Jgraph – A Filter for Plotting Graphs in PostScript

James S. Plank – Princeton University

ABSTRACT

Jgraph is a non-interactive filter for plotting two-dimensional scatter, line, and bar graphs in PostScript. It has also been used as a general-purpose drawing utility. Jgraph's strengths lie in its portability, flexibility, and integration into the UNIX environment. Jgraph is free software available on `netlib` or by anonymous `ftp`.

Introduction

Scientists in all disciplines frequently need to display information graphically on a variety of high-quality output devices. However, there is no standard tool on the UNIX platform that achieves this purpose. Although many software packages exist to facilitate plotting graphs, they all have limitations. Some are only available on certain machines; some can only be integrated into certain text processing systems; some require specific data formats; some are available only as part of colossal computing environments.

Jgraph attempts to provide a simple, yet flexible and powerful graph-plotting package. It is a filter that takes a description of a graph or graphs as input, and produces PostScript [1] as output. PostScript was chosen because it is a standard format for producing high-quality graphic output. PostScript can be viewed on the computer screen with a PostScript viewer like `gs`, printed directly on PostScript printers, or, when in *encapsulated PostScript (EPS)* format, embedded in a text or graphics processing system such as TeX, LaTeX, `troff`, Scribe, or Adobe Illustrator *88*. Moreover, since PostScript is in ASCII format, it can be stored on all hardware platforms and sent freely in all electronic mail systems. Jgraph has the option of producing either EPS or regular PostScript files.

Unlike almost all other graph-plotting packages, jgraph is *non-interactive*. In these days of ''user-friendly'' systems, this might be seen as a disadvantage, but the advantages of this decision are three-fold. First, it allows jgraph to be used on all platforms, as it is not bound to specific terminal types, window systems or even operating systems. Second, it means that jgraph can solve one problem – graph plotting – and solve it well. This is in contrast to systems that provide their own editors, window systems, output viewers, etcetera, which are bound to conflict with the ones to which their users are accustomed. Finally, by being non-interactive, jgraph integrates well with the powerful utilities in UNIX (e.g., `sed`, `nawk`, `make`). Jgraph can be used in makefiles and as part of multistage UNIX pipes, and it can also execute shell commands from within its input. This gives the user a great deal of flexibility often absent from other graph-plotting packages.

Jgraph is free, portable, and well-documented. It is public-domain software that can be obtained over the internet either through `netlib`[1] or by anonymous `ftp`.[2] It is written in machine-independent C and comes with an 18-page manual and many example graphs, including those presented in this paper. It has been installed at over 60 locations under various operating systems, including all flavors of UNIX, as well as VMS and DOS. I am not aware of any environment containing a C compiler on which jgraph cannot be installed.

Jgraph Overview

Jgraph reads a description of graphs on the standard input and produces PostScript on the standard output. The input format is simple enough to let users create useful graphs as soon as they start learning the tool, yet flexible enough be general-purpose. Input consists of keywords followed by values, where a value is either a number, a string or another keyword. White space is ignored except within strings, so that input files may be indented for readability as in the figures below.

Appendix A gives a complete formal specification of the jgraph syntax. This section gives an overview of the salient features of jgraph, as well as a flavor for typical jgraph input and output files.

The major unit of jgraph's input is a *graph*: Users may specify any number of graphs for jgraph to plot on a page. Each graph consists of the following parts: *X* and *Y* axes, curves, strings, a title, a legend, and a position relative to other graphs.

The most important part of a graph are the *curves*. Users may specify any number of curves in a graph. Each curve consists of points, mark

[1]Send email with only the text: `send jgraph.shar from misc` to `netlib@ornl.gov`.

[2]`Ftp` to `princeton.edu`, and get the file `pub/jgraph.Z`.

attributes, line attributes, and a label. The points are (*x*, *y*) pairs that are plotted in the order given. Mark attributes define what gets plotted at each point (e.g., nothing, a circle, a box, text, or a bar-graph line to either axis). Line attributes define what kind of line gets plotted between points (e.g., none, solid, dotted). The label defines the legend entry for the curve.

