AsyForGiNaC

Documentation for the *Asymptote* output extension for *GiNaC*
Edition 0.1, last updated 14 August 2006
for *AsyForGiNaC* version 0.1, from 14 August 2006

Daniel Seidel

This manual is for *AsyForGinac* (version 0.1), a library adding simple *Asymptote* output facilities to the *GiNaC* library.

Copyright © 2006 Free Software Foundation, Inc.

> Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies. Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Table of Contents

1 Basic issues about this approach 1
- 1.1 General problems 1
- 1.2 The approach 1

2 How to use the extensions 2
- 2.1 The output facilities 2
- 2.2 The drawing options 3
- 2.3 Code examples 6
  - 2.3.1 A very simple example 6
  - 2.3.2 A simple examples 6
  - 2.3.3 A not so simple 3D plot 7

3 For programmers 8
- 3.1 Structure overview 8
- 3.2 `fnctset` 9
- 3.3 Stream manipulators 9

4 Future prospects 11
- 4.1 Complete implementation to the *GiNaC* library 11
- 4.2 Adding new features 11

Index 12

1 Basic issues about this approach

1.1 General problems

The main problems providing an graphic output for *GiNaC* (*GiNaC* Is Not A CAS) objects in *Asymptote* consists in the various *outputfeatures* that can be provided. *Asymptote* has really high developed graphical output facilities combined with a high level programming language, so a lot of things to set and choose. The main idea is to provide the best fitting output format for every object you want to draw, but still leave features changeable. Therefore it is not really senseful to wrap all *Asymptote* features in C++ methods. The experienced user will know the *Asymptote* commands and add some features by typing them directly to the output stream, while the less experienced user would probably be happy with some main features to choose, only typing a normal C function call. On the other hand some features have to be given in the drawing command of the object itself, i.e. it is not possible to add them independently from the provided output. So where is the border?

The next task to solve is to *distinguish different objects* and possible kinds of drawing. It should be possible to draw most of the objects in a common way and concomitant provide an opportunity to add specific drawing methods for your objects, if the normal output is not satisfying. By doing so, the user interface for drawing should stay unchanged.

1.2 The approach

This approach is mostly concerned about a *simple, easy to handle function call* for all programmers dealing with it.

One main feature is the class `class asy::options`, which is the interface to adjust your graphic to your one wishes. There is only this one class `class asy::options`, which has *no* derived classes. That makes it very simple for users. Internal the methods used for drawing are adjusted by a classes derived from `class asy::fnctset`.

At the moment the drawing extensions are in a first state of developement. The recent version is a stand-alone version, it is not possible to adjust your drawing method to the kind of *GiNaC* object. But I am looking forward to include this package to the *GiNaC* package and add the different drawing functions as virtual functions to the `GiNaC::basic` class. Then one can overwrite them for every class derived from `GiNaC::basic`.

2 How to use the extensions

2.1 The output facilities

At the moment it is only possible to draw explicitly given functions. To make use simply, there are streammanipulators defined. The list below shows all available manipulators.

All (except `draw_label`) have the same appearence. So, I prepend a general description first.

```
asy::draw_dummy(GiNaC::ex& obj, GiNaC::ex param, GiNaC::lst borders,
                options *opt = new options, bool optimize=true)
```

Here *obj* is the object you want to draw, *param* are the are the expressions in your object you want use as parameter and *borders* are the intervall borders for drawing your parameters. If you have only one parameter, just pass it as parameter. If there are more, pack them in a `GiNaC::lst`, like `lst(param1[,param2])` for example. The borders have to be a list as well, structured as `lst(min,max)`, where *min* is a list of the minima and *max* the list of maxima of all parameters, e.g. *min* should be `lst(min_param1, min_param2)`. If there is only one parameter, you can write the borders without wrapping *min* and *max* in a list. *opt* contains all drawing settings. The `class asy::options` will be discussed later See Section 2.2 [The drawing options], page 3. *optimize* is of no use at the moment. In further versions it can be used to allow or deny changes in the user defined drawing options by the drawing function itself, in case of a provided ideal drawing setting for special objects.

Caution: Note that *draw_dummy* is not a implemented function. It has to be replaced by one of the following function names.

