Chiplotle Documentation

Release 0.4.2

Víctor Adán, Douglas Repetto

Oct 18, 2019

Contents

CHAPTER 1

What is Chiplotle?

Chiplotle is an HPGL plotter driver that implements and extends the HPGL (Hewlett-Packard Graphics Language) plotter control language. It provides direct control of your HPGL-aware hardware via a standard usb<->serial port interface. *Chiplotle* is also a general purpose vector drawing library with functions for creating and transforming shapes, which can then be sent directly to your HPGL plotter for printing.

Chiplotle is written and maintained by Víctor Adán and Douglas Repetto.

Find all there is to know about Chiplotle at: http://www.chiplotle.org

CHAPTER 2

Installing

2.1 Dependencies

To install Chiplotle you must already have Python 2.5 or Python 2.6 installed in your computer.

Chiplotle has the following dependencies:

NumPy (for number crunching): http://numpy.scipy.org/

PySerial (for serial communication): http://pyserial.sourceforge.net/

hp2xx (for image export / hpgl preview): http://www.gnu.org/software/hp2xx/hp2xx.html

2.2 Installing the official release

If you have pip just:

```
pip -U chiplotle
```

If you don't have *pip*, follow these steps:

1. Download the latest Chiplotle release from http://pypi.python.org/pypi/Chiplotle.
2. Untar the downloaded file (e.g. *tar xzvf Chiplotle-NNN.tar.gz*, where *NNN* is the version number of the latest release).
3. Change into the directory created in step 2 (e.g. *cd Chiplotle-NNN*).
4. If you're using Linux, Mac OS X or some other flavor of Unix, enter the command:

   ```
   sudo python setup.py install
   ```

   at the shell prompt. If you're using Windows, start up a command shell with administrator privileges and run the command `setup.py install`.

These commands will install Chiplotle in your Python installation's *site-packages* directory. Note that this requires a working internet connection.

2.3 Installing the development version

If you'd like to be at the cutting edge of the Chiplotle development use the following alternative:

1. Install Subversion if you don't have it already installed (enter `svn help` to verify this).
2. Check out Chiplotle's *trunk* development like so:

```
svn co svn://music.columbia.edu/chiplotle/trunk/ chiplotle-trunk
```

3. Make the Python interpreter aware of Chiplotle. There are two ways to do this:

   a. Make a symlink in your Python *site-packages* directory pointing to the chiplotle-trunk directory previously checked out via Subversion:

```
ln -s 'pwd'/chiplotle-trunk/chiplotle SITE-PACKAGES-DIR/chiplotle
```

   where SITE-PACKAGES-DIR is the Python *site-packages* directory. In Linux this is usually in `/usr/lib/Python2.x/site-packages`.

   b. Alternatively you can include the *chiplotle-trunk* directory in your `PYTHONPATH` environment variable.

4. Have your `PATH` environment variable point to the *scripts* folder. This will allow you to run Chiplotle and all its accompanying scripts from anywhere in your system.
5. Download and install NumPy and PySerial.

CHAPTER 3

Configuration

Chiplotle has a dedicated folder, `$HOME/.chiplotle`, which houses the `config.py` configuration file. This file is used to set a variety of configuration and preference values. The file is executed as a Python module, so its syntax must conform with standard Python syntax. Edit this file to change your Chiplotle default settings.

At present, the `config.py` file allows for the setting of the following parameters:

- *Serial port to plotter map*: This is a dictionary that maps hardware devices (e.g. 'dev/ttyS0') to software Chiplotle plotters (e.g., 'DXY-1300'). Set it to *None* if you want Chiplotle to dynamically find the plotters connected to your computer. This is the default, and can be convenient when your setup changes frequently. For a fixed setup set this to a dictionary mapping serial ports to plotters.
- *Serial connection parameters*: These are the serial communication parameters that *all* your connected plotters must be set to. Chiplotle sets these to common standard values. Make sure your hardware plotter is set to these values, or change them to match those of your hardware.
- *Maximum plotter "wait for response" time*: Plotters take some time to respond to your queries. Set this to the maximum time you expect your plotter to wait for a response.
- *Verbosity*: Chiplotle keeps a log of important events taking place during a Chiplotle session in the `session.log` file under your `.chiplotle` directory. Set verbosity to `True` if you want all the log information to be displayed in the screen during execution.

CHAPTER 4

API

4.1 Chiplotle Geometry / Shapes

4.1.1 Shapes

chiplotle.geometry.shapes.**annotation**(*shape*)
Returns informative shape annotations. Good for debugging and general info displaying.

Annotations: max (x, y) coordinate min (x, y) coordinate width height center centroid

chiplotle.geometry.shapes.**arc_circle**(*radius*, *start_angle*, *end_angle*, *segments=100*, *segmentation_mode='2PI'*)
Constructs an arc from a circle with the given radius, and number of segments. Arc goes from start_angle to end_angle, both of which are in radians.

- ***segmentation_mode*** ['2PI' or 'arc'. The first segments] the whole circle into the given number of segments, the second segments the arc.

chiplotle.geometry.shapes.**arc_ellipse**(*width*, *height*, *start_angle*, *end_angle*, *segments=100*, *segmentation_mode='2PI'*)
Constructs an arc from an ellipse with the given width, height, and number of segments. Arc goes from start_angle to end_angle, both of which are in radians.

- ***segmentation_mode*** ['2PI' or 'arc'. The first segments] the whole ellipse into the given number of segments, the second segments the arc.

chiplotle.geometry.shapes.**arrow**(*path*, *headwidth*, *headheight*, *filled=False*)
Returns an arrow shape.

- *path* is a Path object.
- *headwidth* is the width of the arrow head.
- *headheight* is the height of the arrow head.

chiplotle.geometry.shapes.**catmull_path**(*points*, *interpolation_count=50*)
Path with Catmull-Rom spline interpolation.

chiplotle.geometry.shapes.**circle**(*radius*, *segments=36*)
Returns a circle.

chiplotle.geometry.shapes.**cross**(*width*, *height*)
Draws a cross shape.

- *width* is the length of the horizontal line.
- *height* is the length of the vertical line..

chiplotle.geometry.shapes.**donut**(*width*, *height*, *inset*, *segments=100*)
A donut (ellipse within ellipse) with a width, height, inset, segments.

segments is how many lines should be used to draw ellipse. More segments create a smoother ellipse, but will take longer to draw.

inset is the distance to inset the inner ellipse from the outer.

The donut is drawn with the current pen location as the center. offset may be used to shift this around, for example, to draw from the lower, left corner.

chiplotle.geometry.shapes.**ellipse**(*width*, *height*, *segments=36*)
Constructs an ellipse with the given width, height, and number of segments.

chiplotle.geometry.shapes.**fan**(*radius*, *start_angle*, *end_angle*, *height*, *segments=100*, *filled=False*)
A Fan is a slice of a donut seen from above (when you can see the hole in the middle).

All angles are assumed to be in radians.

chiplotle.geometry.shapes.**frame**(*width*, *height*, *inset*)
A frame (rectangle within a rectangle) with a width, height, and inset.

- *width* is the width of the frame.
- *height* is the height of the frame.
- *inset* is the distance to inset the inner rectangle from the outer.

chiplotle.geometry.shapes.**grid**(*width*, *height*, *width_divisions*, *height_divisions*)
Rectangular grid.

- *width* : `int` or `float`, width of the rectangle.
- *height* : `int` or `float`, height of the rectangle.
- *width_divisions* : `int`, number of horizontal equidistant partitions.
- *height_divisions* : `int`, number of vertical equidistant partitions.

chiplotle.geometry.shapes.**group**(*lst*)

chiplotle.geometry.shapes.**isosceles**(*width*, *height*, *filled=False*)

chiplotle.geometry.shapes.**label**(*text*, *charwidth*, *charheight*, *charspace=None*, *linespace=None*, *origin='bottom-left'*)

chiplotle.geometry.shapes.**layer**(*shapes*, *name*)

chiplotle.geometry.shapes.**line**(*startpoint*, *endpoint*)
Returns a Path with only two points. The path describes a simple straight line.

chiplotle.geometry.shapes.**line_displaced**(*start_coord*, *end_coord*, *displacements*)
Returns a Path defined as a line spanning points *start_coord* and *end_coord*, displaced by scalars *displacements*. The number of points in the path is determined by the lenght of *displacements*.

chiplotle.geometry.shapes.**lock_group**(*shapes*, *lock_transforms*)

chiplotle.geometry.shapes.**path**(*points*)
A simple path.

chiplotle.geometry.shapes.**path_linear**(*coords*, *interpolation_unit*)
Returns a path with linearly interpolated segments. Visually the result is the same as a plain path, but this is useful as an intermediate step in constructing more interesting shapes by deforming this one.

- *coords* is a CoordinateArray.
- ***interpolation_unit* is the magnitude of the path segments created.** Think of it as the sampling period in digital recording. If *interpolation_unit* > coord[i] - coord[i-1], the coord[i] - coord[i-1] segment is not further segmented.

chiplotle.geometry.shapes.**path_interpolated**(*points*, *curvature*, *interpolation_count=50*)
Returns a Path with bezier interpolation between the given *points*. The interpolation is computed so that the resulting path touches the given points.

