UCRL-MA-128569, Manual 4

The Python Graphics Interface, Section III

Plotter Objects Manual

Written by

Zane Motteler
Lee Busby
Fred N. Fritsch

October 13, 1998

Table of Contents

CHAPTER 1: The Python Graphics Interface

1.1 Overview of the Python Graphics Interface

The Python Graphics Interface (abbreviated PyGraph) provides Python users with capabilities for plotting curves, meshes, surfaces, cell arrays, vector fields, and isosurface and plane cross sections of three dimensional meshes, with many options regarding line widths and styles, markings and labels, shading, contours, filled contours, coloring, etc. Animation, moving light sources, real-time rotation, etc., are also available. PyGraph is intended to supply a choice of easy-to-use interfaces to graphics which are relatively independent of the underlying graphics engine, concealing the technical details from all but the most intrepid users. Obviously different graphics engines offer different features, but the intention is that when a user requests a particular type of plot which is not available on a particular engine, the low level interface will make an intelligent guess and give some approximation of what was asked for.

There are two such graphics packages which are relatively independent of the underlying plotting library. The Object-Oriented Graphics (OOG) Package defines geometric objects (Curves, Surfaces, Meshes, etc.), Graph objects which can be given one or more geometric objects to plot, and Plotter objects, which receive geometric objects to plot from Graph objects, and which interface with the graphics engine(s) to do the actual plotting. A Graph can create its own Plotter, or the more capable user can create one or more, handy when one wishes (for instance) to plot on a remote machine, or to open graphics windows of different types at the same time. The second such package is called EZPLOT; it is built on top of OOG, and provides an interface similar to the command-line interface of the Basis EZN package. Some of our long-time users may be more comfortable with this package, until they have mastered the concepts of object-oriented design.

As mentioned above, a Graph object needs at least one Plotter object to plot itself; only the Plotter objects need know about graphics engines. At present we have two types of Plotter objects, one which knows about Gist and one which knows about Narcisse. Some power users may prefer to use the lower-level library-specific function calls, but most users will use EZPLOT or OOG.

Gist is a scientific graphics library written in C by David H. Munro of Lawrence Livermore National Laboratory. It features support for three common graphics output devices: Xwindows, (color) PostScript, and ANSI/ISO Standard Computer Graphics Metafiles (CGM). The library is small (written directly to Xlib), portable, efficient, and full-featured. It produces x-vs.-y plots with ‘‘good’’ tick marks and tick levels, 2-D quadrilateral mesh plots with contours, vector fields, or pseudocolor maps on such meshes. 3-D plot capabilities include wire mesh plots (transparent or opaque), shaded and colored surface plots, isosurface and plane cross sections of meshes containing data, and real-time animation (moving light sources and rotations). The Python Gist module `gist.py` and the associated Python extension `gistCmodule` provide a Python interface to this library (referred to as PyGist).

Narcisse is a graphics library developed at out sister laboratory at Limeil in France. It is especially strong in high-quality 3-D surface rendering. Surfaces can be colored in a variety of ways, including colored wire mesh, colored contours, filled contours, and colored surface cells. Some combinations of these are also possible. We have also added the capability of doing isosurfaces and plane sections of meshes, which is not available in the original Narcisse. The Python Narcisse module `narcissemodule` (referred to as PyNarcisse) provides a low-level Python interface to this library. Unlike Gist, Narcisse does not currently write automatically to standard files such as PostScript or CGM, although it writes profusely to its own type of files unless inhibited from doing so, as described below. However, there is a "Print" button in the Narcisse graphics window, which opens a dialog that allows you to write the current plot to a postscript file or to send it to a postscript printer.

1.2 Using the Python Graphics Interface

In order to use PyGraph, you first need to have Python installed on your system. If you do not have Python, you can obtain it free from the Python pages at `http://www.python.org`. You may need the help of your system administrator to install it on your machine. Once you have Python, you have to know at least a smattering of the language. The best way to do this is to download the excellent tutorial from the Python pages, sit down at your computer or terminal, and work your way through it.

Before using the Python Graphics Interface, you should set some environment variables as follows.

- Your `PATH` variable should contain the path to the `python` executable.
- You should set a `PYTHONPATH` variable to point to all directories that contain Python extensions or modules that you will be loading, which may include the OOG modules, `ezplot`, and `narcissemodule` or `gistCmodule`. Check with your System Manager for the exact specifications on your local systems.
- Unless you create your own plotter objects, PyGraph will create a default Gist Plotter which will plot to a Gist window only. If you want your default Plotter to be a Narcisse Plotter, then set the variable `PYGRAPH` to `Nar` or `Narcisse`.