Jgraph chooses defaults for all attributes, making simple graphs simple to create. The example in Figure 1 below shows the jgraph input and realized PostScript output of a simple graph with three curves. The topmost curve lets jgraph choose all the curve attributes–the only things specified are the points. The middle one plots triangles connected by a solid line, and the bottom one plots just a dashed line between the points. Jgraph sets up default values for all other parts of the graph.

```
newgraph

  newcurve
    pts 0 6  1 9  2 11  3 14  4 18 5 20

  newcurve
    marktype triangle
    linetype solid
    pts 0 3  1 4  2 7  3 9  4 10 5 13

  newcurve
    marktype none
    linetype dashed
    pts 0 0  1 2  2 3  3 5  4 6  5 9
```

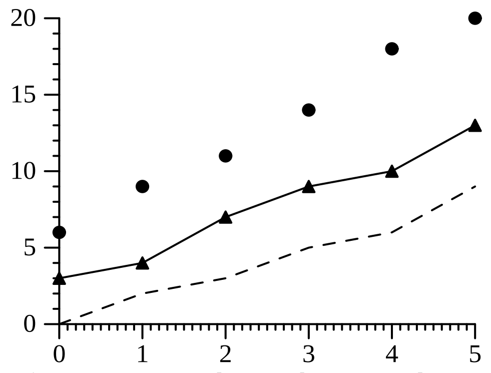

Figure 1: A simple jgraph input and output

Users may change the other graph attributes just as the curve attributes are changed in Figure 1. For example, for both axes, users may alter the axis size, maximum and minimum values, scaling (linear or logarithmic), location and spacing of hash marks, etcetera. Users also have control over the appearance and location of a graph's legends and titles, as well as the ability to plot arbitrary text strings anywhere on a graph.

Example 1: Figure 2 shows a more complex example graph in which many of the jgraph defaults are changed to get a desired effect. Here a label has been added to the x-axis, the y-axis is not drawn, and two strings are plotted with each bar: one to describe the bar, and one to state the bar's value. Note also the use of `copystring`, which copies the default values from the previous string. The tokens `copycurve` and `copygraph` are defined to do the same thing for curves and graphs.

```
newgraph
xaxis
  size 3 min 0 max 41
  mhash 1 (* Put 1 tick between hash marks *)
  hash_labels font Times-Italic
  label : Qualifications...

yaxis
  size 1.5 min 0
  nodraw (* Don't draw the y-axis *)

newcurve marktype ybar marksize 0 .6 fill .9
  pts 41 4   35 3   17 2   14 1

newstring
  hjl vjc  (* These define justification *)
  fontsize 9
  font Helvetica-Narrow
  x 1 y 4 : Led league in wins

(* Copystring copies the defaults *)
copystring y 3 : Played for first place team
copystring y 2 : Led league in ERA
copystring y 1 : Led league in K's

copystring x 42 y 4 : 41
copystring x 36 y 3 : 35
copystring x 18 y 2 : 17
copystring x 15 y 1 : 14

newstring
  hjr vjb
  fontsize 6
  x 41 y 0.1
  : Source: USA Today research
```

