Reference Manual

Contents

Chapter 1

Getting Started

COMPUTER GRAPHICS IS AN exiting field. And when it comes to biological data, graphics is an essential part of it. Whatever may be the form, from a plasmid to gene sequence, for a biologist, data is synonymous with graphics.

When it comes to data source, for a biologist, the source is almost always the Internet. Unfortunately, graphics over the web was traditionally of bitmap raster format (*e.g.*, GIF, JPEG, PNG). These graphics formats are not difficult to create but they have two great big flaws–the lack of scalability (*i.e.*, they lose resolution when scaled to a bigger size), and the text is not text (*i.e.*, the text also becomes a bitmap). They are also very bulky in nature, in spite of the exitance of efficient compression algorithms.

1.1 From raster to vector

There was ofcource an alternative the traditional raster format–vector images. Vector images by definition are created as instruction sets to the display machine. The display machine computes all the points and their attributes on the fly and generates the bitmap to show it to the users.

The advantage of this method of display is that the image immediately becomes resolution independent. All the modern fonts are nothing but vector instructions to the display machine, thus, text is vector display remains text.

For quite sometime there was only one dominant player for transferring vector data over the Web. Created by Macromedia, Flash®, became the standard for vector data distribution over the web. The Flash file format, though open-source is controlled by Macromedia.

The situation changed with the invention of a new file format for vector data called *Scalable Vector Graphics (SVG)* by W3C (`http://www.w3.org/Graphics/SVG/Overview.htm8`).

1.2 What is BioSVG?

BioSVG is a set of perl modules to create SVG for biological data. Idea is to replace all instances of the use of raster format data in graphics for biology. In short, world domination!

BioSVG will consist of all types of graphics object used in biology. Although primarily meant to be used in a CGI environment, BioSVG can be used even in a stand-alone mode.

1.3 Requirements

BioSVG is dependent on Pastel, a framework for creating SVG data. Pastel is called internally from the BioSVG modules. In fact all BioSVG objects are subclasses of Pastel::Graphics class.

The SVG data generated by BioSVG can be viewed by any SVG viewer. The most common viewers are: Adobe SVG viewer (ASV), available from:

`http://www.adobe.com/svg/viewer/install/main.html`.

ASV is just a browser plugin. Once installed, SVG data can be viewed on your browser. ASV is actually installed automatically with the Windows version of Acrobat Reader® version 5.0 onwards.

The second most common viewer is Batik SVG viewer, available from:

`http://xml.apache.org/batik/index.html`.

Batik is written in Java, therefore, can be run in any platform. Batik is a stand-alone SVG viewer.

1.4 Installation

Follow the standard Perl way:

```
make
make test
make  isntall
```

1.5 Code-convention using BioSVG

A very minimal BioSVG file:

```
1 # / usr/bin/perl ;
2 use BioSVG;
3 my $some_object = BioSVG::Some_Object->new(...);
4 $some_object->show(); # to dump the output  to  STDOUT
5 $some_object->get_svg(); # to  get  the  SVG document as  strings
```

Note that there are two ways to get the SVG (line 4 and 5). You can dump it on STDOUT, a method may be more preferable in case of CGI, or you can get the whole string, may be to embed the SVG document into an HTML file.

All constructors should be called with named parameters:

```
my $chromatogram = BioSVG::Chromatogram->create_from_file(-file =>"test.ab1",
                                                          -type=>"ABI",
                                                          -height=>250,
                                                          -hscale=>1 );
```

Note that the constructor, in this case, `create_from_file ()` is called with named parameters. The sequence of parameters could be interchanged.

The method calls does not require this named parameters. But the parameter sequence must be maintained.

Thus, you can either call this:

```
$object->some_method(-foo=>5, -bar=>10);
```

or this:

```
$object->some_method(5, 10);
```

Remember, for the latter case the order of the parameters are crucial.

1.6 Some design decisions

1.6.1 Exception handling

There are some framework in Perl where they go a roundabout way to make Perl as close as Java. They create an elaborate framework of exception handling. I humbly differ from their opinions and believe that Perl is not meant to be used that way. Most of the calls in BioSVG actually die in case of an internal error. Although stalwarts might disagree but it's my belief (a belief can't be argued) that this is the best way. And I have an argument for it- it actually keeps the code simple!

1.6.2 Classes in BioSVG

When complete BioSVG will comprise of several classes, each representing a biological graphics. For example-

BioSVG::Chromatogram
BioSVG::Plasmid
BioSVG::Annotation
BioSVG::GenBank
BioSVG::GFF

The constructor name in each classes will vary depending on the use. Whenever, the data structure is created from a file (*e.g.*, Chromatogram) the constructor name is-

```
BioSVG::ClassFoo->create_from_file(\ldots);
```

The same class might or might not have a new() function. This function is used only when you are creating an object from scratch-

```
my $plasmid = BioSVG::Plasmid->new(-name=>"pMYPLASMID",
                                   -shape=>"circular",
                                   -size=>8000);
```

Chapter 2

Drawing a Chromatogram