A Gist Plotter object automatically creates its own Gist window and then plots to that window. Narcisse, however, works differently. Narcisse is established as a separately running process, to which the Plotter communicates via sockets. Thus, to run a Narcisse Plotter, you must first open a Narcisse.[1] To

1. I am going to assume that you already have Narcisse installed on your system, and its directory path in your `PATH` variable.

do so, you need to go through the following steps:

1. Set your environment variable `PORT_SERVEUR`[1] to 0.
2. Start up Narcisse by typing in the command `Narcisse &`. It will take a few moments for the Narcisse GUI to open, then immediately afterwards it will be covered by an annoying window which you can eliminate by clicking its `OK` button.
3. You will note that there is a server port number given on the GUI. Set your `PORT_SERVEUR` variable to this value.
4. Narcisse has an annoying habit of saving everything it does to a multitude of files, and notifying you on the fly of all its computations. If you do a lot of graphics, these files can quickly fill up your quota. In addition, the running commentary on file writing and computation on the GUI is time-consuming and slows Narcisse down to a truly glacial pace. To avoid this, you need to turn off a number of options via the GUI before you begin. They are all under the `STATE` submenu of the `FILE` menu, and should be set as follows: set ''`Socket compute`'' to "`no`,'' set "`File save`'' to "`nothing`,'' set ''`Config save`'' to "`no`,'' and set "`Ihm compute`'' to "`no`.'' ("IHM'' are the French initials for ''GUI.'')

1.3 About This Manual

This manual is part of a series of manuals documenting the Python Graphics Interface (PyGraph). They are:

- **I.** EZPLOT User Manual
- **II.** Object-Oriented Graphics Manual
- **III.** Plotter Objects Manual
- **IV.** Python Gist Graphics Manual
- **V.** Python Narcisse Graphics Manual

EZPLOT is a command-line oriented interface that is very similar to the EZN graphics package in Basis. The Object-Oriented Graphics Manual provides a higher-level interface to PyGraph. The remaining manuals give low-level plotting details that should be of interest only to computer scientists developing new user-level plot commands, or to power users desiring more precise control over their graphics or wanting to do exotic things such as opening a graphics window on a remote machine.

PyGraph is available on Sun (both SunOS and Solaris), Hewlett-Packard, DEC, SGI workstations, and some other platforms. Currently at LLNL, Narcisse is installed only on the X Division HP and Solaris boxes, however, and Narcisse is not available for distribution outside this laboratory. Our French col-

1. We did tell you that Narcisse was French, didn't we?

leagues are going through the necessary procedures for public release, but these have not yet been crowned with success. Gist, however, is publicly available as part of the Yorick release, and may be obtained by anonymous ftp from `ftp-icf.llnl.gov`; look in the subdirectory `/ftp/pub/Yorick`.

A great many people have helped create PyGraph and its documentation. These include

- Lee Busby of LLNL, who wrote `gistCmodule`, and wrought the necessary changes in the Python kernel to allow it to work correctly;
- Zane Motteler of LLNL, who wrote `narcissemodule`, `ezplot`, the OOG, and some other auxiliary routines, and who wrote much of the documentation, at least the part that was not blatantly stolen from David Munro and Steve Langer (see below);
- Paul Dubois of LLNL, who wrote the `PDB` and `Ranf` modules, and who worked with Konrad Hinsen (Laboratoire de Dynamique Moleculaire, Institut de Biologie Structurale, Grenoble, France) and James Hugunin (Massachusetts Institute of Technology) on `NumPy`, the numeric extension to Python, without which this work could not have been done;
- Fred Fritsch of LLNL, who produced the templates and did some of the writing of this documentation;
- Our French collaborators at the Centre D'Etudes de Limeil-Valenton (CEL-V), Commissariat A L'Energie Atomique, Villeneuve-St-Georges, France, among whom are Didier Courtaud, Jean-Philippe Nomine, Pierre Brochard, Jean-Bernard Weill, and others;
- David Munro of LLNL, the man behind Yorick and Gist, and Steve Langer of LLNL, who collaborated with him on the 3-D interpreted graphics in Yorick. We have also shamelessly stolen from their Gist documentation; however, any inaccuracies which crept in during the transmission remain the authors' responsibility.