So, let's see which different drawing functions we have:

```
draw_param(ex& obj, ex param, lst borders
           [, options *opt, bool optimize])
```

`draw_param` draws parameter curves in 2D or 3D. *obj* has to be a `GiNaC::lst`, containing the koordinates depending on the parameter, e.g. `lst(x(t),y(t)[,z(t)])`. The borders can be given as `lst(tmin,tmax)`, where t is your parameter, both borders have to be real numbers or evaluable to real numbers.

```
draw_2D(ex& obj, ex param, lst borders
        [, options *opt, bool optimize])
```

`draw_2D` draws functions mapping from $\mathbf{R^1} \rightarrow \mathbf{R^1}$, e.g. $y = f(x)$, where $f(x)$ is given as the *obj* and x as *param*. The intervall in which the function should be drawn is defined by *borders*, given similar to the borders in `draw_param` as `lst(xmin,xmax)`. For *opt* see the section about drawing options See Section 2.2 [The drawing options], page 3.

```
draw_surface(ex& obj, ex param, lst borders
             [, options *opt, bool optimize])
```

`draw_surface` draws the surface of a function mapping from $\mathbf{R^2} \rightarrow \mathbf{R^1}$, e.g. $z = f(x, y)$, where $f(x, y)$ is given as the *obj* and x, y as *param*, like `lst(xparam,yparam)`.

The area, in which the function should be drawn has to be given through *borders*, like `lst(lst(xmin,ymin),lst(xmax,ymax))`.

For *opt* see the section about drawing options See Section 2.2 [The drawing options], page 3.

```
draw_palette(ex& obj, ex param, lst borders
            [, options *opt, bool optimize])
```

`draw_palette` draws a colored plane, indicating the function values of a function mapping from $\mathbf{R^2} \to \mathbf{R^1}$, e.g. $z = f(x, y)$, where $f(x, y)$ is given as the *obj* and x, y as *param*, like `lst(xparam,yparam)`.

The area, in which the function should be drawn has to be given through *borders*, like `lst(lst(xmin,ymin),lst(xmax,ymax))`.

For `opt` see the section about drawing options See Section 2.2 [The drawing options], page 3.

```
draw_label(const GiNaC::ex& obj, std::string options="",
          std::string picturename="currentpicture")
draw_label(const GiNaC::ex& obj, options *opt)
```

`draw_label` is only a wrapper for the *Asymptote* function `label(...)`. The LaTeX output from *GiNaC* is used to produce the label text. Over the string *options* the optional parameters of the label function of *Asymptote* can be added, except picture, which has to be given separately in the string *picturename*.

If you use the second kind of manipulator with the `asy::options` pointer *opt*, all settings for picture will be used from the *asy::options* object. As label options the options given with `setoptlabel(std::string opt)` will be used.

2.2 The drawing options

This section describes all options that can be set over `class asy::options`, how they work, in wich cases of drawing they are used and what the default settings are.

Often you can set *strings of options*, these strings are an aggregation of options you can use in *Asymptote*. So if you have problems with this options please check the *Asymptote* manual for the possible options and the right order.

import If you produce an '`*.asy`' file for *Asymptote*, containing your object, usually some special *Asymptote* packages have to be imported. The necessary packages are imported automatically by default. However, it might happen, that you draw several objects in one file or you have already imported the necessary packages by your one. To suppress the import, or to enable it again, use `void import(bool set=true)`. To check, if the import will be done use `bool isimport()`.

picture

Asymptote output is printed on so called *pictures*. The standard *picture* is `currentpicture`. To change this `void setpicture(std::string picturename, bool create=true)` is provided. You should use *false* for the *create* parameter, if your picture is already declared in your '`.asy`' output file. To check, if a new picture will be declared use `bool isnewpicture()`. To force a new picture or disable the creation call `void newpicture(bool set=true)`. For setting the picture options (*size*, *keepAspect*) use `void setpicturesize(float xsize, float ysize=0)` and `void setpicturekeepaspect(bool keepaspect=true)`. After setting this things, they will be applied. But you still can disable the settings by calling

`void picturesettings(bool set=true)` with parameter *false*. To check if the settings are applied use `bool ispicturesettings()`. To get the settings use `std::string getnamepicture()`, `bool getkeepaspectpicture()` and `GiNaC::lst getsizepicture()`.