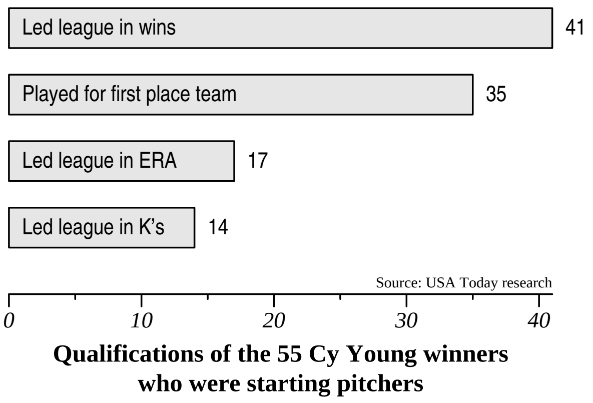

Figure 2: A more complex example

The treatment of strings is one of the elegant features of jgraph. All strings and string-like attributes are treated in the same manner. That is, strings, axis labels, the title, hash labels, legend entries and text marks are all manipulated by the same keywords. For example, there is a special string for each axis called `hash_labels`, which treats all hash labels on that axis as a unit. Thus, for example, the user can change the font on all the hash labels by changing the font of the string hash_labels, as in Figure 2 above. Similarly, there is a special string for legends that controls all the legend entries as a unit.

Jgraph supports grayscales and color. Users can set the color or grayness of every part (strings, axes, lines, marks) of each graph. Figure 2 uses grayscale to shade each bar. Figure 4 shows a far more complex and effective use of grayscale in jgraph.

Accessing UNIX from jgraph

Jgraph's include and shell constructs allow users to include files and shell commands from within the jgraph input. This has two benefits. First, it enables the user to specify his or her own formats for data files and extract the data using UNIX utilities such as sed, nawk, or even C programs. This is in opposition to other programs which require data to be in a specific format.

Second, it frees jgraph from attempting to provide function plotting. Some graph-plotting packages include a facility to plot functions, usually something resembling a subset of a more general language (such as an expression evaluator in C with certain math libraries included, as in gnuplot). Jgraph omits any such facility, because users usually have their own resources for evaluating mathematical expressions which are more robust and powerful than those included in typical graph-plotting packages. The shell construct allows users to tap the powers of these resources in a simple and concise way.

Example 2: This example shows how to use the shell construct for both data extraction and function-plotting. In this example, the user has timed a program which sorts indexed records using a binary tree and would like to see how its running time compares with the theoretical running time of $O(n \log n)$, where n is the number of records. The program's output for varying values of n has been stored in the text file data.txt, which has the following format:

```
Number of records = 0      Time = 0
Number of records = 5000   Time = 2
Number of records = 10000  Time = 3
...
```

Thus the user can extract the data points for a graph of n versus time with a simple nawk script, which prints columns five and eight of data.txt. This is done in the first curve of the jgraph input in Figure 3. Next, the user wants to plot the function $n \log n / k$, where k is a constant that makes the data in data.txt fit the function. After determining a value of $k = 35000$ the user can plot the function using the nawk script in the second curve of Figure 3. Thus, the shell construct of jgraph gives the user all the powers of the tools available under UNIX.

More complex graphs and drawings

Since jgraph allows users to control all parts of a graph and lets them arrange multiple graphs on a page, it can be used to plot arbitrarily complex graphs and even general purpose drawings. Since jgraph is non-interactive, it can be used as a back-end graphics language for making drawings that use graph constructs (such as axes and legends) or that have an iterative structure. Figure 4 is an example by Dave Wortman [11] which uses jgraph in such a way. The input file for this picture was created by a nawk script that processes data and emits jgraph output. ''WYSIWYG'' drawing tools like xfig or MacDraw are not suited for such tasks.

Figures 5 and 6 show further examples of complex, structured drawings that are straightforward to produce with jgraph but would be difficult to produce with a WYSIWYG tool. Figure 5, from [9] is a jgraph drawing which depicts processor communication over time. It makes use of jgraph's ability to plot axes and legends in a general-purpose drawing. Figure 6 is a jgraph drawing produced by a nawk script written by Adam Buchsbaum that takes a

```
xaxis size 2.5
  hash_labels font Helvetica
  label : Number of indexed Records (N)
yaxis size 2.1
  label : Running time (seconds)
  hash_labels font Helvetica

newcurve
  marktype cross
  label : Data
  pts shell : nawk '{print $5, $8}' data.txt

newcurve
  marktype none linetype solid
  label : N log N / 35000
   pts shell : nawk \
     ' $5 != 0 { \
       print $5,  $5 * log($5) / 35000}' \
     data.txt
```

