
 Class `svg_boxplot`, 425, 513
`y_label_font_family`
 Class `svg_2d_plot`, 312, 401, 402
`y_label_font_size`
 Class `svg_2d_plot`, 312, 402
 Class `svg_boxplot`, 425, 513
`y_label_on`
 Class `svg_1d_plot`, 299
 Class `svg_2d_plot`, 312, 400, 401, 402
 Class `svg_boxplot`, 425, 512, 513
`y_label_text`
 Class `svg_boxplot`, 425, 513
`y_label_units`
 Class `svg_1d_plot`, 215, 300

Class `svg_2d_plot`, 312, 402
Class `svg_boxplot`, 425, 513
`y_label_units_on`
 Class `svg_2d_plot`, 312, 402, 403
`y_label_weight`
 Class `svg_2d_plot`, 312, 403
`y_label_width`
 Class `svg_2d_plot`, 312, 403
`y_major_grid_color`
 Class `svg_2d_plot`, 312, 403
`y_major_grid_on`
 Class `svg_2d_plot`, 312, 403, 404
`y_major_grid_width`
 Class `svg_2d_plot`, 312, 404
`y_major_interval`
 Class `svg_2d_plot`, 312, 404
 Class `svg_boxplot`, 425, 514
`y_major_labels_side`
 Class `svg_2d_plot`, 312, 404
`y_major_label_rotation`
 Class `svg_2d_plot`, 312, 404
`y_major_tick_color`
 Class `svg_2d_plot`, 312, 405
 Class `svg_boxplot`, 425, 514
`y_major_tick_length`
 Class `svg_2d_plot`, 312, 405
 Class `svg_boxplot`, 425, 514
`y_major_tick_width`
 Class `svg_2d_plot`, 312, 405
 Class `svg_boxplot`, 425, 514
`y_max`
 Class `svg_2d_plot`, 312, 405
`y_min`
 Class `svg_2d_plot`, 312, 405
`y_minor_grid_color`
 Class `svg_2d_plot`, 312, 405
`y_minor_grid_on`
 Class `svg_2d_plot`, 312, 405, 406
`y_minor_grid_width`
 Class `svg_2d_plot`, 312, 406
`y_minor_interval`
 Class `svg_1d_plot`, 215, 300
 Class `svg_2d_plot`, 312, 406
 Class `svg_boxplot`, 425, 514
`y_minor_tick_color`
 Class `svg_2d_plot`, 312, 406
 Class `svg_boxplot`, 425, 514
`y_minor_tick_length`
 Class `svg_2d_plot`, 312, 406
 Class `svg_boxplot`, 425, 515
`y_minor_tick_width`
 Class `svg_2d_plot`, 312, 406, 407
 Class `svg_boxplot`, 425, 515
`y_num_minor_ticks`
 Class `svg_2d_plot`, 312, 407
 Class `svg_boxplot`, 425, 515
`y_plusminus_color`
 Class `svg_2d_plot`, 312, 407

y_plusminus_on
 Class svg_2d_plot, 312, 407

y_prefix
 Class svg_2d_plot, 312, 407

y_range
 Class svg_2d_plot, 312, 407
 Class svg_boxplot, 425, 515

y_separator
 Class svg_2d_plot, 312, 407

y_size
 1-D Data Values Examples, 43
 Class svg_1d_plot, 215, 239, 300, 301
 Class svg_2d_plot, 312, 339, 340, 408
 Class svg_boxplot, 425, 451, 452, 515

y_suffix
 Class svg_2d_plot, 312, 408

y_ticks
 Class svg_2d_plot, 312, 408

y_ticks_left_on
 Class svg_2d_plot, 312, 408

y_ticks_on_window_or_axis
 Class svg_2d_plot, 312, 408, 409

y_ticks_right_on
 Class svg_2d_plot, 312, 409

y_ticks_values_color
 Class svg_2d_plot, 312, 409

y_ticks_values_font_family
 Class svg_2d_plot, 312, 409

y_ticks_values_font_size
 Class svg_2d_plot, 312, 409

y_ticks_values_ioflags
 Class svg_2d_plot, 312, 410

y_ticks_values_precision
 Class svg_2d_plot, 312, 410

y_values_color
 Class svg_2d_plot, 312, 410, 411

y_values_font_family
 Class svg_2d_plot, 312, 411

y_values_font_size
 Class svg_2d_plot, 312, 411

y_values_ioflags
 Class svg_2d_plot, 312, 411

y_values_on
 Class svg_2d_plot, 312, 411

y_values_precision
 Class svg_2d_plot, 312, 411, 412

y_values_rotation
 Class svg_2d_plot, 312, 412

y_value_ioflags
 Class svg_2d_plot, 312, 410

y_value_precision
 Class svg_2d_plot, 312, 410

Z

z
 Class path_element, 173, 177

Z

Class path_element, 173, 177
zero
Class svg_1d_plot, 298
Class svg_2d_plot, 396, 397
Class svg_boxplot, 510, 511
z_path
Struct z_path, 208



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) hftp (dot) u-net (dot) com or Jake Voytko at jake-voytko (at) gmail (dot) com