

# StarCluster Reference Manual

Generated by Doxygen 1.4.3

Mon Jul 18 15:04:06 2005

## Contents

<a href="#">1 StarCluster Directory Hierarchy</a>	<a href="#">1</a>
<a href="#">2 StarCluster Class Index</a>	<a href="#">1</a>
<a href="#">3 StarCluster Directory Documentation</a>	<a href="#">1</a>
<a href="#">4 StarCluster Class Documentation</a>	<a href="#">2</a>

## 1 StarCluster Directory Hierarchy

### 1.1 StarCluster Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

<a href="#">include</a>	<a href="#">1</a>
<a href="#">src</a>	<a href="#">2</a>

## 2 StarCluster Class Index

### 2.1 StarCluster Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">gnuplot (Gnuplot API)</a>	<a href="#">2</a>
<a href="#">range (Defines criteria for restriction of frames returned by <code>get_next_dyn()</code>)</a>	<a href="#">13</a>

## 3 StarCluster Directory Documentation

### 3.1 include/ Directory Reference

#### Files

- file `gnuplot.hh`
- file `plot.hh`
- file `tool.hh`

## 3.2 src/ Directory Reference

### Files

- file `col2dyn.cc`
- file `dyn2col.cc`
- file `gnuplot.cc`
- file `plot_lagrad.cc`
- file `plot_positions.cc`
- file `plot_radial_density.cc`
- file `plot_radial_vdisp.cc`
- file `tool.cc`
- file `xdynplot.cc`

## 4 StarCluster Class Documentation

### 4.1 gnuplot Class Reference

*Gnuplot* API

```
#include <gnuplot.hh>
```

#### Public Member Functions

- `gnuplot` (const bool=true)  
*constructor for the gnuplot class*
- `~gnuplot` ()  
*destructor for the gnuplot class*
- `gnuplot & flush` (const bool=true)  
*ends the current plot and flushes all data to Gnuplot*
- `gnuplot & operator<<` (const string)  
*passes a string to Gnuplot as a command*
- `gnuplot & operator<<` (const double)  
*passes a number to Gnuplot for plotting*
- `gnuplot & operator<<` (`gnuplot &(*f)(gnuplot &)`)
- void `reset_fp` ()  
*resets output back to Gnuplot*

- void `set_fp` (FILE \*const, const bool=false)  
*redirects output away from the Gnuplot pipe and to a file*
- void `set_graphs` (unsigned short)  
*creates multiple graphs in one plot*
- void `set_log` (const string="")  
*enables log scaling of specified axes*
- void `set_multiplot` (const unsigned char=1, const unsigned char=1)  
*creates tiled plots or plots of multiple datasets*
- void `set_multiplot` (const string)  
*creates tiled plots or plots of multiple datasets*
- void `set_nolog` (const string="")  
*disables log scaling of specified axes*
- void `set_nomultiplot` ()  
*disables multiplot mode*
- void `set_output` (const string)  
*creates a Postscript plot*
- void `set_title` (const string)  
*sets the title of the plot*
- void `set_xlabel` (const string)  
*sets the label of the X axis*
- void `set_xrange` (const string)  
*sets the extend of the X axis*
- void `set_ylabel` (const string)  
*sets the label of the Y axis*
- void `set_yrange` (const string)  
*sets the extend of the Y axis*
- void `set_zlabel` (const string)  
*sets the label of the Z axis*

- void [set\\_zrange](#) (const string)  
*sets the extend of the Z axis*

### Public Attributes

- bool [autoscale](#)  
*if set, autoscale mode is used for every plot*
- int [pause](#)  
*number of seconds to pause between plots*
- bool [plot3d](#)  
*make 3-D plots*
- string [style](#)  
*style of the plot, i.e., points, lines, dots, ...*
- float [time](#)  
*time associated with the current plot*
- string [title](#)  
*title of the plot*
- string [xlabel](#)  
*label for the x axis*
- string [ylabel](#)  
*label for the y axis*
- string [zlabel](#)  
*label for the z axis*

#### 4.1.1 Detailed Description

This class implements a somewhat incomplete but very convenient and simple to use *Gnuplot* API for the *StarCluster* package.

## 4.1.2 Constructor & Destructor Documentation

### 4.1.2.1 gnuplot::gnuplot (const bool = true)

Creates a default `gnuplot` object and sets up a pipe to a *Gnuplot* process with `popen()`.

### 4.1.2.2 gnuplot::~~gnuplot ()

Closes the pipe to the object's *Gnuplot* process with `pclose()` and deallocates its memory.