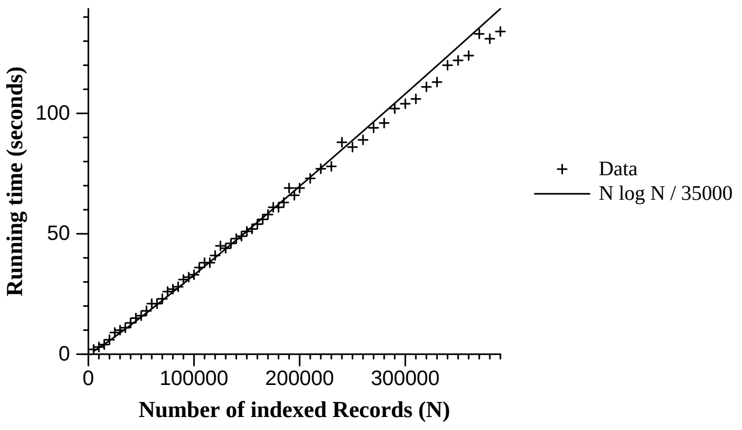

Figure 3: A more complex graph using the shell construct of jgraph

description of trees and produces jgraph output [4]. It treats jgraph as a convenient back-end graphics language.

Related Work

There are many programs that can be used for graph-plotting, ranging from simple filters like jgraph, to more complex software packages. The two standard UNIX programs for graph-plotting are `graph` [7] and `grap` [3]. Like jgraph, both are non-interactive filters, with `graph` producing output for the UNIX `plot` routine, and `grap` producing `pic` output for inclusion in `troff` documents.

`Graph` is a primitive program whose functionality comprises a restricted subset of jgraph's. `Grap` on the other hand, is a powerful tool with