- *points* the key points from which to interpolate.
- *curvature* the smoothness of the curve [0, 1].
- ***interpolation_count* is the number of points to add by interpolation,** per segment.

chiplotle.geometry.shapes.**path_bezier**(*control_points*, *weight=1*, *interpolation_count=50*)
Rational Bezier curve.

chiplotle.geometry.shapes.**radial_ruler**(*radius*, *start_angle*, *end_angle*, *units*, *min_tick_height*, *symmetric=True*)

chiplotle.geometry.shapes.**random_walk_cartesian**(*steps*, *step_size=500*)
Random Walk. Border is a square.

chiplotle.geometry.shapes.**random_walk_polar**(*steps*, *step_size=500*)
Random Walk. Border is a circle

chiplotle.geometry.shapes.**rectangle**(*width*, *height*)

chiplotle.geometry.shapes.**ruler**(*start_coord*, *end_coord*, *units*, *min_tick_height*, *symmetric=False*)
A measuring ruler.

- ***units* is a list of units on which to put marks, from smaller** to larger. e.g., (10, 20, 40).
- ***min_tick_height* is the height of the marks for the smallest units.** The hight of the other units are multiples of this.
- ***symmetric* set to True to draw the tick lines symmetrically around** the invisible center-line.

chiplotle.geometry.shapes.**spiral_archimedean**(*radius*, *num_turns=5*, *wrapping_constant=1*, *direction='cw'*, *segments=500*)
Constructs an Archimedean (arithmetic) spiral with the given number of turns using the specified number of points.

wrapping_constant controls how tightly the spiral is wound. Several classic spirals can be created using different wrapping_constants:

lituus: -2 hyperbolic spiral: -1 Archimedes' spiral: 1 Fermat's spiral: 2

scaler controls how large the spiral is.

The general Archimedean spiral equation is:

r = a * theta^(1/n)

where r is the radius, a is the scaler, and n is the wrapping_constant.

More info: http://mathworld.wolfram.com/ArchimedeanSpiral.html

chiplotle.geometry.shapes.**spiral_logarithmic**(*num_turns=5*, *expansion_rate=0.2*, *direction='cw'*, *segments=500*)
Constructs an logarithmic spiral with the given number of turns using the specified number of points.

expansion_rate controls how large the spiral is for a given number of turns. Very small numbers will result in tightly-wound spirals. Large numbers will give spirals with giant "tails". Typical values range from 0.1 to 0.3

The logarithmic spiral equation is:

r = e^(bt)

where r is the radius, e is e, b is the expansion rate, and t is theta

chiplotle.geometry.shapes.**star_crisscross**(*width*, *height*, *num_points=5*, *jump_size=None*, *find_valid_jump_size=True*)
Draws a star with criscrossing lines.

jump_size determines how many points to skip between connected points. an illegal jump size (one that does not result in a valid crisscross star) is ignored and replaced with a dot in the center of the star.

chiplotle.geometry.shapes.**star_outline**(*width*, *height*, *num_points=5*)
Constructs a star shape in outline.

chiplotle.geometry.shapes.**supershape**(*width*, *height*, *m*, *n1*, *n2*, *n3*, *point_count=100*, *percentage=1.0*, *a=1.0*, *b=1.0*, *travel=None*)
Supershape, generated using the superformula first proposed by Johan Gielis.

- *points_count* is the total number of points to compute.
- ***travel* is the length of the outline drawn in radians.** 3.1416 * 2 is a complete cycle.

chiplotle.geometry.shapes.**symmetric_polygon_side_length**(*count*, *length*)
Creates a symmetric polygon with *count* sides, all with the same given *length*.

chiplotle.geometry.shapes.**target**(*outer_radius*, *inner_radius*, *circles_count*, *segments=36*)
Creates *circles_count* concentric circles. Can be used to create radially filled circles.

4.1.2 Transforms

chiplotle.geometry.transforms.**TransformVisitor**(*transform*)
"Crawler" pattern encapsulation for transformations applied to _Shapes. Separates the "what it does" (action) from "how it does it" (traversal).

chiplotle.geometry.transforms.**arrange_shapes_on_path**(*shapes*, *path*)

chiplotle.geometry.transforms.**center_at**(*shape*, *coord*)
Centers shape at the given coordinate.

chiplotle.geometry.transforms.**noise**(*shape*, *value*)
Distort shape by adding noise.

- ***value* can be a scalar or a tuple (x, y) that sets the range of the** noise for the x and y coordinates.

chiplotle.geometry.transforms.**offset**(*shape*, *value*)
In place offsetting.

- *shape* is the shape to be rotated.
- *value* is the offset value. Can be a scalar or an (x, y) coordinate.

chiplotle.geometry.transforms.**perpendicular_displace**(*path*, *displacements*)
Displaces a path along its perpendiculars.

- *path* is a Path instance.
- *displacement* is a list of displacement values (scalars).

chiplotle.geometry.transforms.**perpendicular_noise**(*shape*, *value*)
Distort shape by adding noise perpendiculary to the path. This is an in-place destructive transformation; no new shapes are created.

- *shape* is the shape to be noisified.
- ***value* must be a scalar defining the range of the noise** for displacement.

chiplotle.geometry.transforms.**rotate**(*shape*, *angle*, *pivot=(0, 0)*)
In place rotation.

- *shape* is the shape to be rotated.
- *angle* is the angle (in radians) of rotation.
- *pivot* is the center of rotation. Must be a Coordinate or (x, y) pair.

chiplotle.geometry.transforms.**scale**(*shape*, *value*, *pivot=Coordinate([0, 0])*)
In place scaling.

- *shape* is the shape to be rotated.
- *value* is the scaling value. Can be a scalar or an (x, y) coordinate.
- *pivot* is the Coordinate around which the shape will be scaled.

4.2 Chiplotle-HPGL commands

class chiplotle.hpgl.commands.**AA**(*xy*, *angle*, *chordtolerance=None*)
Bases: chiplotle.hpgl.abstract.arc._Arc

Arch Absolute Draws an arc, using absolute coordinates, that starts at the current pen location and uses the specified center point.

- *xy* : `(x, y)` position pair.
- *angle* : `float` [-360 to 360]. The arch angle in degrees.
- *chordtolerance* : `float` [0.36 to 180], `None`.

class chiplotle.hpgl.commands.**AF**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Advance full page Advances roll paper one full page length and establishes the origin at the center of the new page.

class chiplotle.hpgl.commands.**AH**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Advance half page Advances roll paper one half page length and establishes the origin at the center of the new page.

class chiplotle.hpgl.commands.**AP**(*n=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Automatic Pen operations Controls automatic pen operations such as returning a pen to the carousel if it has been in the holder without drawing for a certain time.

For 7550:

bit_no	dec_val	state	meaning
0	1	1	lift pen if down too long without motion
0	0	0	do not lift pen until PU received
1	2	1	put pen away if too long without motion
1	0	0	do not put pen away until SP0 received
2	4	1	do not get new pen until drawing starts
2	0	0	get pen immediately after SP command
3	8	1	merge all pen up moves
3	0	0	do not merge all pen up moves

default is 7 on 7550 codes are 0 to 255 with default of 95 on the DraftMaster

format

class chiplotle.hpgl.commands.**AR**(*xy*, *angle*, *chordtolerance=None*)
Bases: chiplotle.hpgl.abstract.arc._Arc

Arch Relative Draws an arc, using relative coordinates, that starts at the current pen location and uses the specified center point.

- *xy* : `(x, y)` position pair.
- *angle* : `float` [-360 to 360]. The arch angle in degrees.
- *chordtolerance* : `float` [0.36 to 180], `None`.

class chiplotle.hpgl.commands.**AS**(*accel=None*, *pen=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Acceleration Select Sets pen acceleration for one or all pens. The default acceleration is suitable for all recommended pen and media combinations. Slowing the acceleration may improve line quality if you are using heavier than recommended media.

- *accel* : `int` [1 to 4] , `None`.
- *pen* : `int` [1 to 8], `None`. When `None`, accel is applied to all pens.

format

class chiplotle.hpgl.commands.**B**
Bases: chiplotle.hpgl.abstract.hpglescape._HPGLEscape

Escape output buffer space

class chiplotle.hpgl.commands.**BF**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Buffer plot.

class chiplotle.hpgl.commands.**BL**(*label=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Buffer label Stores a label in the label buffer. You can then use the output length (*OL*) instruction to determine its space requirement prior to drawing it. Or, you can use the plot buffer (*PB*) instruction to repeatedly plot this label.

format

class chiplotle.hpgl.commands.**CA**(*set=0*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Designate alternate character set Designates a character set as the alternate character set to be used in labeling instructions. Use this instruction to provide an additional character set that you can easily access in a program.

- *set* : int [-1, 0 to 59, 60, 70, 80, 99, 100, 101], default 0.

format

class chiplotle.hpgl.commands.**CC**(*angle=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Character chord angle Sets the chord angle that determines the smoothness of characters drawn when you select one of the arc-font character sets for labeling.

format

class chiplotle.hpgl.commands.**CI**(*radius*, *chordangle=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Circle Draws a circle using the specified radius and chord tolerance. If you want a filled circle, refer to the *WG* or *PM* instruction.

format

radius
The radius of the circle.

class chiplotle.hpgl.commands.**CM**(*switch=None*, *fallback=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Character selection mode Specifies mode of character set selection and usage. Use this instruction to select the alternate HP 8-bit, ISO 7-bit, or ISO 8-bit character modes.