## 4.1.3 Member Function Documentation

### 4.1.3.1 gnuplot & gnuplot::flush (const bool = true)

This function is used to terminate the current plot and flush the data to the *Gnuplot* process. It's optional only in the simplest cases when it's sufficient for `~gnuplot()`, called explicitly or at the end of scope, to complete the plot:

```
#include "gnuplot.hh"

int main(void) {
    gnuplot gp;
    gp << 1 << 2 << ...;
}
```

In situations where more sophisticated features are used, e.g., tiled plots, multiple graphs in one plot, or animations, `flush()` must be invoked to signal the end of datasets. It can be called in two ways, explicitly:

```
gp.flush();
```

or, more naturally, through the overloaded operator `<<` (`gnuplot&` (`*f`) (`gnuplot&`)) and a nonmember `gnuplot& flush(gnuplot&):`

```
gp << flush;
```

#### Note:

The optional argument is intended primarily for internal use. When `flush(false)` is called after a dataset, the value of `pause` is ignored. For example:

```
gp.pause = -1;
gp << 1 << 2 << ... << 3.14159, gp.flush(false); // no pause
gp << 2 << 3 << ... << flush; // pause here
```

Notice that

```
gp << flush(false);
```

syntax isn't supported; it's another indication that this usage is deprecated.

### 4.1.3.2 `gnuplot & gnuplot::operator<< (const double)`

This is the most often used function of this class, and the one that, by far, contains the most logic. It's used to send numbers to be plotted to *Gnuplot*. For example, if `gp` is a `gnuplot` object,

```
gp << 2 << 3;
```

or

```
gp << 2; gp << 3;
```

will send the coordinate pair, (2,3), to Gnuplot. It will not, however, be plotted right away, as `gp` will be expecting more numbers, unless it's the last statement in the program (or `gp~gnuplot()` is invoked through a different mechanism) or `gp.flush()` is called explicitly or, equivalently, by

```
gp << flush;
```

The simplicity of this entire interface comes at the cost of the complexity of this function: it contains about as much code as the rest of the class combined. The goal was such that after setting some flags describing the desired plot configuration, the user would simply send numbers without other formatting information and the `gnuplot` object would keep track of the current state. For example, to plot three datasets in one graph:

```
gnuplot gp;
gp.set_graphs(3);
gp << 1 << 2;      // first point of the first dataset
gp << 3 << 4;      // first point of the second dataset
gp << 5 << 6;      // first point of the third dataset
gp << 7 << 8;      // second point of the first dataset
...               // same pattern continues
gp << flush;      // plot is finished
```

This particular sequence is very convenient as many *Starlab* functions, e.g., `compute_general_mass_radii()`, produce output in exactly this order.

### 4.1.3.3 `gnuplot & gnuplot::operator<< (const string)`

Used to pass arbitrary commands to Gnuplot. This member function is intended to extend *Gnuplot* support beyond the provided interface. For example, if `gp` is a `gnuplot` object,

```
gp << "set title \"{/Symbol=18 r}(t)\"\\n\";
```

will set the title of the plot to  $\rho(t)$  ( $\rho(t)$ ) in 18 pt *Symbol* font. Many provided interface member functions are implemented like this. Another major use of this function is to plot functions, perhaps together with data:

```
gnuplot gp;
gp.set_multiplot();
gp << 0.0 << 0.0 << 1.57 << 1 << 3.14 << 0.0 << flush;
gp << "plot sin(x) with lines 2\n";
```

This will plot the three points and a *sine* curve through them. Notice that the data is given first - this is the preferred order as it defines the limits of the plot, and the function is plotted only in that area.

**Note:**

When this or any other function that issues commands is used during an active plot, i.e., after `operator<<(double)` but before `flush()`, the current plot is terminated with `flush()`. This is, of course, because *Gnuplot* will, otherwise, produce a syntax error as it will not interpret commands in the middle of reading numeric data.

**4.1.3.4 void gnuplot::reset\_fp ()**

Used to reverse the action of `set_fp()`. After invoking this member function, the pipe is restored to point to the original *Gnuplot* process.