The authors of this manual stand as representative of their efforts and those of a much larger number of minor contributors.

Send any comments about these documents to ‘‘`support@icf.llnl.gov`’’ on the Internet or to ‘‘`support`’’ on Lasnet.

CHAPTER 2: Introduction to Plotter Objects

2.1 PyGraph `Plotter` Objects

A `Plotter` object is the lowest level in the PyGraph object-oriented graphics (OOG). Its routines interact with the C extension to Python which drives the particular graphics engine associated with that particular `Plotter`. `Plotter` objects are the only components of the OOG which ‘‘know’’ the details of the innards of a graphics engine. It is our intention that `Graph` objects and geometric objects should be independent of the graphics used, and `Plotter`s should ignore (or second guess) requests that cannot be honored by their brand of graphics. The interface that a `Plotter` supplies to the outside world is intended to be essentially the same no matter what the underlying graphics is; unfortunately this is not completely true, because the ways that one connects to different graphics engines is generally incompatible. Thus the calling sequences for instantiations will differ somewhat; however, the internal functions of `Plotter`s will have the same names and calling sequences.

Under most circumstances, a user will not need to instantiate or interact with `Plotter` objects. If you create some geometric objects, hand them to a `Graph` object, and ask the `Graph` to plot itself without having first created a `Plotter` object, then the `Graph` object will examine the environment variable `PYGRAPH`, and depending on whether it has been set to `Gist` or `Nar`, will instantiate a PyGist or Py-Narcisse `Plotter` object and then use it to plot itself. If there is no `PYGRAPH` environment variable, then a default PyGist `Plotter` will be instantiated.

The following circumstances, however, mean that the user must have at least some basic knowledge about `Plotter` objects and their instantiation:

- Sending a picture to more than one `Plotter`.
- Sending a picture to a `Plotter` open on a remote machine.
- Plotting both to PyNarcisse and PyGist.
- Plotting to a cgm or postscript file (PyGist only).

In the next two chapters we shall discuss how to instantiate PyGist and PyNarcisse `Plotter` objects. Once instantiated, these objects can be passed to `Graph` objects for them to use in their plotting. If a `Graph` object has been informed of more than one `Plotter`, then when it is asked to plot itself, it will cycle through the ones that it knows about and plot its geometric object(s) on each one.

2.2 Functions Common to all `Plotter` Objects

The methods in `Plotter` (in alphabetical order) are described below. Bear in mind that some of these methods are peculiar to one kind of `Plotter` and may not do anything at all for another kind (they are present but the body consists only of a `Python` “`pass`’’ statement). In other cases their actions may be different depending on the `Plotter` type. Most users, even those who need to instantiate multiple and/or remote `Plotter`s, will not need to use these methods, as the `Graph` objects will do the interfacing.

Note that in accordance with Python naming conventions, if you do call any of these methods, then you must precede the method name by the plotter name followed by a period.

`add_object (crv)` : will add a curve `crv` to an existing plot. (See `plot_object`, below.)

`add_text (str, x, y, size, color="fg", tosys = 1)` : will add the specified text `str`, in `color`, to the plot at point `(x, y)` in data coordinates if `tosys` is `1`, otherwise in absolute window coordinates.

`clear_text ()` : gets rid of all text strings in the plot.

`close ()` : closes the connection to the graphics engine.

`do_generic (graf)` : apply the graph-generic attributes of the `Graph` object `graf` to the plot (generic attributes do not depend on the number of dimensions).

`freeze_graph ()` : keeps a graph from being plotted until `send_graph ()` is called.

`move_light  (i)` : a drawing function which returns 0 if `i` exceeds the internal variable `nframes`, otherwise computes lighting angles and calls `light3` (See page 57 in Python Gist Graphics Manual) and `draw3` (See page 59 in Python Gist Graphics Manual).

`move_light_source (graf, angle, nframes)` : Makes a movie (See “The movie module and function” on page 80. of Python Gist Graphics Manual) of a moving light source shining on the 3-D geometric object in the `Graph` object `graf`. The light source will rotate through `angle` radians in each of `nframes` frames.

`new_frame ()` : carries out a frame advance.

`plot2d (graf)` : A `Graph2d` object calls `plot2d` with itself (`graf`) as argument. `plot2d` sorts out everything for the graph and then does the plot.