By *default* the picturesettings will be done and your picture is resized to 300x300 pxl with keepAspect. No new picture will be declared by default.

newfile The newfile option only adds a little comment header to your file. For access to this feature the methods `void newfile(bool set=true)` and are provided. By default the header will be drawn.

internalpen

To alter color and linestyle of your picture you can change your pen. There are two options: First, you can use an already declared pen. Therefore use `void setpen(std::string name)` and give the name of your existing pen. Second it is possible to define your pen through the class options. Therefore call `void setpen(std::string name, std::string options)`, or `void setoptpen(std::string options)`. To check if a pen will be defined call `bool isinternalpen()`, to change it call `void internalpen(bool set=true)`.

By *default* the penname is `currentpen` and no internalpen will be defined.

Note that only the drawn object and the label will be affected by these settings, the axis or anything else. For changing things there write to the options of these things, respectively. **Note** that **color** settings (see [color], page 4) are *overwritten* if

color If you don't create your special pen for drawing the object (see [internalpen], page 4), it is also possible to change only the color. The color has to be set in RGB color space. Three different methods to set it are available:

```
void setcolor(unsigned short red, unsigned short green,
              unsigned short blue)
void setcolor(unsigned short color[3])
void setcolor(GiNaC::lst c)
```

The use is self-explaining. To check wether you have set own color settings use `bool iscolor()`. To enable or disable your color settings call `color(bool set=true)`.

By *default* the color settings are disabled and the color is set to black (i.e. in case you only enable the settings, black will appear).

Note if you activate an internal pen and use the color settings parallel, the colorsettings will override the pen color.

axis By *default* axis are drawn with the options "*RightTicks,Arrow*" and the names "\\$*x*\\$", "\\$*y*\\$" and "\\$*z*\\$". To change the settings use the functions

```
void setnamexaxis(std::string name)
void setoptxaxis(std::string opt)
void addoptxaxis(std::string opt)
```

with x replaced by y or z, as the case may be. In the 2D case only the x and y axis will be used. z has no impact then.

To disable or enable axis drawing use `void axis(bool set=true)`. To check if axis will be drawn call `bool isaxis()`.

In the 3D case you can also set the box your axis should go along. This is done by

```
void setaxisbox(GiNaC::lst box)
void setaxisbox(GiNaC::lst min, GiNaC::lst max)
```

, where *box* should be `lst(min,max)`, `min=lst(xmin,ymin,zmin)` and `max=lst(xmax,ymax,zmax)`. Elsewise the axisbox is orientated at the clipregion or the function itsself. Especially the last variant may lead to an unaesthetic picture. Therefore it is *recommended to set* either the *clipregion* or the *axisbox*. The axisbox is assumed to be unset when it is an empty list (*default*). To check this call `bool isaxisbox()`.

clipping To show just a part of the drawn graph it is possible to clip it. This works in 2D and 3D. It will be especially useful, if you draw a 2D function with a pole. This might elsewise not be drawn proper. The clipregion is set by

```
void setclipregion(GiNaC::lst region)
void setclipregion(GiNaC::lst min, GiNaC::lst max)
```

, where *region* should be `lst(min,max)`, `min=lst(xmin,ymin[,zmin])` and `max=lst(xmax,ymax[,zmax])`.

By *default* clipping is disabled. It is automatically enabled when you call one of the `setclipregion(...)` functions. To disable/enable it elsewise use `void clip(bool set=true)`. To check the clip status call `bool isclip()`.

Note that clipping is skipped when the clipregion is not given in the proper format.

scale The axis of your function can be scaled *Linear* or *Log*, for logarithmic. Scaling is disabled by *default*. To enable it call `void setscale(std::string scale)` with a string like '`Log,Linear,Log`' for a 3D plot and something similar to '`Log,Log`' for a 2D plot.

Note that these options are never checked until you try to run your output file with *Asymptote*. So handle them carefully.

Note that scaling only changes the axis. It has *no* impact on the graph itself.