**4.1.3.5 void gnuplot::set\_fp (FILE \* const, const bool = false)**

This member function is intended for debugging purposes or if tweaking the produced *Gnuplot* script is desired. It redirects the output of the API to a file and away from the *Gnuplot* process. The file pointer is usually provided by `c_file_ptr()` from `tool.hh`; it must be valid and point to a file opened for writing. If the second, optional, argument is `true`, then, instead of the entire script (commands and data), only the data is output. It is, however, in the same order as would have been output to *Gnuplot*, which is not necessarily the same as input order if plotting more than one dataset with `set_graphs()`. For example:

```
gnuplot gp;
gp.set_fp(stdout, true);
gp.set_graphs(3);
gp << 1 << 2;
gp << 2 << 3;
gp << 3 << 4;
...
```

will produce this output:

```
1 2
```

```

...
<newline>
2 3
...
<newline>
3 4
...
<newline>

```

It's action can be reversed with [reset\\_fp\(\)](#), which resets the file pointer back to the original *Gnuplot* process.

#### 4.1.3.6 void gnuplot::set\_graphs (unsigned short)

This member function allows to plot multiple datasets together. The data is expected in the order described under [operator<<\(double\)](#) and [set\\_fp\(\)](#).

See also:

[set\\_multiplot\(unsigned char, unsigned char\)](#) if a "dataset at a time" order is preferred

#### 4.1.3.7 void gnuplot::set\_log (const string = " ")

Enable a logarithmic scale for the specified axes. The argument can specify the desired axes as well the the log base. If no argument is supplied, the defaults, all axes and base 10, are used. Some examples are:

```

gp.set_log();           // log base 10 scaling of all axes
gp.set_log("x");       // log base 10 scaling of the X axis
gp.set_log("xy 2.71828") // natural log scaling of both axes

```

#### 4.1.3.8 void gnuplot::set\_multiplot (const string)

This function parses its string argument and calls [set\\_multiplot\(unsigned char, unsigned char\)](#) with appropriate arguments. It's intended to be given a command-line option directly, which must be in one of two `scanf ( )` formats:

- **MxN** (" %hhux%hhu ") - M by N tile of plots
- **N** (" %hhu ") - N by N square tile of plots

See also:

[set\\_multiplot\(unsigned char, unsigned char\)](#)

**4.1.3.9 void gnuplot::set\_multiplot (const unsigned char = 1, const unsigned char = 1)**

Creates tiled plots or plots of multiple datasets. If called without arguments, it simply plots all datasets in the same window until [set\\_nomultiplot\(\)](#) is invoked. Otherwise, it creates a tile of plots of the specified, M by N, configuration. In the first case, it differs from [set\\_graphs\(\)](#) in two respects: the expected order of data, which a dataset at a time in this mode, and the inability to specify in advance the number of desired plots. In particular, both functionalities can't be utilized simultaneously, i.e., if several datasets are to be plotted and the plots tiled, [set\\_graphs\(\)](#) must also be used. For example, to produce a 3x3 tile of plots:

```
gnuplot gp;
gp.set_multiplot(3,3);           // or gp.set_multiplot("3")
gp << 1 << 2 << ... << flush;   // the first plot
gp << 3 << 4 << ... << flush;   // the second one, placed next to it
...                               // and so on, for seven more plots
```

To plot several datasets together in the same plot:

```
gnuplot gp;
gp.set_multiplot();
gp << 1 << 2 << ... << flush;   // the first dataset
gp << 3 << 4 << ... << flush;   // the second one, placed in the same plot
...                               // and so on, ...
```

**4.1.3.10 void gnuplot::set\_nolog (const string = " ")**

Disable logarithmic scaling for the specified axes. For this function, the argument must either specify the axis or be empty, in which case linear scaling is restored for all axes.

**4.1.3.11 void gnuplot::set\_nomultiplot ()**

Disables multiplot mode previously enabled with [set\\_multiplot\(unsigned char, unsigned char\)](#).

See also:

[set\\_graphs\(\)](#)

**4.1.3.12 void gnuplot::set\_output (const string)**

Instead of producing an X display plot, make a *Postscript* file. If the argument is `-`, it will be sent to *stdout*, otherwise to the requested file.

#### 4.1.3.13 void gnuplot::set\_title (const string)

Sets the title of the plot; it will be placed at the top of the plot in the heading. And if `time` is not `INVALID_TIME`, it will also be followed, on the next line, by "`t = X`", where `X` is the time associated with the snapshot. Additionally, when producing *Postscript* plots, it may be desirable to use *LaTeX* escapes for Greek letters and other symbols for titles as well as axis labels. These are defined in *Gnuplot* documentation but some examples are:

- `rho(t): "{/Symbol r}(t)"  $\implies \rho(t)$`
- `erf(x): "2/{/Symbol \326 p} {{/Symbol=18 \362}@_0^x} e^{-u^2} du"  $\implies 2/\sqrt{\pi} \int_0^x e^{-u^2} du$`

**See also:**  
[title](#)

#### 4.1.3.14 void gnuplot::set\_xlabel (const string)

Sets the label of the X axis.