`plot3d (graf)` : A `Graph3d` object calls `plot3d` with itself (`graf`) as argument. `plot3d` sorts out everything for the graph and then does the plot.

`plot_object (crv)` : a general purpose 2-D plotting routine. It should be called with one argument (`crv`), a `Curve`, `QuadMesh`, `PolyMap`, `CellArray`, or `Lines` object. In the case of multiple objects on one graph, the first call only should be to this routine, subsequent calls to `add_object`. `plot_object` does some one-time things, such as plotting the titles.

`plot_text ()` : sends the accumulated texts out to the graph (see `set_text` and `clear_text`)..

`query ()` : returns -1 if it is not connected to a graphics engine or is but can't seem to find it; 0 if it’s not sure; and 1 if it is connected.

`quick_plot (graf)` : plots without recomputing.

`reset_xyequal ()` : turns off a flag that forces equal scales on the axes.

`rotate_graph (axis, angle, nframes)` : rotates the current plot about `axis`, for a total of `nframes` frames, with rotation through `angle` in each frame. Uses `spin3` (See “The spin3 function” on page 83. in Python Gist Graphics Manual).

`send_color_card ()` : If the `Plotter` has been told about a color card (or palette) (see set_color_card, below), then apprise the graphics engine of this.

`send_generics (graf)` :sets up all the things that are generic to any graph. It does not actually do any plotting yet.

`send_graph ()` : causes a plot that has been accumulated after `freeze_graph` was called, to be plotted.

`set_3d_grid_type (gt)` : sets what the wire grid will look like in a 3d surface plot in one of the wire modes. The choices for `gt` are `'x'` (x lines only), `'y'` (y lines only) and `'xy'` (both x and y lines).

`set_axis_labels ('x_label', 'y_label', 'z_label', 'yr_label')` : All arguments are optional. Default values (from right): `' '`, `'Z axis'`, `'Y axis'`, `'X axis'`.

`set_axis_lin (ax)` : `ax` can be `'x'`, `'y'`, `'yr'`, `'z'`, `'c'`, or `'all'`. The specified axis will have a linear scale.

`set_axis_log (ax)` : `ax` can be `'x'`, `'y'`, `'yr'`, `'z'`, `'c'`, or `'all'`. The specified axis will have a logarithmic scale.

`set_axis_max (ax, val)` : `ax` can be `'x'`, `'y'`, `'yr'`, `'z'`, or `'c'`.The maximum of the specified axis will be set to `val`. `val` should be a PyFloat object.

`set_axis_min (ax, val)` : `ax` can be `'x'`, `'y'`, `'yr'`, `'z'`, or `'c'`. The minimum of the specified axis will be set to `val`. `val` should be a PyFloat object.

`set_bytscl (cmin, cmax)` : ensures that `bytscl` will be called for the next `plf` command, with these values of `cmin` and `cmax`.

`set_c_contours (arg)` : sets various properties when doing 4d contour (iso), smooth, or flat plots. It accepts one argument, as follows: if an integer `n`, sets the number of contours to `n`. This also clears the contour levels array. Countour levels will be computed automatically from the data. if a string: `'lin'` plots the contours linearly spaced. `'log'` plots the contours logarithmically spaced. if an array of type `Float`: sets the contour levels to the values in the array.

`set_color_card ( cardspec, now = 0)` : indicates a predefined color card (palette) for a plot. `cardspec` can be the name of a palette or, in Narcisse, an integer `n`. See the Narcisse manual for the values of `n` and the color card selected (sec. 4.2.134, parametre_map). If `now` is set to `1`, then the palette will be sent immediately.

`set_connect (cn)` : tells whether to connect two or more surface plots, which presumably improves masking. `cn=1` to connect, `cn=0` to disconnect.

`set_default_axes_limits ()` : sets the graphics to compute the maximum and minimum of the axes depending on the data.

`set_distance (arg)` : sets the distance of the view point from a 3-D plot. If called with no argument, or 0, this distance is effectively infinite. Otherwise it should be called with an integer from 1 to 20. Smaller means closer, and hence somewhat more distortion. The size of the plot is not changed.

`set_freeze_each (val)` : tells whether or not to re-freeze the graphics after each `send_graph` call. 1 to re-freeze, 0 not to.

