

Data Visualization with Chart::Clicker

Version 1.0

Cory Watson

Data Visualization with Chart::Clicker

Copyright © 2009 by Cold Hard Code, LLC.

Version 1.0

All rights reserved. This work is released under the Creative Commons Attribute-Noncommercial-No Derivative Works 3.0 United States License.¹ This means you must attribute any usage of this work to the author, that you may not use it for commercial purposes and that you may not alter, transform or build upon this work. It's mine. I wrote it! Corrections, suggestions, well-wishes and cheques are welcome.

¹ <http://creativecommons.org/licenses/by-nc-nd/3.0/us/>

To Jennifer, who displayed deitic patience throughout numerous explanations, debugging sessions and fitful attempts at sleep whilst I was designing the software described in this book. This is but one of the countless reasons I've chosen to spend my life with you.

Table of Contents

Preface

.....

7

Contents

.....

7

How to read this book

.....

8

Cory's Introduction

.....

9

Acknowledgements

.....

10

Chapter X:

.....

12

Examples

.....

12

Colophon

.....

22

Preface

There are a handful of reasons I decided to write a book about Clicker. The first is because I thought it would be fun. The second is that I'm studying to be a designer and the idea of laying out a book and applying my lessons is extremely enticing. The final reason is to communicate the intricacies of Clicker to an audience who is interested in learning more than the online documentation can easily convey.

Contents

This book is divided into four sections. The first section will cover simple usage. We will start with common scenarios so that you can immediately see some benefit from your time with this text.

The second section will delve into more advanced topics that will be useful for users who have more demanding needs of Clicker.

The third section will cover some of the fundamental building blocks, filling in the backstory

necessary to understand what Clicker's strengths and limitations are.

The fourth section will build on the lessons of the previous chapter. Here we will discuss how to extend Clicker to meet needs that may arise as you use it. We will cover everything from gratuitous graphical embellishments to reworkings of fundamental concepts.

How to read this book

Chart::Clicker isn't a canned set of templates that you plug data into. It's a toolkit for visualizing data. The "canned" pieces that you find when reading the code's documentation are provided for drive-through users who need to accomplish a simple goal. With a bit of effort we can provide these people with an elegant result using sane defaults. If you just want to throw together a line chart then reading this book might not be for you.

If you are interested in visualizing your data in interesting ways and standard tools fall short of your expectations, then this book (and Clicker) is for you.

The sections are written to flow cover to cover. Unless you've been using Clicker and are comfortable with it's various built-in capabilities, I highly recommend at least skimming the first chapter.

Examples will be common and diagrams will be used when visualizing a concept may be helpful. Code samples will be presented in a monospaced font for easy identification.

Important or peculiar notes will be separated in a special enclosure like this one so they are easy to find.

Cory's Introduction

I've been writing software for over 10 years and an inexcusable amount of that time has been dedicated to charting and data visualization.

My first need was server monitoring. I setup some nightly cron jobs that generated a new chart every night. After some time I made them transparent GIFs and positioned them over top each other in a browser, allowing the user to choose how many days of history to show.

This obsession continued for years. I tried libraries in various languages and never found exactly what I was looking for. I eventually became foolish enough to write my own. Clicker is my third attempt. I like to think it's the best. I also hope it will be the last. Maybe I'll take up book writing instead so I never have to spend the sleepless nights that accompanied embarking on this insane task.

Acknowledgements

Many great ideas were absorbed along the way and I'd be remiss to overlook the likes of Chart¹, JFreeChart² the Java 2D API³, Processing⁴ and the countless charting libraries out there.

As for websites there are more than I can mention. Countless blogs and reference sites have provided me inspiration, information and confusion.

¹ <http://search.cpan.org/perldoc?Chart>

² <http://www.jfree.org/jfreechart/>

³ <http://java.sun.com/products/java-media/2D/index.jsp>

⁴ <http://processing.org/>

The works of Edward Tufte, the “da Vinci of Data” have shaped many of my opinions and informed much of my work. His classic *The Visual Display of Quantitative Information* completely reshaped my opinions with regards to data visualization.