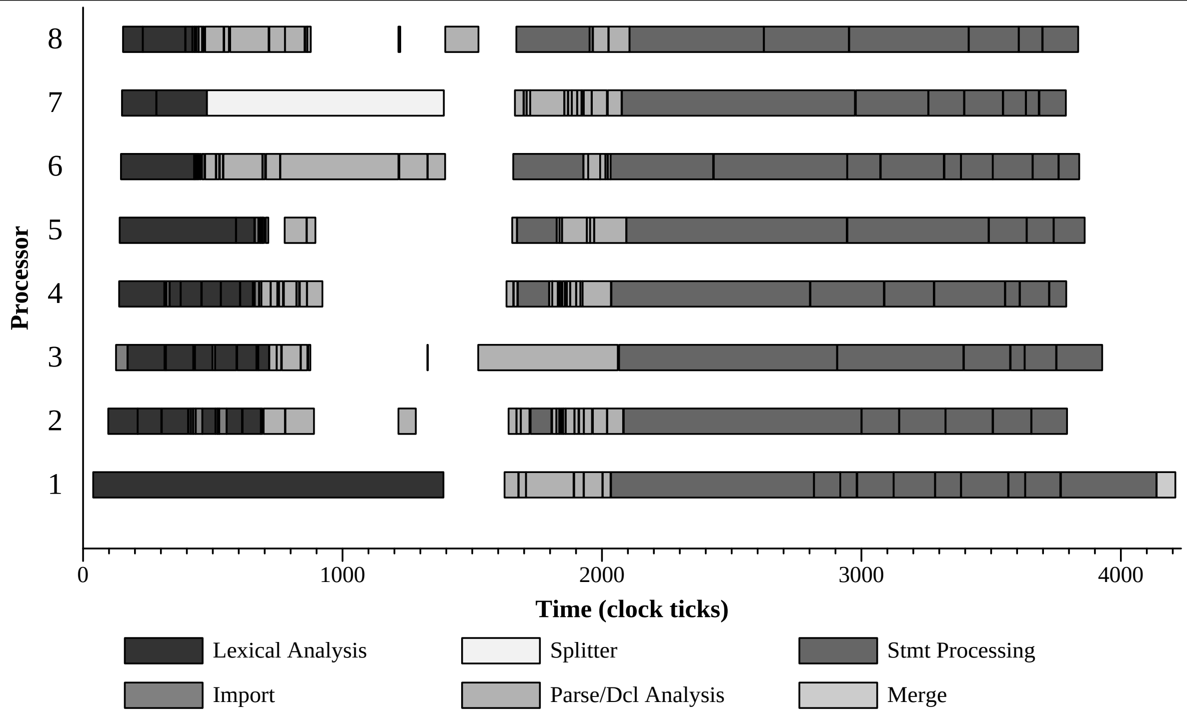

Figure 4: Results of a `nawk`-to-jgraph data processing program

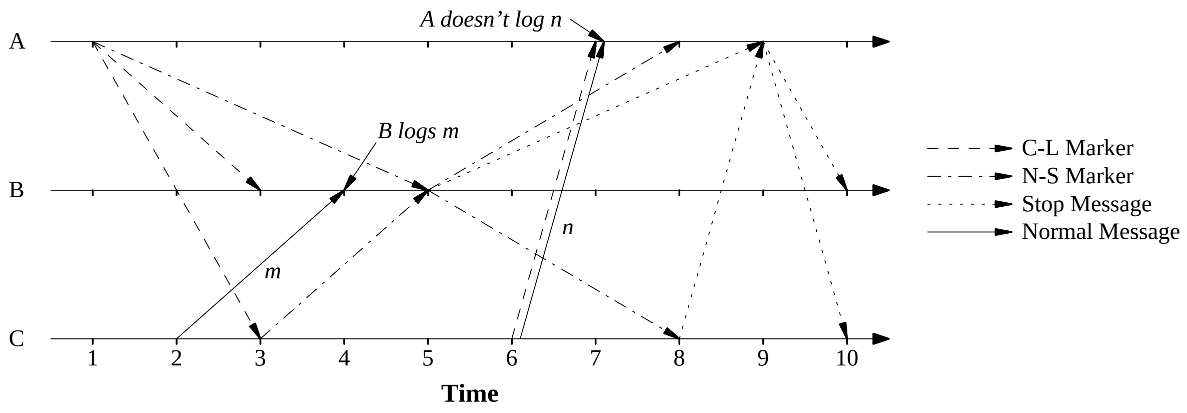

Figure 5: A jgraph drawing depicting processor communication

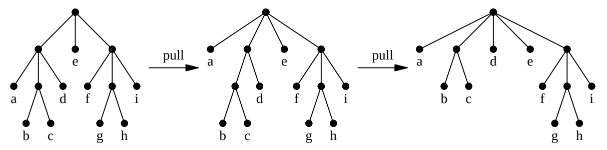

Figure 6: Results of a `nawk`-to-jgraph tree-drawing program

many of the same advantages as jgraph in terms of flexibility. However, `grap` was designed for use with `pic` and `troff` and therefore suffers from a few problems. First, `troff` and its family of programs were designed before the advent of today's high-quality PostScript printers. Therefore, the output of such programs, even when converted into PostScript, is often inferior to programs such as LaTeX, Scribe, or Adobe Illustrator *88*. Second, it is non-trivial to convert `grap` output into usable PostScript. For example, one can get TeX from `grap` by using the program `tpic`, and one can get printable PostScript from `grap` by using `psroff` or `psdit`. However, it is impossible to get encapsulated PostScript without hand-editing output files. Third, although `grap` is considered a standard part of UNIX, it is not available on all UNIX systems and is not easily ported to non-UNIX systems. Finally, most users (at least in the computer science community) use TeX and LaTeX instead of `troff` to process text, so they aren't prepared to take advantage of the flexibility offered by `grap`, as it relies on a thorough knowledge of `pic` and `troff` macros and constructs.