`set_grid_type ( string )` : determines how intrusive the axes and coordinate grids are. The legal arguments are: `'none'`--no axes and grids are drawn. `'axes'`--axes with tick marks. `'wide'`--widely spaced grid in x and y (2-D or 3-D). `'full'`--narrowly spaced grid in x and y (2-D or 3-D). If no argument is specified, the default is `'axes'`.

`set_label_type (arg)` : determines whether curve labels will be attached to the ends of curves, or enclosed in a box. The allowed arguments are thus `'end'` and `'box'`.

`set_link (ln)` : tells whether to link two or more surfaces plotted with different 3d options into one plot (otherwise all surfaces will have the same options). `ln=1` to link, `ln = 0` not to link. This needs to be set to 1 for all surfaces except the last. Connection must not be set (see `set_connect ()`). The axes must not be plotted for surfaces after the first.

`set_linlin ()` : sets both x and y axes to linear scale.

`set_linlog ()` : sets x axis to linear, y axis to logarithmic.

`set_loglin ()` : sets x axis to logarithmic, y axis to linear.

`set_loglog ()` : sets both x and y axes to logarithmic scale.

`set_mask (arg)` : determines whether hidden parts of a 3-D plot will be shown, and if not, what algorithm will be used to determine what is hidden. The allowed arguments and masking algorithm are as follows: `'none'`--no masking. in wire grid mode, all grid lines are visible. `'min'`--the surface is traced beginning in the corner closest to the observer. `'max'`--the surface is traced beginning in the corner farthest from the observer. `'sort'`--a cell sorting is carried out to determine the masking.

`set_no_concat ()` : turns off the 2-D and 3-D concatenation mode.

`set_text (str, ix)` : sets the `ix`th text to `str`.

`set_text_color (col, ix)` : sets the `ix`th text color to `col`, which is a color number associated with a color table, or the name of a common color.

`set_text_pos (x, y, ix)` : positions the `ix`th text at `(x, y)`, which are real numbers between 0 and 1 giving relative position in the graphics window.

`set_text_size (sz, ix)` : sets the `ix`th text size to `sz`. Narcisse and Gist differ on this (sorry!). In Narcisse, `sz` represents essentially the number of characters that will fill the width of the graphics screen, so the larger the number, the smaller the text. In Gist, `sz` is the point size, so the larger the number, the larger the text.

`set_title_colors (bottom_color, top_color, left_color, right_color)` : All arguments are optional, integers representing a color in some color map, or the names of common colors. Missing arguments default to `"fg"`.

`set_titles ('bottom', 'top', 'left', 'right')` : All arguments are optional. Missing ones default to `' '`.

`set_tosys (val)` : if `val` is nonzero, use user coordinates. If zero, use absolute window coordinates. This applies to plotted text only.

`set_x_axis_limits (min, max)` : sets the limits on the x axis to the specified (pyFloat) sizes.

`set_xyequal ()` : Make the x and y axes the same scale.

`set_y_axis ( 'left' , n )` or `set_y_axis ( 'right' , n )` : causes curve number `n` to be associated with the left or right y axis.

`set_y_axis_limits (min, max)` : sets the limits on the y axis to the specified (pyFloat) sizes.

`set_yr_axis_limits (min, max)` : sets the limits on the yr axis to the specified (pyFloat) sizes.

`set_z_c_switch (sw)` : tells whether to switch the roles of the z and c variables in a 4-D plot. `sw=1` to do the switch, `sw=0` not to do it.

`set_z_contours (arg)` : sets various properties when doing 3-D contour (iso), smooth, or flat plots. It accepts one argument, as follows: if an integer `n`, sets the number of contours to `n`. This also clears the contour levels array. Countour levels will be computed automatically from the data. if a string: `'lin'` plots the contours linearly spaced. `'log'` plots the contours logarithmically spaced. if an array of type `Float`: sets the contour levels to the values in the array.

`synchronize ()` : Synchronizes with Narcisse. Important because of socket communication; race conditions can occur. Does nothing in Gist.

`type ()` : Returns `NarType` or `GistType` depending on the type of Plotter. (You need to import `graftypes` in order to test for these.)