- *switch* : int [0 to 3], default 0.
- *fallback* : int 0 or 1, default 0.

format

class chiplotle.hpgl.commands.**CP**(*spaces=None*, *lines=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Character Plot Move the pen the specified number of character plot cells from the current pen location.

format

class chiplotle.hpgl.commands.**CS**(*set=0*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Standard character set Designates a character set as the standard character set for labeling instruction. Use this instruction to change the default ANSI ASCII english set to one with characters appropriate to your application. This instruction is particularly useful if you plot most of your labels in a language other than english.

format

class chiplotle.hpgl.commands.**CT**(*type=0*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Chord tolerance Determines whether the chord tolerance parameter of the *CI*, *AA*, *AR* and *WG* instructions is interpreted as a chord angle in degrees or as a deviation distance in current units.

- *type* : `int` 0 or 1, default 0.

format

class chiplotle.hpgl.commands.**CV**(*n=None*, *inputdelay=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Curved line generator Collects coordinates (line segments) in the coordinate buffer so that they can be plotted as a group. This allows the plotter to plot in a continuous motion, rather than stopping and starting at each coordinate endpoint. As a result, curves appear smoother.

- *n* : `int` 0 or 1, default 1 (on).
- *inputdelay* : `int` [0 to 8,388,607] msec, default 100.

format

class chiplotle.hpgl.commands.**DC**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Digitizer Clear Terminates digitize mode. For example, if you are using an interrupt routine in a digitizing program to branch to another plotting function, use DC to clear the digitize mode immediately after branching.

class chiplotle.hpgl.commands.**DF**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Default Sets certain plotter functions to predefined default conditions. Use this instruction to return the plotter to a known state while maintaining the current location of P1 and P2. When you use DF at the beginning of a program, unwanted graphics parameters such as character size, slant, or scaling are not inherited from another program.

class chiplotle.hpgl.commands.**DI**(*run=None*, *rise=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Absolute direction Specifies the direction in which labels are drawn, independent of P1 and P2 settings. Use this instruction to change labeling direction when you are labeling line charts, schematic drawings, blueprints, and survey boudaries.

- *run* : `float`. cos(angle)
- *rise* : `float`. sin(angle)

format

class chiplotle.hpgl.commands.**DP**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Digitize Point Returns the X,Y coordinates of a selected point on a plot to the computer for later use. Use this instruction to input data for a graphics program or to obtain the coordinates of a point or points on plot.

class chiplotle.hpgl.commands.**DR**(*run=None*, *rise=None*)
Bases: *chiplotle.hpgl.commands.DI*

Relative Direction Specifies the direction in which labels are drawn relative to the scaling points P1 and P2. Label direction is adjusted when P1 and P2 change so that labels maintain the same

relationship to the plotted data. Use *DI* if you want label direction to be independent or P1 and P2.

- *run* : `float`. cos(angle)
- *rise* : `float`. sin(angle)

class chiplotle.hpgl.commands.**DS**(*slot=None*, *set=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Designate Character Set into Slot Designates up to four character sets to be immediately available for plotting. Used with ISO character sets and modes.

format

class chiplotle.hpgl.commands.**DT**(*terminator='x03'*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Define Label Terminator Specifies the ASCII character to be used as the label terminator. Use this instruction to define a new label terminator if your computer cannot use the default terminator (ETX, decimal code 3).

format

class chiplotle.hpgl.commands.**DV**(*vertical=0*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Direction Vertical Specifies vertical mode as the direction for subsequent labels. Use this instruction to 'stack' horizontal characters in a column. A carriage return and a line feed lace the next 'column' to the left of the previous one.

format

class chiplotle.hpgl.commands.**EA**(*xy*)
Bases: chiplotle.hpgl.abstract.positional._Positional

Edge Rectangle Absolute Defines and outlines a rectangle using absolute coordinates.

- *xy* : `(x, y)`. The absolute coordinates of the remaining corner.

class chiplotle.hpgl.commands.**EC**(*n=0*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Enable Cut Line Draws a dashed cut line between 'pages' on roll paper to indicate where to cut the paper. Used with *AF*, *AH* and *PG* instructions.

format

class chiplotle.hpgl.commands.**EP**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Edge Polygon Outlines the polygon currently stored in the polygon buffer. Use this instruction to edge polygons that you defined in polygon mode (*PM*) and with the rectangle and wedge instructions (*RA*, *RR* and *WG*).

class chiplotle.hpgl.commands.**ER**(*xy*)
Bases: chiplotle.hpgl.abstract.positional._Positional

Edge Rectangle Relative Defines and outlines a rectangle using relative coordinates.

- *xy* : `(x, y)`. The relative coordinates of the remaining corner.

class chiplotle.hpgl.commands.**ES**(*charspace=None*, *linespace=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Extra space Adjust space between characters and lines of labels without affecting character size.

- *charspace* : float, None. Spacing between characters.
- *linespace* : float, None. Spacing between lines.

Character and line spacing values add (or substract) a fraction of the standard spacing. 0 is the standard, positive values increase space and negative values reduce space. 1 doubles the standard space, 0.5 adds half the standard space, and -1 substracts the standar space, causing the characters to draw on top of each other.

format

class chiplotle.hpgl.commands.**EW**(*radius*, *startangle*, *sweepangle*, *chordangle=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Edge Wedge Outlines any wedge. Use these instructions to produce sectors of a pie chart.

- *radius* : float.
- *startangle* : float [0 - 360] degrees.
- *sweepangle* : float [0 - 360] degrees.
- *chordangle* : float [0.36 - 50] degrees.

format

class chiplotle.hpgl.commands.**ExtendedError**
Bases: chiplotle.hpgl.abstract.hpglescape._HPGLEscape

ExtendedError Get RS-232-C related error message.

error num	meaning
0	no i/o error
10	output request received while still processing previous one
11	invalid byte received after escape sequence (“ESC.”)
12	invalid byte received as part of a device control instruction
13	parameter out of range
14	too many parameters received
15	framing, parity, or overrun error
16	input buffer overflow

class chiplotle.hpgl.commands.**FP**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Fill Polygon Fills the polygon currently in the polygon buffer. Use FP to fill polygons defined in polygon mode (*PM*) and defined with the edge rectangle and wedge instructions (*EA*, *ER*, and *EW*).

class chiplotle.hpgl.commands.**FR**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Advance Frame Advances paper to the next plot frame and calculates a relative coordinate system for that frame. Use FR to do multi-frame long-axis plotting.

class chiplotle.hpgl.commands.**FS**(*force=None*, *pen=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Force Select Sets pen pressure to the paper for one or all pens. Use this instruction to optimize pen life and line quality for each pen and paper combination.

- *force* : `int` [1 to 8]
- *pen* : `int` [1 to 8]. If pen is `None` then all pens are set.

format

class chiplotle.hpgl.commands.**FT**(*type=None*, *space=None*, *angle=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Fill Type Selects the shading pattern used in polygons (*FP*), rectangles (*RA* or *RR*), or wedges (*WG*). Use this instruction to enhance plots with solid fill, parallel lines (hatching), cross-hatching, or a fill pattern you designed using the user-defined fill type (UF) instruction.

- *type* : `int` 1 or 2, Solid (space and angle ignored) 3: Hatching, 4: Cross hatching.

format

class chiplotle.hpgl.commands.**GC**(*count=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Group Count Allows you to assign an arbitrary number that will be output by the *OG* instruction. Use *GC* with the *OG* instruction to monitor the successful transfer of data blocks in spooling applications.

format

class chiplotle.hpgl.commands.**IN**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Initialize Resets most plotter functions to their default settings. Use this instruction to return the plotter to a known state and to cancel settings that may have been changed by a previous program.

class chiplotle.hpgl.commands.**IP**(*coords=None*)
Bases: chiplotle.hpgl.abstract.twopoint._TwoPoint

Input P1 and P2 Allows you to establish new or default locations for the scaling points P1 and P2. P1 and P2 are used by the scale instruction (*SC*) to establish user-unit scaling. The *IP* instruction is often used to ensure that a plot is always the same size, regardless of how P1 and P2 might have been set from the front panel or the size of media loaded in the plotter.

class chiplotle.hpgl.commands.**IV**(*slot=None*, *left=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Invoke Character Slot Invokes a character set slot into either the right or left half of the in-use code table. Primarily used with ISO modes of character selection.

format

class chiplotle.hpgl.commands.**IW**(*coords=None*)
Bases: chiplotle.hpgl.abstract.twopoint._TwoPoint

Input Window Defines a rectangular area, or window, that establishes soft-clip limits. Subsequent programmed pen motion will be restricted to this area. Use this instruction when you want to be sure that your plot falls within a specified area.

class chiplotle.hpgl.commands.**K**
Bases: chiplotle.hpgl.abstract.hpglescape._HPGLEscape

Abort command Tells the plotter to discard commands in its buffer.

class chiplotle.hpgl.commands.**KY**(*key=None*, *function=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Define Key Assigns a predefined function to one of the frontal panel function keys. Use this instruction with the WD instruction when designing interactive programs.

format

class chiplotle.hpgl.commands.**LB**(*text*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Label Plots text using the currently defined character set.

format

class chiplotle.hpgl.commands.**LO**(*origin=1*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Label Origin Positions labels relative to current pen location. Use LO to center, left justify, or right justify label. The label can be drawn above or below the current pen location and can also be offset by an amount equal to 1/2 the character's width and height.