Chapter X:

Examples

One of the more surprising developments of creating a charting library – a completely visual endeavour – is the importance of textual examples. This should come as no surprise to me. I’ve been developing software for most of my life and one of the most irritating shortcomings I’ve encountered when using someone else’s code is the lack of simple examples.

This chapter is an attempt to free Clicker’s users from this horrid fate. We will begin with some simple, scalable examples that prepare you for future work, then progress into more specialized work.

A Note On Examples

Clicker’s namespace is somewhat *verbose*. As such, I’ve made liberal use of the `aliased` module from the CPAN to shorten examples without losing any

¹ <http://search.cpan.org/perldoc?aliased>

useful information. It is certainly not necessary for use to use aliased if you aren't inclined to. Simply look at the top of the example to map the class name to it's fully qualified name.

Simple Line Chart

```
use Chart::Clicker;
use aliased 'Chart::Clicker::Data::DataSet';
use aliased 'Chart::Clicker::Data::Marker';
use aliased 'Chart::Clicker::Data::Series';

my $cc = Chart::Clicker->new(
    width => 500, height => 300
);

my $series1 = Series->new(
    keys    => [qw(1 2 3 4 5 6 7 8 9 10 11 12)],
    values => [qw(8 5 4 8 5 4 3 9 2 8 9 8)]
);

my $ds = DataSet->new(series => [ $series1 ]);
```

```
$cc->add_to_datasets($ds);
```

```
$cc->draw;
```

```
$cc->write('chart.png');
```

This is the proverbial seed of all other examples. You will see it's touch on most every other example you encounter. Many future examples will even refer back to this one. Let's break it down.

```
use Chart::Clicker;
```

```
use aliased 'Chart::Clicker::Data::DataSet';
```

```
use aliased 'Chart::Clicker::Data::Marker';
```

```
use aliased 'Chart::Clicker::Data::Series';
```

This is merely housekeeping. We've got to import the packages we'll be using. As mentioned earlier, we are using aliased to keep things succinct in examples.

```
my $cc = Chart::Clicker->new(  
    width => 500, height => 300  
);
```

This instantiates a new `Chart::Clicker` object with a width of 500 and a height of 300. There are lots of options that can be provided here. These are summarized in the POD for `Chart::Clicker`.

```
my $series1 = Series->new(  
    keys    => [qw(1 2 3 4 5 6 7 8 9 10 11 12)],  
    values => [qw(8 5 4 8 5 4 3 9 2 8 9 8)]  
);
```

A bit longer, this block creates a new `Series` with its keys and values set to a list of numbers. This is an example wherein you already know all the values you will be charting and can specify them in one go. Refer to the example on building data dynamically if this isn't what you had in mind.

```
my $ds = DataSet->new(series => [ $series1 ]);
```

After creating the series, we add it to a `DataSet`. `DataSets` are, in this example, merely a formality. They do, however, provide a useful abstraction internally.

```
$cc->add_to_datasets($ds);
```

After creating a DataSet to hold our precious Series, we must make our Clicker object aware of it with `add_to_datasets`.

```
$cc->draw;  
$cc->write('chart.png');
```

Finally we ask Clicker to draw the chart and write its contents to a file.

Changing the Renderer (Bar, Area, etc)

Your author is lazy. Isn't there an old saying that you can give a man a chart, but teach him to generate them himself and he'll visualize data for a lifetime? Well, there should be. As such, I'm going to refer back to the first example, *Simple Line Chart*. If you've studied it and follow it completely, changing the renderer is very simple. I'll provide a few opening lines for context.

```
use aliased 'Chart::Clicker::Renderer::Bar';  
# ...
```

```
my $cc = Chart::Clicker->new(  
    width => 500, height => 300  
);  
my $def = $cc->get_context('default');  
$def->renderer->($);  
$def->renderer(Bar->new);
```