Additional you can set the scale status by `void scale(bool set=true)` and check it with `bool isscale()`. If you just enable scaling without setting the options, it will only produce a comment line in the '`.asy`' file, saying scaling failed.

label To add a label to your graph you can set the text by `void settxtlabel(std::string label)` and the options for this label by `void setoptlabel(std::string opt)`. By *default* no label is drawn, by setting a label name you enable the label automatically. Otherwise it is possible to change the label status by `void label(bool set=true)` and check it by `bool islabel()`. About how to position you label and for other options, please look the *Asymptote* documentation for Label.

perspective

This option is only useful for a *3D plot*. *Asymptote* allows you to set your viewpoint in the threedimensional space. The *Asymptote* `perspective(...)` function is wrapped and can be accessed over

```
void setperspective(GiNaC::lst &perspective)
void setperspective(double x, double y, double z)
void setperspective(double p[3])
```

, where always the new viewpoint is required as parameter. By setting the viewpoint the viewpoint change is activated automatically. By *default* it is disabled. To enable/disable it you can also use `void perspective(bool set=true)`. And to check if it is enabled use `bool isperspective()`.

2.3 Code examples

The use of *AsyForGiNaC* becomes clear by some code snippets. For detailed examples please have a look to the example files, provided in the distribution package (subfolder '`example`').

2.3.1 A very simple example

This example draws us a circle (given as parameter representation) using the standard settings.

First we need to include the appropriate header files

```
#include <ginac/asydraw.h>  //header to include AsyForGiNaC
#include <ginac/ginac.h>    //header to include GiNaC
#include <iostream>         //basic I/O operators
#include <fstream>          //file stream operators
```

Then we define the function and all other needed things as `GiNaC::ex`:

```
ex t = symbol("t");
ex f = lst(sin(t),cos(t));
```

and define our ofstream file

```
ofstream ofstd("./output/param2D_std.asy");
```

Now only one function call produces us the '`.asy`' file:

```
ofstd << asy::draw_param(f,t,lst(0,2*Pi));
```

Easy? Yes, but I have to admit not always sufficient for a really good result (look the other examples in the subfolder '`example`'). But that could improve by including the package to *GiNaC* and make the drawing functions dependend on the objects.

2.3.2 A simple examples

Let us extend our *very simple example* See Section 2.3.1 [A very simple example], page 6 a little bit by adjusting some drawing options. Here are only the additional code snippets listed.

In the end a quarter of the cycle should appear, labeled and also changed in color and penstyle.

First of all we need to declare an `asy::options` structure:

```
asy::options opt;
```

This we can adjust now:

```
opt.setpen("mypen","rgb(150,10,150)+longdashdotted+linewidth(5)");
opt.setclipregion(lst(0,0),lst(1.5,1.5));
opt.setnamexaxis("$x(t)$");
opt.setoptxaxis("BottomTop,LeftTicks");
opt.setnameyaxis("$y(t)$");
opt.setoptyaxis("LeftRight,RightTicks");
opt.settxtlabel("Quarter a circle");
opt.setoptlabel("(20,100),rgb(red)");
```

And finally we call our draw function, now with *&opt* as third parameter:

```
ofman << asy::draw_param(f,t,lst(0,2*Pi),&opt);
```

ofman is here our file output stream.

2.3.3 A not so simple 3D plot

To show some of the 3D options, let us do a 3D surface draw of the function $1 + e^{-(|x|+|y|)^2}$. The whole sourcecode can be find under '`example/surface.cpp`'. First we have to include the headers:

```
#include <ginac/asydraw.h>  //header to include AsyForGiNaC
#include <ginac/ginac.h>    //header to include GiNaC
#include <iostream>         //basic I/O operators
#include <fstream>          //file stream operators
```

and then define our *GiNaC* objects:

```
ex x = symbol("x");
ex y = symbol("y");
ex f = 1+exp(- pow(abs(x)+abs(y),2));
```

also the remaining code stays quite similar to the examples above: First we introduce the `std::ofstream` and our `asy::options` object:

```
ofstream ofman("./output/surface_man.asy");
asy::options opt;
```

and than we adjust our drawing options

```
opt.setcolor(lst(0,150,150));
opt.setaxisbox(lst(-2,-2,0),lst(2,2,5));
opt.setoptxaxis("RightTicks");
opt.setoptyaxis("RightTicks");
opt.setoptzaxis("RightTicks");
opt.settxtlabel("$1+e^{- (|x|+|y|)^2}$");
opt.setoptlabel("rgb(red)");
opt.setsteps(50,50);
```

At last: Let's draw it:

```
ofman << asy::draw_surface(f,lst(x,y),lst(lst(-1,1),lst(-1,1)),&opt);
```

Notice the way the parameters x and y are given and the borders.