- *origin* : int [1-9] or [11-19].

format

class chiplotle.hpgl.commands.**LT**(*pattern=None*, *length=4*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Line Type Specifies the line pattern to be used when drawing linese and nonsolid fill types. Use LT to emphasize or de-emphasize other plotter lines and shapes.

- *pattern* : int [-6 to 6]
- *length* : float [0 to 100]

0	plot point at given point.
1	. . . . .
2	
3	___ ___ ___ ___ ___
4	__.__.__.__.__.__.
5	___ _ ___ _ ___ _ ___ _
6	___ _ _ ___ _ _ ___ _ _ ___

format

class chiplotle.hpgl.commands.**NR**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Not Ready Programmatically simulates pressing VIEW. However, you cannot take the plotter out of the view state with NR instruction.

class chiplotle.hpgl.commands.**OA**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Actual Pen Status Outputs the current pen location (in plotter units) and up/down position. Use this information to position a label or figure, to determine the parameters of a window, or to determine the pen's curent location if you moved it using front-panel cursor buttons.

class chiplotle.hpgl.commands.**OC**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Commanded Pen Status Ouput the location and up/down position of the last commanded pen move instruction. Use OC to position a label or determine the parameters of an instruction that tried to move the pen beyond the limits of some window. You can also use this instruction when you want to know the pen's location in user units.

class chiplotle.hpgl.commands.**OD**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Digitized Point and Pen Status Outputs the X,Y coordinates and up/down pen position associated with the last digitized point. Use this instruction after the *DP* instruction to return the coordinates of the digitized point to your computer.

class chiplotle.hpgl.commands.**OE**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Error Output a number corresponding to the type of HP-GL error (if any) received by the plotter after the most recent *IN* or *OE* instruction. Use this instruction for debugging programs.

bit value	error no	meaning
0	0	no error
1	1	unrecognized command
2	2	wrong num of parameters
4	3	out-of-range parameter
8	4	unused
16	5	unknown character set
32	6	position overflow
64	7	unused
128	8	pinch wheels raised

Note: some error meanings change depending on the plotter!

class chiplotle.hpgl.commands.**OF**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Factors Outputs the number of plotter units per millimeter in each axis. This lets you use the plotter with sofware that needs to know the size of a plotter unit.

class chiplotle.hpgl.commands.**OG**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Group Count Outputs the data block number of the current group count and whether the escape function has been activated. Use this instruction at the end of a data block in spooling applications, where it is important to know the current data block number and whether the data block has been transferred.

class chiplotle.hpgl.commands.**OH**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Hard-Clip Limits Outputs the X,Y coordinates of the current hard-clip limits. Use this instruction to determine the plotter unit dimension of the area in which plotting can occur.

class chiplotle.hpgl.commands.**OI**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Identification Outputs the plotter's identifying model number. This information is useful in a remote operating configuration to determine which plotter model is on-line, or when software needs the plotter's model number.

class chiplotle.hpgl.commands.**OK**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Key Outputs a number that indicates which, if any, of the front-panel function keys has been pressed. use this instruction with the *WD* instruction when designing interactive programs.

class chiplotle.hpgl.commands.**OL**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Label Length Outputs information about the label contained in the label buffer.

class chiplotle.hpgl.commands.**OO**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Options Outputs eight option parameters indicating the features implemented on the plotter. Some software packages use this feature to determine which plotter capabilities exist.

class chiplotle.hpgl.commands.**OP**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output P1 and P2 Outputs the X,Y coordinates (in plotter units) of the current scaling points P1 and P2. Use this instruction to determine the numberic coordinates or P1 and P2 when they have been set manually, and to help compute the number of plotter units per user units when scaling is on. This instruction can also be used with the input window (*IW*) instruction to programmatically set the window to P1 and P2.

class chiplotle.hpgl.commands.**OS**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Status Outputs the decimal value of the status byte. Use this instruction in debugging operations and in digitizing applications.

bit value	bit position	meaning
1	0	pen down
2	1	P1 or P2 changed ("OP" clears)
4	2	digitized point ready ("OD" clears)
8	3	initialized ("OS" clears)
16	4	ready to recieve data (always 0)
32	5	There is an error ("OE" clears)
64	6	unused
128	7	unused

power-on status == 24 (bits 3 & 4 set)

class chiplotle.hpgl.commands.**OT**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Carousel Type Outputs information on the type of carousel loaded and the stalls occupied.

class chiplotle.hpgl.commands.**OW**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Output Window Outputs the X,Y coordinates of the lower-left and upper-right corners of the window area in which plotting can occur. This instruction is especially useful when the window area (defined by *IW*) extends beyond the hard-clip limits.

class chiplotle.hpgl.commands.**Off**
Bases: chiplotle.hpgl.abstract.hpglescape._HPGLEscape

Off Places the plotter in a programmed off-state.

class chiplotle.hpgl.commands.**On**
Bases: chiplotle.hpgl.abstract.hpglescape._HPGLEscape

On Places the plotter in a programmed on-state. Instructs the plotter to interpret data as HPGL and DCI instructions, rather than plotting the data stream as literal text characters.

class chiplotle.hpgl.commands.**PA**(*xy=None*)
Bases: chiplotle.hpgl.abstract.penplot._PenPlot

Plot Absolute Establishes absolute plotting and moves the pen to specified absolute coordinates using the current pen position.

class chiplotle.hpgl.commands.**PB**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Print Buffer Label Prints the contents of the label buffer.

class chiplotle.hpgl.commands.**PD**(*xy=None*)
Bases: chiplotle.hpgl.abstract.penplot._PenPlot

Pen Down Lowers the pen onto the writing surface for drawing and moves it to the coordinates/increments you specified.

- *xy* : A `list` or `tuple` of x, y positions of the form `(x1, y2, x2, y2, x3, y3, ..., xn, yn)`.

class chiplotle.hpgl.commands.**PG**(*n=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Page Feed Advances roll paper one page length and establishes the plotter-unit origin at the center of the new page.

format

class chiplotle.hpgl.commands.**PM**(*n=0*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Polygon Mode Enter polygon mode for defining shapes such as block letters, logos, surface charts, or any unique or intricate area for subsequent filling and/or edging. Fill polygons using the fill polygon (FP) instruction and/or outline them using the edge polygon (EP) instruction.

format

class chiplotle.hpgl.commands.**PR**(*xy=None*)
Bases: chiplotle.hpgl.abstract.penplot._PenPlot

Plot Relative Establishes relative plotting and moves the pen (using the current position) to the specified points, each successive move relative to the last current pen location.

class chiplotle.hpgl.commands.**PS**(*length=None*, *width=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Page Size Changes the size of the hard clip limits.

format

class chiplotle.hpgl.commands.**PT**(*thickness=0.3*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Pen Thickness Determines the spacing between the parallel lines in solid fill patterns, according to the pen tip thickness.

- *thickness* : `float` [0.1 to 5] mm, default is 0.3mm.

format

class chiplotle.hpgl.commands.**PU**(*xy=None*)
Bases: chiplotle.hpgl.abstract.penplot._PenPlot

Pen Up Raises the pen from the plotting surface. Use this instruction to prevent stray lines from being drawn.

- *xy* : A `list` or `tuple` of x, y positions of the form `(x1, y2, x2, y2, x3, y3, ..., xn, yn)`.

class chiplotle.hpgl.commands.**RA**(*xy*)
Bases: chiplotle.hpgl.abstract.positional._Positional

Filled Rectangle Absolute Defines and fills a rectangle using absolute coordinates.

- *xy* : `(x, y)` tuple. The absolute coordinates of the remaining corner.

class chiplotle.hpgl.commands.**RO**(*angle=0*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Rotate coordinate system

format

class chiplotle.hpgl.commands.**RP**(*n=1*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Replot

format

class chiplotle.hpgl.commands.**RR**(*xy*)
Bases: chiplotle.hpgl.abstract.positional._Positional

Filled Rectangle Relative Defines and fills a rectangle using relative coordinates.

- *xy* : `(x, y)` tuple. The relative coordinates of the remaining corner.

class chiplotle.hpgl.commands.**SA**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Select alternate character set

class chiplotle.hpgl.commands.**SC**(*coords=None*)
Bases: chiplotle.hpgl.abstract.twopoint._TwoPoint

Scale Establishes a user-unit coordinate system by mapping user-defined values onto the scaling points P1 and P2. Thus, you can plot in units convenient to your application. In addition, you can use this instruction to establish automatic isotropic scaling or to relocate the origin and set a specific ratio of plotter units to user units.

Note: DraftMaster also has a more complex version of 'SC' that is not implemented yet.

---

class chiplotle.hpgl.commands.**SI**(*width=None*, *height=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Absolute character size Specifies the size of labeling characters in centimeters. Use this instruction to establish character sizing that is not dependent on the settings of P1 and P2.