Changing a render is accomplished via a Context. If this word is foreign to you then please look back to the chapter on Contexts. It will explain – at length – that every Clicker chart has a “default” context. Contexts allow you to create different groupings of data and then chart them in different ways. In this case we are changing the default context from using a Line renderer to using a Bar. There we are, easy as cake!¹

Pie Renderer

The pie renderer doesn’t behave quite like the others. It’s different because it doesn’t make sense

¹ Please note that your author’s favorite cake is spice cake with cream cheese icing. If you are ever feeling generous, you can make him one. He’ll be ever so appreciative.

use different datasets or series with a pie chart, as it represents pieces of a sum. As such, when you create data for use with the Pie renderer, you must create a series *for each slice of the pie*. The renderer will automatically sum all the datapoints in the series and represent each pie slice as the percentage of the whole dataset.

```
my $series1 = Series->new(  
    keys    => [ 1, 2, 3 ],  
    values => [ 1, 2, 3 ],  
);  
my $series2 = Series->new(  
    keys    => [ 1, 2, 3 ],  
    values => [ 1, 1, 1 ],  
);  
my $series3 = Series->new(  
    keys    => [ 1, 2, 3 ],  
    values => [ 1, 1, 0 ],  
);
```

The above series will combine to form a total dataset with a value of 11 (1 + 2 + 3, 1 + 1 + 1 and 1 + 1 + 0).

Remember, each series represents one slice so the 3 slices will have values of 6, 3 and 2.

```
use aliased 'Chart::Clicker::Renderer::Pie';  
# ...  
my $pie = Pie->new;  
my $defctx = $cc->get_context('default');  
$defctx->renderer($pie);
```

With the data in hand, we can tell the default context to use a Pie renderer in the same fashion that we'd change the renderer for any other chart.

The keys in the series really serve no purpose for the Pie renderer. Honestly, the Pie renderer was added late in the game and doesn't fit as cleanly into Clicker's ideology as I'd like. It will likely be revisited at some point the future.

Disabling the Legend

The legend is added by default to all Clicker charts. Disabling it, however, is quite simple:

```
my $cc = Chart::Clicker->new(  
    width => 500, height => 300  
);
```

```
# Some other code
$cc->legend->(0);
```

All of the components that make up a Clicker chart can be hidden in this way. Be careful what you hide, however, as hiding some components may have unintended results.

Hiding an Axis

In the *Disabling the Legend* it was revealed that nearly any component in a chart may have its visible attribute set to 0. One exception to this is the Axis. Setting it to not display causes problems, as the Axis needs to be laid out in the chart to know how to position data. Hiding an axis requires slightly different magic. The hidden attribute is a Clicker-specific feature added to address this limitation. The underlying `Graphics::Primitive` treats the Axis normally, but Clicker skips drawing it.

```
my $cc = Chart::Clicker->new(
    width => 500, height => 300
);
# Some other code...
```

```
# If you are using multiple contexts, be sure and
# specify the correct one.
my $context = $cc->get_context('default');
# This example uses the range axis, domain works
# just the same
my $range_axis = $context->range_axis;
$range_axis->hidden(1);
```

Colophon

The majority of choices made when laying out this book can be credited to *The Elements of Typographic Style* by Robert Bringhurst (H&M Publishers).

This page's size is 5x8 inches. This resolves to a ratio of 1:1.6. The textblock of the page is 62.5% of 5 inches, or 3.125 inches. The remaining 1.175 inches are split proportionally with 62.5% of the padding on the inside (1.175 inches) and 0.7 inches on the outside.

The sizes of the fonts are 21 (section heading), 16 (heading 1), 13 (heading 2), 10 (body) and 8 (little bits). This is a two-stranded Fibonacci sequence representing the Golden ratio.¹

The body font is Hoefler Text by Hoefler & Frere-Jones. The title font is Optima by Hermann Zapf. The monospace font used for code examples is Inconsolata by Raph Levien.²

¹ http://en.wikipedia.org/wiki/Golden_section

² <http://www.levien.com/type/myfonts/inconsolata.html>