3 For programmers

3.1 Structure overview

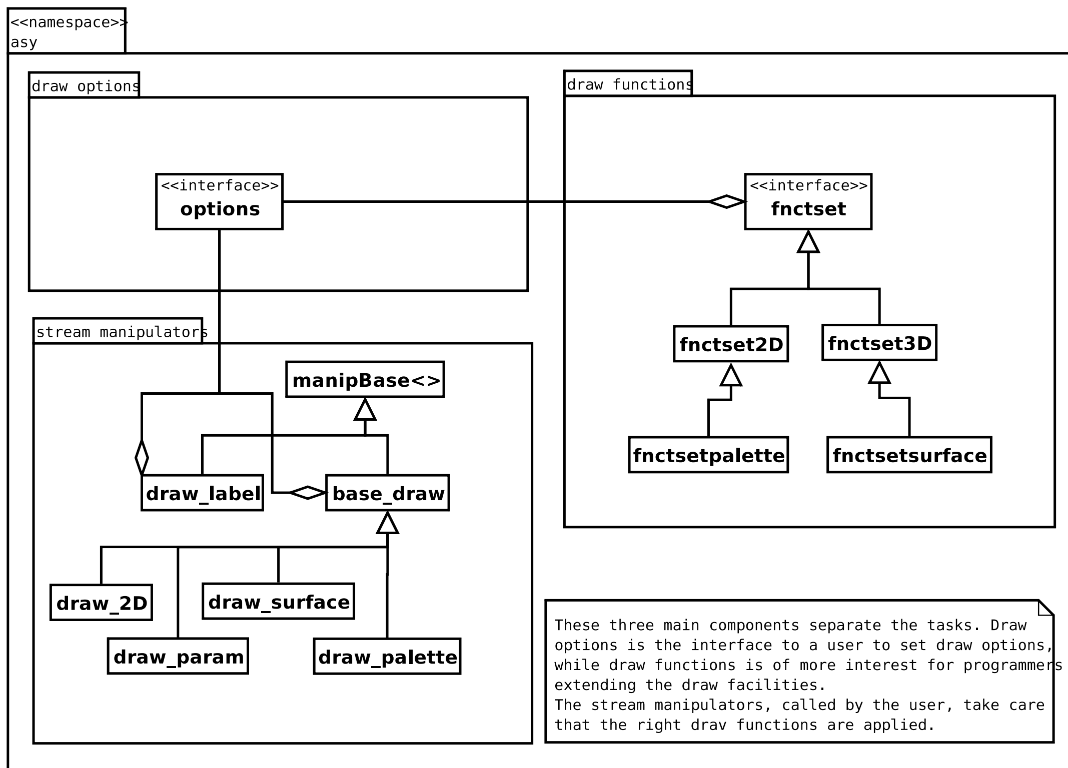

Figure 3.1: structural overview

As shown in Figure (see Figure 3.1) the structure decays in three main parts. The *draw options* component consist only of the class options, which should *abide* in this way, due to the consistant user interface.

The *draw functions* component includes an inheritance tree with the root `fnctset`. This root class `fnctset` provides the *interface* to several draw functions, which are discussed laterwards in the next subsection.

Here it is possible to add new classes, derived from `fnctset` directly or indirectly. This new classes can than be used in your own stream manipulator, or in already existing stream manipulators. The included drawing settings are mainly *independed* from the object which is drawn. And so they can be called for *common* use when drawing specific objects.