- *width* : float [-110 to 110] cm, excluding 0.
- *height* : float [-110 to 110] cm, excluding 0.

format

class chiplotle.hpgl.commands.**SL**(*tan=0*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Character Slant Argument is tan of desired angle.

format

class chiplotle.hpgl.commands.**SM**(*char=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Symbol Mode Plots the char at each plotted point. char can be any printing ascii char, except ';' Calling without an argument cancels symbol mode.

format

class chiplotle.hpgl.commands.**SP**(*pen=0*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Select Pen

format

class chiplotle.hpgl.commands.**SR**(*width=None*, *height=None*)
Bases: *chiplotle.hpgl.commands.SI*

Relative character size Specifies the relative size of characters as a percentage of the distance between P1 and P2. Use this instruction to establish relative character sizes so that if the P1/P2 distance changes, the character sizes adjust to occupy the same relative ammount of space.

- *width* : float [-100 to 100] percent, excluding 0.
- *height* : float [-100 to 100] percent, excluding 0.

class chiplotle.hpgl.commands.**SS**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Select standard character set

class chiplotle.hpgl.commands.**SetHandshakeMode**(*mode=None*)
Bases: chiplotle.hpgl.abstract.hpglescape._HPGLEscape

Set Handshake Mode Set one of three standard handshakes.

0 (none) 1 (Xon-Xoff) 2 (ENQ-ACK) 3 (hardwire)

format

mode

class chiplotle.hpgl.commands.**TL**(*tp=0.5*, *tn=0.5*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Length of ticks drawn with the XT and YT instructions.

- ***tp*** [percentage of (P2y - P1y) for XT or (P2x - P1x) for YT. Denotes portion above X-axis or to the right of the Y-axis when] difference is positive.
- *tn* : same as *tp* except denotes portion below the X-axis and to the left of the Y-axis. 0.5 is default for both.

format

class chiplotle.hpgl.commands.**VS**(*vel=None*, *pen=None*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Pen Velocity Set's pen velocity.

- *vel* : `float` [0.0 to 127.9999] (depends on plotter), `None`.
- *pen* : `int` [1 to 8].

format

class chiplotle.hpgl.commands.**WD**(*text*)
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Write to display

format

class chiplotle.hpgl.commands.**WG**(*radius*, *startangle*, *sweepangle*, *chordangle=None*)
Bases: *chiplotle.hpgl.commands.EW*

Filled wedge

class chiplotle.hpgl.commands.**XT**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

X tick

class chiplotle.hpgl.commands.**YT**
Bases: chiplotle.hpgl.abstract.hpglprimitive._HPGLPrimitive

Y tick

4.3 Chiplotle Known Plotters

class chiplotle.plotters.**DPX2000**(*ser*, ***kwargs*)
Bases: chiplotle.plotters.drawingplotter._DrawingPlotter

class chiplotle.plotters.**DPX3300**(*ser*, ***kwargs*)
Bases: chiplotle.plotters.drawingplotter._DrawingPlotter

class chiplotle.plotters.**DXY1300**(*ser*, ***kwargs*)
Bases: chiplotle.plotters.drawingplotter._DrawingPlotter

class chiplotle.plotters.**DXY880**(*ser*, ***kwargs*)
Bases: chiplotle.plotters.drawingplotter._DrawingPlotter

class chiplotle.plotters.**HP7475A**(*ser*, ***kwargs*)
Bases: chiplotle.plotters.drawingplotter._DrawingPlotter

class chiplotle.plotters.**HP7550A**(*ser*, ***kwargs*)
Bases: chiplotle.plotters.drawingplotter._DrawingPlotter

class chiplotle.plotters.**HP7575A**(*ser*, ***kwargs*)
Bases: chiplotle.plotters.hp7576a.HP7576A

class chiplotle.plotters.**HP7576A**(*ser*, ***kwargs*)
Bases: chiplotle.plotters.drawingplotter._DrawingPlotter

class chiplotle.plotters.**HP7585B**(*ser*, ***kwargs*)
Bases: chiplotle.plotters.drawingplotter._DrawingPlotter

Use with Houston Instruments DMP-60.

class chiplotle.plotters.**HP7595A**(*ser*, ***kwargs*)
Bases: chiplotle.plotters.drawingplotter._DrawingPlotter

class chiplotle.plotters.**HP7596A**(*ser*, ***kwargs*)
Bases: chiplotle.plotters.drawingplotter._DrawingPlotter

class chiplotle.plotters.**Plotter**(*ser*, ***kwargs*)
Bases: chiplotle.plotters.drawingplotter._DrawingPlotter

actual_position
Output the actual position of the plotter pen. Returns a tuple (Coordinate(x, y), pen status)

advance_frame()

advance_full_page()

advance_half_page()

allowedHPGLCommands

carousel_type

clear()
Tells the virtual serial port to forget its stored commands. Used to "erase" the drawing on the virtual plotter.

clear_digitizer()

commanded_position
Output the commanded position of the plotter pen. Returns a tuple [Coordinate(x, y), pen status]

digitize_point()

digitized_point
Returns last digitized point. Returns a tuple [Coordinate(x, y), pen status]

enable_cut_line(*n*)

escape_plotter_on()

format
This lets us pass the VirtualPlotter directly to io.view() Returns None if called on a plotter with a real serial port.

get_allowedHPGLCommands()

goto(**args*)
Alias for PA( ) with only one point. Pass in either an x, y pair: goto(100, 100) or a tuple pair: goto((x, y)) or a Coordinate: goto(Coordinate(100,100))

goto_bottom_left()

goto_bottom_right()

goto_center()

goto_origin()

goto_top_left()

goto_top_right()

id
Get id of plotter. Returns a string.

initialize_plotter()

label_length

margins
Read-only reference to MarginsInterface.

nudge(*x*, *y*)

options

output_error

output_key

output_p1p2
Returns the current settings for P1, P2. Returns two Coordinates

page_feed(*n=None*)

pen_down(*coords=None*)
Pen Down.

pen_up(*coords=None*)
Pen Up.

replot(*n=1*)

rotate(*angle=0*)

scale(*xMin*, *xMax*, *yMin*, *yMax*)

select_pen(*penNum=0*)

set_allowedHPGLCommands(*allowed_hpgl_commands*)

set_origin_bottom_left()
Set origin to bottom, left

set_origin_bottom_right()
Set origin to bottom, right

set_origin_center()
Set origin to center, center

set_origin_current_location()
Set origin to current location

set_origin_to_point(*point*)
Set origin to given point [x, y]

set_origin_top_left()
Set origin to upper, left

set_origin_top_right()
Set origin to top, right

set_plot_window(*left_bottom*, *right_top*)
Programatically set new margins for the plotting window. Arguments must be two tuple pairs (x, y) or two Coordinates.

status

write(*data*)
Public access for writing to serial port. data can be an iterator, a string or an _HPGL.

write_file(*filename*)
Sends the HPGL content of the given *filename* to the plotter.

chiplotle.plotters.**baseplotter**
alias of *chiplotle.plotters.baseplotter*

chiplotle.plotters.**dpx2000**
alias of *chiplotle.plotters.dpx2000*

chiplotle.plotters.**dpx3300**
alias of *chiplotle.plotters.dpx3300*

chiplotle.plotters.**drawingplotter**
alias of *chiplotle.plotters.drawingplotter*

chiplotle.plotters.**dxy1300**
alias of *chiplotle.plotters.dxy1300*

chiplotle.plotters.**dxy880**
alias of *chiplotle.plotters.dxy880*

chiplotle.plotters.**hp7475a**
alias of *chiplotle.plotters.hp7475a*

chiplotle.plotters.**hp7550a**
alias of *chiplotle.plotters.hp7550a*

chiplotle.plotters.**hp7575a**
alias of *chiplotle.plotters.hp7575a*

chiplotle.plotters.**hp7576a**
alias of *chiplotle.plotters.hp7576a*

chiplotle.plotters.**hp7585b**
alias of *chiplotle.plotters.hp7585b*

chiplotle.plotters.**hp7595a**
alias of *chiplotle.plotters.hp7595a*

chiplotle.plotters.**hp7596a**
alias of *chiplotle.plotters.hp7596a*

chiplotle.plotters.**margins**
alias of *chiplotle.plotters.margins*

chiplotle.plotters.**plotter**
alias of *chiplotle.plotters.plotter*

4.4 Chiplotle Tools

4.4.1 HPGL Tools

chiplotle.tools.hpgltools.**convert_coordinates_to_hpgl_absolute_path**(*coords*)
Converts an iterator of lists of coordinates e.g., [<x1, y1>, <x2, y2>, <x3, y3>, . . .] into a list of PA, PD and PU HPGL commands.

chiplotle.tools.hpgltools.**convert_relatives_to_absolutes**(*lst*)

chiplotle.tools.hpgltools.**get_all_coordinates**(*arg*)
Returns all absolute coordinates for a given list of Chiplotle-HPGL commands.

Example:

```
>>> t = [PA((1, 2)), PR((1, 1)), ER((1, 1)), CI(100)]
>>> c = hpgltools.get_all_coordinates(t)
>>> print c
[CP(1, 2), CP(2, 3), CP(3, 4)]
```

chiplotle.tools.hpgltools.**get_bounding_box**(*arg*)
Returns the pair of coordinate pairs outlining the bounding box of the given HPGL drawing.

chiplotle.tools.hpgltools.**get_centroid**(*arg*)
Returns the centroid of the given Chiplotle-HPGL shapes.

chiplotle.tools.hpgltools.**inflate_hpgl_string**(*string*, *filter_commands=None*)
Reads a string or bytes and "inflates" it by creating Chiplotle-HPGL class instances of the found HPGL commands.