There are many interactive programs for drawing graphs: `Xgraph` [8], `Gnuplot` [10], and `Mathematica` [12] all run under UNIX. `Xgraph` is best described as `graph` with an Xwindows interface. Like `graph`, it suffers from a lack of flexibility. `Gnuplot` and `Mathematica` on the other hand are quite powerful, including facilities for function-plotting and 3D graph-plotting as well as for scatter, line, and bar graph plotting. Their interactiveness, however, makes them more cumbersome to use than jgraph for all but the simplest of plots, and in the tasks to which both they and jgraph are applicable, jgraph has the simpler interface.

There are other graph-plotting programs for non-UNIX systems, such as CricketGraph [6] and Excel [5] for the Macintosh and other personal computers, and RS/1 [2], a massive data processing package available on VMS. None of these are portable to Unix systems, nor are any of them free software.

Acknowledgements

The author would like to thank Matt Blaze, Heather Booth, Adam Buchsbaum, and Norman Ramsey for their comments concerning this paper, Kent Landfield and Reed Wade for helping to distribute the software, and Dave Wortman for creating beautiful jgraph drawings. The author has been funded in part by an AT&T fellowship.

References

[1] Adobe Systems Incorporated. *PostScript Language Reference Manual*. Addison-Wesley, Reading, Massachusetts, 1985.

[2] Bolt, Beranek, and Newman. *RS/1 Users Guide*. BBN Software Product Corporation, 1987.

[3] Jon L. Bentley and Brian W. Kernighan. GRAP – A language for typesetting graphs tutorial and user manual. Technical Report #114, AT&T Bell Laboratories, December 1984.

[4] Adam L. Buchsbaum and Robert E. Tarjan. Confluently persistent deques via data structural bootstrapping. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms*, January 1993.

[5] Douglas F. Cobb, Allan McGuffy, and Mark Dodge. *Microsoft Excel 3 companion*. Microsoft Press, Redmond, Washingon, 1991.

[6] *Desktop Software Guide*. Computer Associates International Inc., Islandia, NY, 1992.

[7] graph – draw a graph. Unix man page, 1983.

[8] David Harrison. xgraph – draw a graph on an x11 display. Unix man page, 1989.

[9] Kai Li, Jeffrey F. Naughton, and James S. Plank. An efficient checkpointing method for multicomputers with wormhole routing. *International Journal of Parallel Processing*, 20(3), June 1992.

[10] Thomas Williams. gnuplot – an interactive plotting program. Unix man page, 1990.

[11] David B. Wortman and Michael D. Junkin. A concurrent compiler for Modula-2+. *ACM SIGPLAN '92 Conference on Programming Language Design and Implementation, in SIGPLAN Notices*, 27(7):68-81, July 1992.

[12] Stephen Wolfram. *Mathematica, A System for Doing Mathematics by Computer*. Addison-Wesley, Redwood City, California, 1988.

Author Information

Jim Plank is an assistant professor at the University of Tennessee in Knoxville. He received his PhD from Princeton University in December, 1992. His research area is general fault-tolerance in parallel and distributed computing. Jgraph is a hobby. He can be reached at: Department of Computer Science; University of Tennessee; Knoxville, TN 37966 or by Email at plank@cs.utk.edu.