CHAPTER 3: Plotter Instantiation

3.1 Instantiation of a Gist `Plotter` Object

Calling Sequence

```
import GistPlotter
pl1 = GistPlotter.Plotter ( [<display>] [, <keylist>] )
```

Description

All arguments to the instantiator are optional. All arguments except the first must be given as keyword arguments. Even the first may be given as a keyword argument, using either `filename` or `display` as the keyword. The `<display>` argument tells where (i. e., on what device) you wish the PyGist Xwindow to appear. It should be given in standard IP format, as a quoted character string, e. g. `"al-laria.llnl.gov:0.0"` or `"128.112.45.118:0.0"`. If you omit this argument, or supply a `" "` (blank), then PyGist will attempt to read your `DISPLAY` environment variable, and if it has a value, will open a window where it specifies. If your `DISPLAY` variable is not defined, PyGist gives up in despair. If you wish to write only to a file and not have a display Xwindow, you may set this keyword to `""`, `"none"`, or `None`.

The Plotter instantiator accepts the following keywords:

`n, dpi, wait, private, hcp, dump, legends, style`

Keyword Arguments

The following keywords are allowed:

`n (default 0)`: the number of the graphics window (0 to 7 are allowed). each plotter object corresponds to a separate window.
`dpi (default 100 for 2-D, 75 for 3-D)`:the size of the window wanted. 100 and 75 are allowed; 100 is the larger size.
`wait (1)`: used to make sure everything is plotted before changing frames.
`private (0)`: use a common colormap.
`hcp:` if not present, use default hardcopy file used by all windows. If present, names a file unique to this window. Use `".ps"` suffix for a postscript file or `".cgm"` suffix for a cgm file.
`dump (0)`: if 1, dumps the color palette at the beginning of each page of hardcopy output, otherwise converts to grey scale.
`legends (0)`: controls whether (1) or not (0) curve legends are dumped to the hardcopy.

`style ("work.gs")`: name of a Gist style sheet to use for this window.

3.2 Instantiation of a Narcisse `Plotter` Object

Calling Sequence

```
import NarPlotter
pl2 = NarPlotter.Plotter ( [filename])
```

Description

Remember that Narcisse is different from Gist, in that a PyGist `Plotter`, when instantiated, creates its own Xwindow. A PyNarcisse `Plotter` expects to find a Narcisse process already running. The `filename` argument and/or certain environment variables are used to tell PyNarcisse where to look for this process. To refresh your mind on how to fire up a Narcisse process, Section 1.2 on page 2.

Use the `filename` variable to specify the location of the Narcisse process to which you wish to connect. This is absolutely essential if you wish to connect to a Narcisse process on a remote machine. The form of a filename is:

```
"machine+port_serveur++user@ie.32"
```

where machine is the IP address, e. g., `allaria.llnl.gov:0.0`; `port_serveur` is the server port number shown on the Narcisse GUI; and `user` is your userid.

If `filename` is blank or missing, then PyNarcisse looks first for the environment variable `DEST_SP3`. If it exists, then it should specify the destination in the same format as given above, and this specifies the Narcisse to which connection will be made. If it does not exist, then PyNarcisse next looks at the environment variable `PORT_SERVEUR`. If this variable is defined, then it looks on the current machine for a Narcisse with that port number. If PyNarcisse is unsuccessful in finding a Narcisse process to which to connect, it goes into a tight loop, repeatedly printing an error message.

CHAPTER 4: Examples

General comments that apply to this whole chapter.

4.1 First Section Title

Calling Sequence

```
     general call example
```

Description

Insert description of all arguments, with xrefs to other related information, etc.

More information.

Optional Attributes

The following optional attributes can be specified with this command. (or some such)

```
     list of attributes
```

Additional information as needed.

Examples

Description of example(s).

```
     first line code
     middle lines of code
     last line of code
```

Whatever.

4.1.1 First subsection

Discuss material that is sufficiently voluminous and self-contained to warrant a subsection.

4.1.2 Second subsection

Next subsection.

Note: May want to force page break before next Section.

4.2 Second Section Title

Calling Sequence

```
     general call example
```

Description

Insert description of all arguments, with xrefs to other related information, etc.

More information.

Optional Attributes

The following optional attributes can be specified with this command. (or some such)

```
     list of attributes
```

Additional information as needed.

Examples

Description of example(s).

```
     first line code
     middle lines of code
     last line of code
```

Whatever.

4.3 Third Section Title

Calling Sequence

```
     general call example
```

Description

Insert description of all arguments, with xrefs to other related information, etc.

More information.

Optional Attributes

The following optional attributes can be specified with this command.   (or some such)

```
     list of attributes
```

Additional information as needed.

Examples

Description of example(s).

```
     first line code
     middle lines of code
     last line of code
```

Whatever.

Index