Example:

```
chiplotle> sp = inflate_hpgl_string('SP1;')
chiplotle> sp
[SP(1)]
```

Example:

```
chiplotle> move = inflate_hpgl_string('IN;SP1;PA10,10;', ['IN'])
chiplotle> move
[SP(1), PA((10, 10))]
```

chiplotle.tools.hpgltools.**inflate_hpgl_string_command**(*cmd_string*)
Converts a string representing a single HPGL command into a Chiplotle HPGL instance. e.g., 'PD1,2,3,4' –> PD((1,2,3,4)).

chiplotle.tools.hpgltools.**is_primitive_absolute**(*command*)
Returns True of *command* is a primitive HPGL with absolute position, False if *command* is a non-absolute position primitive HPGL. Otherwise the function raises a TypeError exception.

chiplotle.tools.hpgltools.**parse_hpgl_string**(*arg*)
The function takes a string *arg* of HPGL commands, parses them (separates them) and returns them in a list.

chiplotle.tools.hpgltools.**pens_updown_to_papr**(*lst*)
Converts all PU((x1, y1, x2, y2) and PD(x1, y1, x2, y2) found in *lst* into (PU( ), PA(x1, y1, x2, y2)) pair sequences. The function removes the coordinates from PU and PD and places them in PR or PA, whatever was last found in lst prior to a PU or PD.

chiplotle.tools.hpgltools.**pr_to_pa**(*arg*, *starting_position=None*)
Converts a given PR into PA starting at *starting_position*.

- *arg* is a PR object.
- *starting_position* is a coordinate pair. Default is (0, 0).

chiplotle.tools.hpgltools.**relativize**(*data*)
Converts all absolute coordinate commands (PA, RA, EA, AA) into relative commands (PR, RR, ER, AR), so that everything has in realtive coordinate values.

chiplotle.tools.hpgltools.**rotate_hpglprimitives**(*arg*, *angle*)

chiplotle.tools.hpgltools.**scale**(*obj*, *val*)

chiplotle.tools.hpgltools.**transpose**(*arg*, *val*)

4.4.2 Input-output tools

chiplotle.tools.io.**export**(*expr*, *filename*, *fmt='eps'*)

Export Chiplotle-HPGL objects to an image file format via hp2xx.

- ***expr* can be an iterable (e.g., list) of Chiplotle-HPGL objects or a** single Chiplotle-HPGL object.
- *filename* the file name, including path but without extension.
- ***fmt* is a string describing the format of the file to which the** Chiplotle-HPGL objects will be exported. Default is 'eps'. Valid formats are: jpg, png, tiff and many others. Please see the hp2xx documentation for details.

Note: You must have hp2xx installed before you can export Chiplote-HPGL objects to image files.

---

chiplotle.tools.io.**import_hpgl_file**(*filename*, *filter_commands=None*)

Reads a text HPGL file and "inflates" it by creating Chiplotle-HPGL class instances of the found HPGL commands.

Example:

```
chiplotle> square = import_hpgl_file('examples/square.hpgl')
chiplotle> square
[SP(pen=1), PU(xy=[ 100.  100.]), PD(xy=[ 200.  100.]),
PD(xy=[ 200.  200.]), PD(xy=[ 100.  200.]),
PD(xy=[ 100.  100.]), SP(pen=0)]
```

chiplotle.tools.io.**save_hpgl**(*expr*, *filename*)

Save text HPGL from Chiplotle-HPGL.

- *expr* can be an iterable (e.g., list) or a Chiplotle-HPGL object.
- *filename* the full file name, including path and extension (usually .hpgl or .plt)

chiplotle.tools.io.**view**(*expr*, *fmt='eps'*)

Displays Chiplotle-HPGL objects for prevewing.

- *expr* can be an iterable (e.g., list) or a Chiplotle-HPGL object.
- ***fmt* is the file format to which the given *expr* will be** converted for viewing. The default is 'eps'.

4.4.3 Math tools

chiplotle.tools.mathtools.**bezier_interpolation**(*control_points*, *points_to_compute*, *weight*)

Computes Bezier interpolations from given *control_points*. This uses the generalized formula for bezier curves: http://en.wikipedia.org/wiki/B%C3%A9zier_curve#Generalization

- *control_points* : A list of (x, y) control points.
- *points_to_compute*: An int of the number of points to compute.
- *weight*: A list of weights for control points.

chiplotle.tools.mathtools.**catmull_interpolation**(*control_points*, *points_to_compute*)
Computes Catmull-Rom interpolations from given *control_points*. first and last point are not on the curve, but define initial and final tangent - *control_points* : A list of (x, y) control points. - *points_to_compute*: An int of the number of points to compute.

chiplotle.tools.mathtools.**cumsum**(*lst*)
Returns the cumulative sum of the values in *lst*.

chiplotle.tools.mathtools.**difference**(*seq*)
Returns the difference between consecutive elements in *seq*. i.e., first derivative.

chiplotle.tools.mathtools.**factors**(*n*)
Return factors of positive *n* in increasing order:

```
>>> mathtools.factors(84)
[1, 2, 2, 3, 7]
```

```
>>> for n in range(10, 20):
...   print n, mathtools.factors(n)
...
10 [1, 2, 5]
11 [1, 11]
12 [1, 2, 2, 3]
13 [1, 13]
14 [1, 2, 7]
15 [1, 3, 5]
16 [1, 2, 2, 2, 2]
17 [1, 17]
18 [1, 2, 3, 3]
19 [1, 19]
```

chiplotle.tools.mathtools.**interpolate_cosine**(*y1*, *y2*, *mu*)
Cosine interpolate *y1* and *y2* with *mu* normalized `[0, 1]`.

Example:

```
>>> mathtools.interpolate_cosine(0, 1, 0.5)
0.49999999999999994
```

chiplotle.tools.mathtools.**interpolate_exponential**(*y1*, *y2*, *mu*, *exp=1*)
Exponential interpolate *y1* and *y2* with *mu* normalized `[0, 1]`.

Example:

```
>>> mathtools.interpolate_exponential(0, 1, 0.5, 4)
0.0625
```

Set *exp* to the exponent of interpolation.

chiplotle.tools.mathtools.**interpolate_linear**(*y1*, *y2*, *mu*)
Linear interpolate *y1* and *y2* with *mu* normalized `[0, 1]`.

Example:

```
>>> mathtools.interpolate_linear(0, 1, 0.5)
0.5
```

chiplotle.tools.mathtools.**lcm**(*a*, *b*)
returns the lowest common multiple of a & b

chiplotle.tools.mathtools.**pascal_row**(*n*)
Returns the nth row of Pascal's Triangle.

chiplotle.tools.mathtools.**polar_to_xy**(*args*)
Converts polar (r, A) to Cartesian (x y) coordinates, where r is the radius and A is the angle in radians.

chiplotle.tools.mathtools.**rotate_2d**(*xy*, *angle*, *pivot=(0, 0)*)
2D rotation.

- *xy* is an (x, y) coordinate pair or a list of coordinate pairs.
- *angle* is the angle of rotation in radians.
- *pivot* the point around which to rotate *xy*.

Returns a Coordinate or a CoordinateArray.

chiplotle.tools.mathtools.**rotate_3d**(*xyz*, *xyzrot*)
3D rotation.

- *xyz* is a triple (x, y, z) of coordinates.
- *xyzrot* is a triple (xr, yr, zr) of angles of roation.

chiplotle.tools.mathtools.**superformula**(*a*, *b*, *m*, *n1*, *n2*, *n3*, *phi*)
Computes the position of the point on a superformula curve. Superformula has first been proposed by Johan Gielis and is a generalization of superellipse. see: http://en.wikipedia.org/wiki/Superformula

chiplotle.tools.mathtools.**xy_to_polar**(*args*)
Converts cartesian to polar coordinates. Argument may be two coordinates x, y, a tuple (x, y), or a Coordinate(x, y).

Returns an (r, a) tuple, where *r* is the magnitude, *a* is the angle in radians.

CHAPTER 5

Tutorial

5.1 Quick Start Tutorial

There are three main ways of using Chiplotle:

1. *Command line*: an interactive mode where you type in commands one at a time and the plotter executes them immediately.
2. *Python script*: you create an Python script that generates the desired Chiplotle commands and sends them to the plotter.
3. *HPGL pipeline*: you send pre-existing HPGL commands from a file to the plotter using Chiplotle as a simple serial interface.

5.1.1 Running Chiplotle from the command line

An easy way to get a feel for Chiplotle is to directly enter some commands via the command line. Start Chiplotle by typing `chiplotle` from the terminal:

```
$ chiplotle
```

Note: If you've installed Chiplotle using git, make sure the `chiplotle/scripts` directory is in your `PATH` variable so that your system knows about Chiplotle's scripts.

The `chiplotle` script is a simple convenience script that does two things:

1. It loads the Python interpreter.
2. It finds all the plotters connected to your computer, instantiates their corresponding software interfaces and assigns these to the `plts` list variable. The first plotter found is also assigned to the `plotter` variable for convenience. i.e., if you only have one plotter you can call `plotter` instead of `plts[0]`.