APPENDIX A: Formal Syntax of Jgraph

```
<top-level> :=   % Top Level commands
   <nil> |
   <top-level>* |

   newgraph <graph> |  % Choose/edit graphs
   graph <int> <graph> |
   copygraph [<int>] <graph> |

   newpage |           % General layout commands
   bbox <int> <int> <int> <int> |
   X [<float>] | Y [<float>] |
   preamble <file> | epilogue <file>

<graph> :=
   <nil> |
   <graph>* |

   newcurve <curve> |       % Edit curves
   curve <int> <curve> |
   copycurve [<int>] <curve> |
   newline <curve> |

   xaxis <axis> |           % Edit the attributes
   yaxis <axis> |           % of each axis

   newstring <string> |     % Edit and plot
   string <int> <string> |  % arbitrary strings
   copystring [<int>] <string> |

   title <string> |      % Edit the graph’s title
   legend <legend> |     % Edit the legend

   border | noborder |   % Draw a border around the graph
   clip | noclip |       % Clip inside this border

   x_translate [<float>] |  % The graph’s position
   y_translate [<float>]    % relative to other graphs

<curve> :=       % These commands allow the user to
   <nil> |       % enter curve points and attributes
   <curve>* |   % enter curve points and attributes

   pts [<float> <float>]* |  %Point definitions
   x_epts [<float> <float> <float> <float>]* |
   y_epts [<float> <float> <float> <float>]* |

   marktype <marktype> |     % Mark definitions
   marksize [<float>] [<float>] |
   mrotate [<float>] |
   gmarks [<float> <float>]* |
   postscript <file> |
   fill [<float>] | cfill [<float> <float> <float>] |

   linetype <linetype> |      % Line definitions
   linethickness [<float>] |
   glines [<float>]* |
   gray [<float>] | color [<float> <float> <float>] |
   pfill [<float>] | pcfill [<float> <float> <float>] |
   bezier | nobezier |

                              % Arrowheads on lines
   larrow | rarrow | nolarrow | norarrow |
   larrows | rarrows | nolarrows | norarrows |
   asize [<float>] [<float>] |
   afill [<float>] | acfill [<float> <float> <float>] |

   label <string>   % The legend entry
   clip | noclip |  % Whether to show points outside the
                    % max and min axis values

<marktype> :=            % Different types of marks
   none | general |
   circle | box | diamond | triangle | x |
   cross | ellipse | xbar | ybar | text | postscript

<linetype> :=             % Different types of lines
   none | general |
   solid | dotted | dashed | longdash |
   dotdash | dotdotdash | dotdotdashdash
```

```
<string> :=     % These commands let the user change the
   <nil> |      % appearance and location of any string
   <string>* |

   : <chars> |
   x [<float>] | y [<float>] |
   rotate [<float>] |
   hjl | hjr | hjc | vjt | vjb | vjc |  % Justification
   font <fontname> | fontsize [<float>] |
   linesep [<float>] |
   lgray [<float>] | lcolor [<float> <float> <float>] |

<axis> :=       % These commands let the user edit the
   <nil> |      % attributes of an axis
   <axis>* |

   linear | log | log_base [<float>] |
   min [<float>] | max [<float>] | size [<float>] |

   label <string> |

   draw | nodraw |
   gray [<float>] | color [<float> <float> <float>] |

   draw_axis | nodraw_axis | draw_at [<float>] |
   draw_axis_label | nodraw_axis_label |
   grid_lines | no_grid_lines |
   mgrid_lines | no_mgrid_lines |

   hash [<float>] |    % These commands let the user change
   shash [<float>] |   % the appearance of the hash marks
   mhash [<int>] |     % and labels
   precision [<int>] |
   hash_at [<float>] | mhash_at [<float>] |
   hash_label <hash_label> |
   hash_labels <string> |
   hash_scale [<float>] |
   draw_hash_marks | nodraw_hash_marks |
   draw_hash_labels | nodraw_hash_labels |
   draw_hash_marks_at [<float>] |
   draw_hash_labels_at [<float>] |
   auto_hash_marks | no_auto_hash_marks |
   auto_hash_labels | no_auto_hash_labels

<hash_label> :=    % These commands let the user create
   <nil> |         % his or her own hash labels
   <hash_label>* |
   at [<float>] |
   : <chars>


<legend> :=        % These commands govern the legend
   <nil> |
   <legend>* |

   on | off | left | right |        % Location
   top | bottom | custom |
   x [<float>] | y [<float>] |

   linelength [<float>] |
   linebreak [<float>] |
   midspace [<float>] |

   defaults <string>


% Other tokens are obvious -- e.g. <int> and <float>.
% At any point in the input, you may have:

include <file>    % Include the contents of the <file>.
shell : <chars>   % Execute the <chars> as a shell command
                  % and include the contents of stdout.

(* <chars> *)     % Comments, which are ignored
```