The *stream manipulators* implement the calculation of the object specific data. So until now it's possible to change things here, according to the type of object you want to draw. This is not a perfect solution and will be changed, if this package will be included to *GiNaC*. It is intended to make this calculations in some methods of `GiNaC::basic` and the derived

classes. So specifialized drawing functions for each object are easier to implement and the job of the manipulator is more clearly separated from the calculation job.

3.2 fnctset

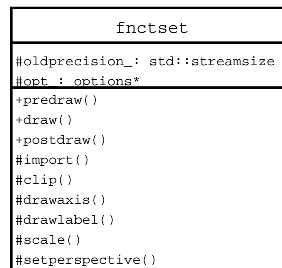

Figure 3.2: draw functions structure

Whenever drawing an object, an *object* of a class derived from `fnctset` should be used. The object provides the methods `predraw()`, `draw()` and `postdraw()`. This public methods call the protected methods, according to the given `options`. E.g., if `options::drawaxis_==true`, `fnctset::drawaxis()` will be called. To get a better understanding of this, contact the source code.

3.3 Stream manipulators

All stream manipulators are derived from an instance of `manipBase<>`. To build your own stream manipulator you have to implement the same scheme. Derive your class directly or indirectly from `maniBase<>` (using `base_draw` as base class is most times useful) and implement it like this:

derived from `base_draw`

```
class my_draw : public base_draw {
public:
        my_draw(GiNaC::ex& obj, GiNaC::ex param,
                GiNaC::lst borders, options *opt = new options,
                bool optimize=true)
         :base_draw(obj,param,borders,opt,optimize) {}
public:
        virtual std::ostream& fct(std::ostream& ost) const;
};
```

derived from `manipBase<>` **instance**

```
class my_draw : public manipBase<my_draw> {
public:
        base_draw::base_draw(GiNaC::ex& obj /*[,my_type my_parameter]*/)
         : manipBase<base_draw>(*this), obj_(obj)
           /*[,my_parameter_(my_parameter)]*/ {}
protected:
```

```
        const GiNaC::ex &obj_;
        /*[my_type my_parameter_]*/
public:
        virtual std::ostream& fct(std::ostream& ost) const {}
};
```

Using the manipulator will work by calling the constructor. This causes `fct()` to be called. So all your draw work should be done in `fct()`. For an example read the source of such an manipulator. This will help you to get a feeling for it and for the collaboration with the `fnctset` functions `predraw()`, `draw()` and `postdraw()`.

4 Future prospects

4.1 Complete implementation to the *GiNaC* library

In the next version this approach should be included into the *GiNaC* library. Therefore the stream manipulator method `fct()` should only call the appropriate draw method implemented *virtual* in `GiNaC::basic`. This allows different draw functions for each class derived from `GiNaC::basic`, respectively.

To manage this, the current draw method code from `fct()` might be copied to the appropriate `GiNaC::basic` draw methods.

4.2 Adding new features

To make the *Asymptote* output really useful, the derived classes of `GiNaC::basic` should provide more specific output facilities. Also to allow implicit given functions to be drawn is really necessary. E.g. a circle given as $r^2 = x^2 + y^2$, and other simple geometrical objects. That might depend on an equation solver.

On the other hand special draw functions for the mathematical functions in *GiNaC* can be added.

All in all feel free to think about extentions that might be useful and try to add it or contact me

Index

3

3D example . . . 7

A

Approach . . . 1
axis options . . . 4

B

Basic issues . . . 1

C

clipping options . . . 5
code examples . . . 6
color options . . . 4

D

design, package . . . 8
drawing functions . . . 9
Drawing functions . . . 2
Drawing functions, structure . . . 2
Drawing, options . . . 3

E

example, not so simple 3D . . . 7
example, simple . . . 6
example, very simple . . . 6
examples . . . 6
extending drawing functions . . . 9
extensions . . . 11

F

features to add . . . 11
`fnctset` . . . 9
functions, drawing . . . 2
future prospects . . . 11

G

General design problems . . . 1
General, structure . . . 1

I

implementation into *GiNaC* . . . 11
import options . . . 3
internalpen options . . . 4