Note: If your plotter type is unrecognized, the function will display a list of plotters that Chiplotle knows about for you to choose from. Select the one that most closely matches the plotter ID identifying your hardware. If there is no match, you can use the generic Plotter, although you should be aware that some HPGL commands may not work with your hardware.

Chiplotle will then print out some basic information about your plotter(s), including drawing area and memory. Now it's time to plot!

5.1.2 HPGL

Let's pick up a pen. In HPGL, the command to pick up a pen is `SP`, which stands for "select pen". In Chiplotle we instantiate an instance of that command like so:

```
chiplotle> hpgl.SP(1)
```

Many HPGL commands take one or more parameters; if a command takes parameters you put them inside a set of `()` after the command name. `SP` takes a pen number, so to select pen 1 we pass `1` as a parameter as we did above.

Note: Remember that you can always refer to the *Chiplotle API* for information on the HPGL commands and its required parameters. You can also use the Python `help( )` function to find information about any Python object. Thus, to learn what parameters you need to pass to a command you can type:

```
chiplotle> help(hpgl.SP)
```

To pass the command to the plotter, you use `plotter.write( )`. So:

```
chiplotle> plotter.write(hpgl.SP(1))
```

Your plotter should pick up pen one. Some common commands, like `SP`, can be directly sent from the plotter. i.e., the plotter has methods equivalent to some of the HPGL commands. Such is the case of `SP`:

```
chiplotle> plotter.select_pen(1)
```

This effectively instantiates a `SP` command instance and send the command to the plotter.

Now let's move the pen. To move the pen while it is in the up position, you use `hpgl.PU([(x,y)])`, and to move the pen while it's down you use `hpgl.PD([(x,y)])`. *x* and *y* are the coordinates you want to move the pen to. If you want to do a `PU` or `PD` without moving, just pass a blank set of coordinates. So to draw a square you might do something like:

```
chiplotle> plotter.select_pen(1)
chiplotle> plotter.write(hpgl.PU([(100,100)]))
chiplotle> plotter.write(hpgl.PD([(200,100), (200,200), (100,200), (100,100)]))
chiplotle> plotter.write(hpgl.PU([]))
```

There are plotter shortcuts for `PU` and `PD`:

```
chiplotle> plotter.pen_up([(100,100)])
chiplotle> plotter.pen_down([(100,100)])
```

To replace the pen and have a look at your magnificent square, you select pen zero:

```
chiplotle> plotter.select_pen(0)
```

5.1.3 Running Chiplotle from a Python script

Simple drawings can be done by hand from the command line, but you'll quickly find that it's much easier to put your commands into a Python script so that you can edit them, rerun them, etc. And of course since you're writing in Python, you can use all the features of the language in addition to the Chiplotle commands.

It's very easy to create a Python script with Chiplotle commands. The first thing to do is to import the Chiplotle library. So open a new text file and type:

```
from chiplotle import *
```

Next you want your script to run the Chiplotle setup routine and import the plotter definitions:

```
plts = instantiate_plotters( )
```

If you only have one plotter (or only care to use one plotter) you can get the first and only plotter in the list returned by `instantiate_plotters( )`, like so:

```
  plotter = instantiate_plotters( )[0]

Now you can simply enter a series of Chiplote commands::

  plotter.select_pen(1)
  plotter.write(hpgl.PU([(100,100)])
  plotter.write(hpgl.PD([(200,100), (200,200), (100,200), (100,100)]))
  plotter.select_pen(0)
```

and save your script as a .py file (see examples/square.py for an example). To use your new program just run it as you would any Python script:

```
$ python square.py
```

A slightly more sophisticated Python script that draws a random zigzag:

```
from chiplotle import *
import random

plotter = instantiate_plotters( )[0]

plotter.select_pen(1)

coords = [(x, random.randint(0, 1000)) for x in range(0, 1000, 10)]
plotter.write(hpgl.PD(coords))

plotter.select_pen(0)
```

See the .py files in the examples and scripts folders for some more elaborate examples.

5.1.4 HPGL pipeline

If you already have a file containing HPGL commands (from a CNC design package, old design data, etc), you can use Chiplotle to send those commands to your plotter. Your HPGL file will be a text file with commands like:

```
SP1;
PU100,100;
PD200,100;
PD200,200;
PD100,100;
PD100,100;
SP0;
```

To plot the file while running Chiplotle you can use the plotter's own `write_file(filename)` method:

```
chiplotle> plotter.write_file('my_file.hpgl')
```

You can also plot the file from the command line without first running Chiplotle by using the `plot_hpgl_file.py` script found in the scripts folder:

```
$ plot_hpgl_file.py my_file.hpgl
```

Chiplotle will take care of all buffering and timing issues, so even large HPGL files should plot reliably. See `examples/media/square.hpgl` for a sample HPGL file.

5.2 Shapes

Chiplotle comes with a variety of *primitive* shape constructors. A circle constructor takes at least a *radius* parameter:

```
c = shapes.circle(1000)
```

A rectangle constructor takes two parameters, a *width* and a *height*:

```
r = shapes.rectangle(500, 1000)
```

Note: The shapes module is loaded after running `chiplotle` from the terminal prompt, or calling `from chiplotle import *` in the Python interpreter.

There are many primitive shape constructors in Chiplotle. To see all available shape constructors call `dir()` on the shapes module:

```
dir(shapes)
```

To get information about a function call `help()` on it:

```
help(shapes.circle)
```

5.2.1 Properties

Shapes have properties, like center, centroid, width, height, etc.:

```
>>> c = shapes.circle(1000)
>>> c.center
Coordinate([0.0, 0.0])
>>> c.width
2000.0
```

5.2.2 Groups

Primitive shapes can be grouped to create more complex shapes:

```
c = shapes.circle(1000)
r = shapes.rectangle(500, 1000)
g = shapes.group([c, r])
```

A group can be treated just like any other shape. It can be transformed and inserted into a higher level group. Groups thus behave just like primitive shapes.

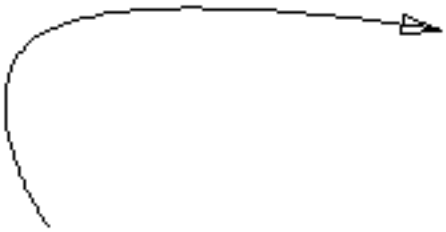
5.2.3 Examples

A set of arcs:

```
import math
g = shapes.group()
for radius in range(100, 1000, 100):
   a = shapes.arc_circle(radius, 1.0, math.pi)
   g.append(a)
```

Note: Remember that to view a shape you call `io.view(arg)`, where *arg* is the shape object. Here we would execute `io.view(g)`.

An arrow:



```
coords = [(0, 0), (0, 1000), (1000, 1000)]
p = shapes.path_bezier(coords, 1)
a = arrow(p, 100, 200)
```

5.3 Transforms

Shapes can be transformed:

Offset:

```
>>> c = shapes.circle(1000)
>>> c.center
Coordinate([0.0, 0.0])
>>> transforms.offset(c, (100, 200))
>>> c.center
Coordinate([100.0, 200.0])
```

Scale:

```
>>> c = shapes.circle(1000)
>>> c.width
2000.0
>>> transforms.scale(c, 2.4)
>>> c.width
4800.0
```

Rotate:

```
>>> r = shapes.rectangle(100, 200)
>>> r.height
200.0
>>> transforms.rotate(r, 3.14 / 4)
>>> r.height
212.16017194397654
```

CHAPTER 6

Plotters

6.1 Plotter Device Files

Chiplotle comes with device files for a number of plotters from different manufactuers, including Hewlett-Packard and Roland. Look in the /chiplotle/plotters folder to see if there's a device file that matches your plotter. If not, then look in your plotter manual to see if it emulates any of those plotters. For example, plotters by other manufacturers often emulate the HP7475a.

See *Configuration* for info on telling Chiplotle which device file you'd like to use.

Note: If Chiplotle does not have a device file for your plotter and you'd like to help us create one, all you need to do is send us the list of commands that your plotter recognizes and the ID string that it presents when Chiplotle opens it.

6.2 Offline Plotting

Sometimes you may want to work on your plotter code without having to actually connect your plotter and physically plot every command. There are several ways to use Chiplotle offline:

- Run python (not the chiplotle script!) and collect your commands into a Group, tuple or string, and when you want to see the results send the collected commands to `io.view( )`. This method works well when you're generating commands algorithmically and don't need any interaction with the plotter:

```
>>> from chiplotle import *
>>> commands = []
>>> commands.append(hpgl.SP(1))
>>> commands.append(hpgl.PA([(0,0)]))
>>> commands.append(hpgl.PD())
>>> commands.append(hpgl.PA([(1000,1000)]))
>>> commands.append(hpgl.PU())
>>> commands.append(hpgl.SP(0))
>>> io.view(commands)
```

- The above technique can also be used to write your commands out to an hpgl file for later viewing with your favorite hpgl viewer/converter:

```
>>> io.save_hpgl(commands, "diagonal.plt")
```

- Use a virtual serial port. This method allows you to simulate "live plotting," including sending queries to the plotter (for example, to find out where the pen is) or responding to user input.:

```
>>> from chiplotle import *
>>> from chiplotle.tools.plottertools import instantiate_virtual_plotter
>>> plotter =  instantiate_virtual_plotter(type="HP7550A")
>>> plotter.margins.hard.draw_outline()
>>> plotter.select_pen(2)
>>> plotter.goto(0,0)
>>> plotter.pen_down()
>>> plotter.goto(1000,1000)
>>> plotter.pen_up()
>>> plotter.select_pen(0)
>>> io.view(plotter)
```

Note: `io.view()` requires hp2xx to be installed on your system in order to convert the hpgl files into postscript for viewing by your OS's native ps viewer application.