**See also:**

- [set\\_title\(\)](#) for examples of *LaTeX* escapes
- [xlabel](#) for slightly different semantics

#### 4.1.3.15 void gnuplot::set\_xrange (const string)

The range of the X axis is passed as a string in *Gnuplot* format, omitting the outer brackets. For example, to set the extend of the X axis from -2 to an autoscaled value,

```
gp.set_xrange("-2:*")
```

#### 4.1.3.16 void gnuplot::set\_ylabel (const string)

Sets the label of the Y axis.

**See also:**

[set\\_xlabel\(\)](#)

#### 4.1.3.17 void gnuplot::set\_yrange (const string)

Sets the limits of the Y axis.

**See also:**

[set\\_xrange\(\)](#)

**4.1.3.18 void gnuplot::set\_zlabel (const string)**

Sets the label of the Z axis.

See also:

[set\\_xlabel\(\)](#)

**4.1.3.19 void gnuplot::set\_zrange (const string)**

Sets the limits of the Z axis.

See also:

[set\\_xrange\(\)](#)

**4.1.4 Member Data Documentation****4.1.4.1 bool gnuplot::autoscale**

If set, autoscale mode is used for every plot, not just the first one. Otherwise, only the first plot is auto-scaled, and the following ones inherit those axis limits. This is off by default as it produces very annoying rescaling and recentering of plots when only the central portion is "interesting."

**Warning:**

The action of this variable isn't completely reversible. More precisely, the limits are saved only from the first dataset and if autoscaling is desired after the first dataset was plotted, `set_xrange(":*")/set_yrange(":*")` must be used instead. In other words, `autoscale` has the described effect only before the first call to `flush()`, where it's decided whether or not to save the limits.

**4.1.4.2 int gnuplot::pause**

When this variable is set to a positive value, it's interpreted as the number of seconds to pause between successive plots (or screens, if in multiplot mode). If it's equal to -1, *StarCluster* will exit. And if the value of `pause` is less than -1, *StarCluster* will wait for a key press before producing the next plot/screen unless `stdin` isn't connected to a terminal, e.g., data is being read from `stdin`. This value can be ignored under two other circumstances: `flush(false)` was called (instead of `flush()`) or no actual *Gnuplot* session is active, instead scripts or data are being output as a result of `set_fp()`.

**4.1.4.3 bool gnuplot::plot3d**

make 3-D plots

#### 4.1.4.4 string `gnuplot::style`

This determines the style of the plot. Any valid *Gnuplot* style can be specified: `points`, `lines`, `linespoints`, ...

#### 4.1.4.5 float `gnuplot::time`

Holds the time associated with the current plot. If set, this number will be used to add time information to the title of the plot. To make a simple X-Y projection of an N-body system:

```
gnuplot gp;
gp.title = "X-Y Positions";
gp.xlabel = "x", gp.ylabel = "y";
while (dyn* r = get_next_dyn()) {
    gp.time = r->get_system_time();
    for_all_leaves(dyn, r, d)
        gp << d->get_pos()[0] << d->get_pos()[1];
    gp << flush;
    rmtree(r);
}
```

If it's desirable, on the other hand, to omit this information, `time` must be set to `INVALID_TIME`.

See also:

[set\\_title\(\)](#)

#### 4.1.4.6 string `gnuplot::title`

This variable holds the title of the plot and is used before the first dataset is given with [operator<<\(double\)](#). It's commonly used in tools utilizing the `gnuplot` library to set the default title before parsing command-line options. Using [set\\_title\(\)](#) would be equivalent except for one subtle difference: the *Gnuplot* code would be output immediately and if the output were changed with [set\\_fp\(\)](#) using command-line options, which are parsed after setting the default, the code setting the title would never be output to the specified stream. When using the variable, however, [set\\_title\(\)](#) doesn't have to be called explicitly and is only necessary if the title is to be changed after the first dataset was plotted:

```
gnuplot gp;
gp.title = "Title Example";           // convenient to use the variable here
gp << 1 << 2 << ... << flush;
gp.set_title("Another Example");     // here, the function form must be used
gp << 2 << 3 << ... << flush;
```

The same difference applies to the `xlabel/set_xlabel()` and `ylabel/set_ylabel()` pairs as well as, partly, to [autoscale](#).

#### 4.1.4.7 string `gnuplot::xlabel`

Label for the X axis.

