

# Commentary on the Architecture of Matplotlib

## Scripting Layer: *matplotlib.pyplot*

*pyplot* is a user-facing interface to matplotlib mainly for interactive plots and simple cases of plot generation. It is not as powerful as the object-oriented API when dealing with more complicated plots but is much more convenient for new and experienced users alike.

Pyplot is not a class but instead a collection of functions. Each function updates the state which is preserved across all functions. Since all the functions in *pyplot* are global, we put them all in a box in the UML called “pyplot.py” just to show them as a collection of functions working for *pyplot*. We’ve also left out inputs and returns for the function as they aren’t necessary for understanding how each function changes the state.

We have further organized the methods in *pyplot* into groups according to the functionality. Firstly, `figure()`, `show()`, `draw()`, and `close()` is responsible for creating, displaying, updating and respectively closing a figure the user see. The next group of methods is for plotting various types of graphs like histogram, spectrogram. And the third group of methods to add various elements such as arrows or legends to the graph.

**How pyplot interacts with artist:** *pyplot* uses `setp()`, `getp()`, and `get()` to set and get object properties, and `findobj()` to find artist objects.

**How pyplot interacts with backend:** *pyplot* uses `mpl_connect` and `mpl_disconnect` to connect and disconnect to the `FigureCanvasBase` in backend. Also, The canvas of a figure created through *pyplot* is associated with a figure manager(`FigureManagerBase` or its derivatives), which handles the interaction between the figure and the backend. *pyplot* keeps track of figure managers using an identifier, the “figure number” or “manager number” (which can actually be any hashable value); this number is available as the `:attr:`number`` attribute of the manager. The figure manager sets up default key and mouse button press handling by hooking up the `.key_press_handler`` to the matplotlib event system. This ensures the same shortcuts and mouse actions across backends.

## Artist Layer: `matplotlib.artist`

*artist* is an abstract base class designed for objects that render onto a `FigureCanvas`. Practically all visible elements in a figure have *artist* as a superclass.

**Collection:** A base class which handles the efficient drawing of meshes, lines, and shapes.

**Patch:** 2D elements with designated face and edge color, with subclasses capable of drawing arrows and polygons using said color.

**Axis:** The base class for the X-axis and Y-axis.

**OffsetBox:** A base class with subclass designed to be drawn/positioned relative to its parent.

**Text:** A class to store and manage the drawing of text for given positions/points.

**Tick:** An abstract base class for the axis ticks (which mark a position on an axis), and their corresponding label.

**Table:** A table/grid of cells indexed by their row and column position.

**Figure:** The top-level container for all the plot elements.

**Line2D:** Allow for the generation of lines using various styles (solid, dashed, etc.).

**Legend:** Responsible for placing legends on axes.

**How artist interacts with backend:** *artist* relies on the `draw(renderer)` method as the main means of communication to the backend. Given a renderer, the draw method is responsible for converting the representation of *artist* into machine readable data (such as instructions to create an SVG), which is then rendered by the backend.

## Backend Layer: matplotlib.backend\_bases

FigureCanvasBase: An abstract class that represent a drawing area.

RendererBase: An abstract class to draw/render plots.

Event: A base class that handles all the events(user inputs), e.g. mouse click, keyboard input.

GraphicContextBase: An abstract base class that provides style, color and other properties for canvas.

ToolContainerBase; The base class for the Toolbar class of each interactive backend.

StatusbarBase: The base class for the various types of status bar from widget toolkit for creating graphical user interfaces including tk, gtk3, qt5, wx)

## Reference:

<https://matplotlib.org/3.1.3/api/index.html>