CHAPTER 7

Hardware

7.1 Serial cables

Not all serial cables will work with all (or any) pen plotters, so make sure you've got the right cable before you decide your plotter is broken! Ideally you would want to get your plotter's operation manual to see the schematics of the serial cable it expects. Unfortunately these are sometimes hard to find. If you are completely in the dark, try this cable. The schematics are `here`. We have successfully used this cable with the Roland DX series.

Note: A 9 to 25 pin serial cable is usually what you need.

7.2 USB to Serial Adapters

In Windows, Chiplotle currently only supports **COM** ports to communicate with your plotter. On computers with real good-old RS-232 serial ports Chiplotle has no problem. Modern computers usually no longer have serial ports, so you need to use a USB to Serial interface to connect your plotter to your computer. Because Chiplotle only supports **COM** ports, what you need is a USB to Serial interface with drivers that supports VCP (Virtual COM Port), so that your USB to Serial interface shows up as a **COM** port. You may want to get USB to Serial interface with the FTDI Chip; it has VCP drivers and works well on Windows.

In OSX and Linux your USB to Serial interface will appear in your /dev directory as:

```
/dev/tty.XXX
```

To find the correct device name you can simply list the /dev directory before and again after plugging in the interface. The new `tty.XXX` entries in the /dev directory are your serial ports.

For example, before plugging in the adapter:

```
douglas$ ls /dev/tty.*
/dev/tty.Bluetooth-Modem        /dev/tty.Bluetooth-PDA-Sync
```

After:

```
douglas$ ls /dev/tty.*
/dev/tty.Bluetooth-Modem        /dev/tty.KeySerial1
/dev/tty.Bluetooth-PDA-Sync     /dev/tty.USA19Hfa14P1.1
```

So `/dev/tty.KeySerial1` and `/dev/tty.USA19Hfa14P1.1` are the new serials ports (in this case they're actually the same serial port, `/dev/tty.KeySerial1` is an alias added by the vendor for convenience).

CHAPTER 8

Chiplotle fundamentals

Warning: This section needs fleshing out.

In addition to being an HPGL plotter driver, *Chiplotle* is a vector drawing librarly specifically designed to work with these HPGL plotters. While other drawing computer tools are designed to create art on the screen (or for ordinary raster printing), Chiplotle knows about and understands some of the mechanics of drawing with pen plotters.

One can think of Chiplotle as consisting of three layers:

1. A high abstraction layer consisting of 2D shapes, like *line*, *circle*, *label*, etc.'
2. An interface / communication layer consisting of the HPGL language.
3. A plotter driver which manages communication between your hardware and software.

8.1 HPGL

How does Chiplotle communicate with a plotter? During the 70s and 80s, a variety of languages were developed by different manufacturers to control different brands of pen plotters, but the one language that gained most popularity and eventually became sort of a standard is HPGL (Hewlett-Packard Graphics Language).

Chiplotle supports all the standard HPGL commands, giving you full control of these plotters.

Further, Chiplotle provides plotter interfaces that allow you to contol the plotter as if through a control panel.

8.2 Chiplotle vector drawing

In addition to being an HPGL plotter driver, Chiplotle is also a general purpose vector drawing librarly. With Chiplotle you can create generic shapes that can be sent to an HPGL plotter directly for drawing, without you knowing anything about the underlying HPGL language.

8.3 Chiplotle geometry

8.3.1 Shapes

Chiplotle comes built in with a set of common shapes, like *line*, *circle*, *rectangle*, *ellipse*, etc.

These shapes are agnostic of any particular drawing language, such as HPGL or g-code.

8.3.2 Transforms

Chiplotle allows you to apply your standard geometric transformations to any shapes you may create with it.

8.4 Chiplotle-HPGL commands

In addition to the generic shape constructors, in Chiplotle you have access to specific HPGL command definitions.

All the standard HPGL commands are implemented in Chiplotle, and their class names corresponds to the two letter mnemonic used in the HPGL. Refer to the *Chiplotle API* for a list and documentation of all the HPGL commands.

Chiplotle HPGL commands can be instantiated as you would normally instantiate any other class. Some commands require arguments, others don't:

```
chiplotle> hpgl.PD()
PD(xy=[])

chiplotle> hpgl.CI(10)
CI(chordangle=None, radius=10.0)
```

All Chiplotle HPGL commands have a `format` attribute. This attribute returns a string representation of the HPGL command as sent to the plotter.

```
chiplotle> t = hpgl.PD()
chiplotle> t.format
'PD;'
```

CHAPTER 9

FAQ

Q: I'm trying to use Chiplotle with Windows but it seems Chiplotle can't find my hardware. What should I do?

A: No communication between Chiplotle and your hardware could be due to a variety of reasons. Check out the *Hardware* section for some possible causes.

Q: When I send a text file with HPGL commands to my serial port in the following way:

```
$ stty /dev/ttyUSB0 9600
$ cat bird.hpgl > /dev/ttyUSB0
```

my plotter starts drawing fine but will eventually just start pausing and drawing random straight lines. What's going on? Do I have to be concerned with overflowing the plotter's internal RAM?

A: Yes. The plotters buffer will fill up quickly, so you need to be listening to the plotter for any buffer overflow warnings and errors. This is generally done in one of two ways:

1. Setting up hardware hand-shaking between the plotter and your computer.
2. Querying the plotter for its buffer size before sending data to avoid truncation.

This is one of the tasks that Chiplotle manages for you so you don't have to worry about these low level technicalities. The easiest way to communicate with a plotter is to run Chiplotle by typing `chiplotle` from your terminal. This will run python, load Chiplotle, and instantiate soft-plotters for your hardware plotters found. Once in Chiplotle, send your HPGL file with the `write_file(filename)` method on the instantiated plotter(s), or send newly created HPGL commands via the `write()` method, like so:

```
chiplotle> plotter.write_file('my_file.hpgl')
chiplotle> plotter.write(hpgl.PA())
```

The `plotter` does the buffer managing for you. See the *Tutorial* for more details.

Q: Is there a facility in Chiplotle to send over already existing HPGL command files?

A: Yes. Chiplotle comes with the `plot_hpgl_file.py` executable script designed exactly for this purpose. To send HPGL files to your plotter simply run the script from the command prompt with the file as the argument:

```
$ plot_hpgl_file.py my_file.hpgl
```

To see the usage instructions run `plot_hpgl_file.py` with no arguments. Note that Chiplotle simply pipes the file to the plotter and does not check the syntax of the HPGL file.

You can also send HPGL files to your plotter from within a live Chiplotle session using a Plotter's own `write_file(filename)` method, like so:

```
chiplotle> plotter.write_file('my_file.hpgl')
```

Q: I installed chiplotle in Windows 98. Unfortunately running `chiplotle` from the `cmd` shell does not work. Windows isn't recognizing the `chiplotle` command.

A: Windows is not very friendly with Chiplotle! You will have to add Python (if not done so already) and the Chiplotle script files to your path. These are usually installed under `C:\Python26` and `C:\Python26\Scripts`.

Windows has a built-in dialog for changing environment variables (following guide applies to XP classical view): Right-click the icon for your machine (usually located on your Desktop and called "My Computer") and choose Properties. Then, open the Advanced tab and click the Environment Variables button.

> My Computer → Properties → Advanced → Environment Variables

In this dialog, you can add or modify User and System variables.

Note: To change System variables, you need non-restricted access to your machine (i.e. Administrator rights).

Another way of adding variables to your environment is using the set command:

```
set PATH=%PATH%;C:\path\to\chiplotle_executable
```

To make this setting permanent, you could add the corresponding command line to your `autoexec.bat`. `msconfig` is a graphical interface to this file.

Viewing environment variables can also be accomplished via `cmd.exe`: The command prompt will expand strings wrapped into percent signs automatically:

```
echo %PATH%
```

Note: Don't forget to also install `hp2xx`.

CHAPTER 10

Web Resources

10.1 HPGL references

- http://local.wasp.uwa.edu.au/~pbourke/dataformats/hpgl/
- http://cstep.luberth.com/hpgl.htm
- http://www.isoplotec.co.jp/HPGL/eHPGL.htm

10.2 Tools

- **hp2xx: an HPGL to *x* converter. Very handy for converting your HPGL files to other image formats. hp2xx in part of the**

> **Note:** **hp2xx** is required for exporting and previewing in Chiplotle.

- pstoedit: a Post Script to HPGL converter.
- autotrace: transform your bitmaps into vector graphics.
- potrace: transform your bitmaps into vector graphics. Works a bit differently than `autotrace`.

10.3 Hardware

- HP pen plotter museum

CHAPTER 11

Questions? Want to contribute?

If you have questions, found a bug or want to contribute, please join the Chiplotle mailing list.

CHAPTER 12

Coding Guidelines and Standards

Please refer to PEP8.

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

E

F

G

H

I

K

L

M

N

O

P

R

S

T

V

W

X

Y