See also:

[title](#)

#### 4.1.4.8 string `gnuplot::ylabel`

Label for the Y axis.

See also:

[xlabel](#)

#### 4.1.4.9 string `gnuplot::zlabel`

Label for the Z axis.

See also:

[xlabel](#)

The documentation for this class was generated from the following files:

- `gnuplot.hh`
- `gnuplot.cc`

## 4.2 range Class Reference

defines criteria for restriction of frames returned by `get_next_dyn()`

```
#include <tool.hh>
```

### Public Member Functions

- `range` (const char[ ]="", const range\_type=NONE)  
*constructor for the range class*

### Friends

- `dyn * get_next_dyn` (const `range` &)  
*given a range object, returns the next frame*

### 4.2.1 Detailed Description

The `range` class contains all the necessary information such that `get_next_dyn()` will return the next frame as specified by its `range` argument. This class currently implements two kinds of ranges, based on frame number and time. Only one type of selection criterion, however, can be active at one time. See the description of the constructor below for further details.

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 `range::range (const char[] = " ", const range_type = NONE)`

A `range` object is initialized from a string and range type that describes how to interpret the string, which, in turn, must be in a format compatible with that type. Only one type, or criterion, can be in effect at one time. Currently, the defined range types are:

- `NONE` Unrestricted range, i.e., all frames in the input are accepted.
- `FRAME` The argument is interpreted as a range of frames, where the first frame is 1 (not 0). This means that the string contains positive integers and conforms to one of the following, *Standard C scanf()* formats:
  - `X-Y,Z (" %u-%u, %u ")` - frame `X` to frame `Y` skipping every `Z` frames
  - `X-Y (" %u-%u ")` - frame `X` to frame `Y`
  - `,Z (" , %u ")` - all frames skipping every `Z` frames
  - `X (" %u ")` - frame `X`
- `TIME` The argument is interpreted as a range of times. In other words, only frames from the specified time interval, as determined by *Starlab's* `dyn::get_system_time()`, are returned. The string must be in one of the following formats:
  - `X-Y,Z (" %g-%g, %g ")` - frames in the time interval `[X,Y]` at least `Z` time units apart
  - `X-Y (" %g-%g ")` - frames in the time interval `[X,Y]`
  - `,Z (" , %g ")` - all frames at least `Z` time units apart
  - `X (" %g ")` - frame at time `X`

The documentation for this class was generated from the following files:

- `tool.hh`
- `tool.cc`

## Index

- ~gnuplot
  - gnuplot, 4
- autoscale
  - gnuplot, 11
- flush
  - gnuplot, 5
- gnuplot, 2
  - ~gnuplot, 4
  - autoscale, 11
  - flush, 5
  - gnuplot, 4
  - operator<<, 5, 6
  - pause, 11
  - plot3d, 11
  - reset\_fp, 7
  - set\_fp, 7
  - set\_graphs, 8
  - set\_log, 8
  - set\_multiplot, 8
  - set\_nolog, 9
  - set\_nomultiplot, 9
  - set\_output, 9
  - set\_title, 9
  - set\_xlabel, 10
  - set\_xrange, 10
  - set\_ylabel, 10
  - set\_yrange, 10
  - set\_zlabel, 10
  - set\_zrange, 11
  - style, 11
  - time, 12
  - title, 12
  - xlabel, 12
  - ylabel, 13
  - zlabel, 13
- include/ Directory Reference, 1
- operator<<
  - gnuplot, 5, 6
- pause
  - gnuplot, 11
- plot3d
  - gnuplot, 11
- range, 13
  - range, 14
- reset\_fp
  - gnuplot, 7
- set\_fp
  - gnuplot, 7
- set\_graphs
  - gnuplot, 8
- set\_log
  - gnuplot, 8
- set\_multiplot
  - gnuplot, 8
- set\_nolog
  - gnuplot, 9
- set\_nomultiplot
  - gnuplot, 9
- set\_output
  - gnuplot, 9
- set\_title
  - gnuplot, 9
- set\_xlabel
  - gnuplot, 10
- set\_xrange
  - gnuplot, 10
- set\_ylabel
  - gnuplot, 10
- set\_yrange
  - gnuplot, 10
- set\_zlabel
  - gnuplot, 10
- set\_zrange
  - gnuplot, 11
- src/ Directory Reference, 1
- style
  - gnuplot, 11
- time

gnuplot, [12](#)  
title  
gnuplot, [12](#)  
xlabel  
gnuplot, [12](#)  
ylabel  
gnuplot, [13](#)  
xlabel  
gnuplot, [13](#)