A

`addoptxaxis` . . . 4

L

label options . . . 5

M

Manipulators . . . 2
Manipulators, structure . . . 2

N

newfile options . . . 4

O

options, axis . . . 4
options, clipping . . . 5
options, color . . . 4
Options, drawing . . . 3
options, import . . . 3
options, internalpen . . . 4
options, label . . . 5
options, newfile . . . 4
options, perspective . . . 6
options, picture . . . 3
options, scale . . . 5
Output facilities . . . 2
overview, structure . . . 8

P

package design . . . 8
perspective options . . . 6
picture options . . . 3
Problems, design . . . 1
programmer advices . . . 8
programmers, drawing function description . . . 9
programmers, stream manipulators . . . 9

S

scale options . . . 5
simple example . . . 6
stream manipulators . . . 9
Streammanipulators . . . 2
Streammanipulators, structure . . . 2
structure overview . . . 8
structure stream manipulators . . . 9
Structure, Streammanipulators . . . 2

V

very simple example . . . 6

`addoptyaxis` . . . 4

addoptzaxis . . . 4
asy::draw_param . . . 6
asy::options . . . 3
axis . . . 5

B

base_draw . . . 9

C

class asy::options . . . 3
class base_draw . . . 9
class manipBase<> . . . 9
clip . . . 5
color . . . 4

D

draw . . . 9
draw_2D . . . 2
draw_label . . . 3
draw_palette . . . 2
draw_param . . . 2, 6
draw_surface . . . 2

F

fnctset::draw . . . 9
fnctset::postdraw . . . 9
fnctset::predraw . . . 9
functions, draw_2D . . . 2
functions, draw_label . . . 3
functions, draw_palette . . . 2
functions, draw_param . . . 2
functions, draw_surface . . . 2

G

getkeepaspectpicture . . . 4
getnamepicture . . . 4
getsizepicture . . . 4

I

import . . . 3
internalpen . . . 4
isaxis . . . 5
isaxisbox . . . 5
isclip . . . 5
iscolor . . . 4
isimport . . . 3
isinternalpen . . . 4
islabel . . . 5
isnewfile . . . 4
isnewpicture . . . 3
isperspective . . . 6
ispicturesettings . . . 4
isscale . . . 5

L

label . . . 5

M

manipBase<> . . . 9

N

newfile . . . 4
newpicture . . . 3

O

options . . . 3
options::addoptxaxis . . . 4
options::addoptyaxis . . . 4
options::addoptzaxis . . . 4
options::axis . . . 5
options::clip . . . 5
options::color . . . 4
options::getkeepaspectpicture . . . 4
options::getnamepicture . . . 4
options::getsizepicture . . . 4
options::internalpen . . . 4
options::isaxis . . . 5
options::isaxisbox . . . 5
options::isclip . . . 5
options::iscolor . . . 4
options::isinternalpen . . . 4
options::islabel . . . 5
options::isnewfilebool isnewpicture() . . . 4
options::isnewpicture . . . 3
options::isperspective . . . 6
options::ispicturesettings . . . 4
options::isscale . . . 5
options::label . . . 5
options::newfile . . . 4
options::newpicture . . . 3
options::perspective . . . 6
options::picturesettings . . . 4
options::scale . . . 5
options::setaxisbox . . . 5
options::setclipregion . . . 5
options::setcolor . . . 4
options::setnamexaxis . . . 4
options::setnameyaxis . . . 4
options::setnamezaxis . . . 4
options::setoptlabel . . . 5
options::setoptpen . . . 4
options::setoptxaxis . . . 4
options::setoptyaxis . . . 4
options::setoptzaxis . . . 4
options::setpen . . . 4
options::setperspective . . . 6

options::setpicture 3
options::setpicturekeepaspect 3
options::setpicturesize 3
options::setscale 5
options::settxtlabel 5

P

perspective 6
picturesettings 4
postdraw 9
predraw 9

S

scale 5
setaxisbox 5
setclipregion 5
setcolor 4
setnamexaxis 4
setnameyaxis 4
setnamezaxis 4
setoptlabel 5
setoptpen 4
setoptxaxis 4
setoptyaxis 4
setoptzaxis 4
setpen 4
setperspective 6
setpicture 3
setpicturekeepaspect 3
setpicturesize 3
setscale 5
settxtlabel 5