RFC 025 - DSL - Custom Plotting and Charting Library

Title: Proposal for Developing a Custom (Static) Plotting and Charting Library
Primary Context: Dataflow
Tags: #Graph Native, @Release-Ready, @Standard Library, DSL, Plotting, RFC
Related: RFC 025, RFC 042, RFC 121

> Making Divooka THE MOST COMPREHENSIVE exhaustive and customizable plotting/charting tool.

Contents:

1. Introduction

As we continues to innovate with our visual scripting platform, Divooka, it has become evident that an integral component of our ecosystem is a robust, flexible, and easy-to-use plotting and charting library. While existing solutions like **ScottPlot**, **GNUPlot**, and others provide powerful tools, developing our own library offers several advantages that align with our vision of a seamless, graph-native experience for our users.

1. **Graph-Native Experience**: Existing libraries are primarily designed for traditional programming environments. By developing our own library, we can tailor the API to integrate smoothly with our visual scripting paradigm. Functions in the library will become nodes in our graph-based environment, allowing users to connect inputs and outputs intuitively. This native integration will reduce the learning curve and enhance productivity by making the data visualization process more intuitive and visual.
2. **Layer of Abstraction**: Developing a custom API provides us with the flexibility to abstract away the underlying implementations. While we currently use a combination of **ScottPlot**, **GNUPlot**, **SkiaSharp**, and **matplotlib**, our custom library will allow us to switch or combine these implementations seamlessly in the future. This abstraction ensures that our users are shielded from changes and can continue to work with a consistent API, regardless of the underlying technology.
3. **Enhanced Customization**: A custom library will enable us to include specific features and optimizations that are crucial for our users. We can prioritize performance improvements, add unique plot types, and ensure compatibility across different platforms, providing a superior user experience.

1.1 Requirements

The goal is to provide a canonical stable Divooka-first (native or not) node-based API, irrelevant of underlying implementation. Even if we are going to embed Python and switch to a Python based library (thus being more "straightforward" in implementation) in the future - it might be a long future, and the API stays the same. What matters we need something native, fits within current framework/infrastructure, light-weight and works right now[^1].

[^1]: Implementation Note: This is especially true for embedded Python, which we will NOT make it canonical/out-of-box in any near future anyway (i.e. it will be not available in "version 1") - so don't expect making use of existing Python libraries for the purpose of DSL Divooka (standard) package implementation.

- Provide (rough number) 600+ uniquely crafted plot/chart/configurations, making Divooka the most feature complete 2D plotting/charting platform. Making plots from data and optionally text/markdown! Ideally everything in a single package.

To ensure our plotting library meets the needs of our users, especially in a graph-native environment, the following requirements are essential:

1. Simplicity and Clean API:
    - Intuitive Design: The API should be designed so that users can easily pass in data and generate plots without needing to manage complicated parameters. The goal is to minimize the configuration burden on users.
    - Defaults and Presets: Provide sensible defaults and presets for common plotting scenarios, allowing users to create plots with minimal input. Advanced options should be available but not mandatory for basic usage.
2. Comprehensive Plot Types:
    - Basic Plots: Line, bar, scatter, pie, and histogram charts should be included as foundational plot types.
    - Advanced Plots: Support for heatmaps, 3D plots, financial charts (e.g., candlestick charts), and geospatial plots.
    - (Experimental) Custom Plots: Enable users to define custom plot types or extend existing ones to cater to specific needs.
3. Cross-Platform Compatibility:
    - Ensure the library works seamlessly on Windows, macOS, and Linux (also on Linux based servers).
    - Utilize cross-platform graphics libraries like SkiaSharp to render plots consistently across different operating systems.
4. Performance Optimization:
    - The library should be able to handle large datasets fairly efficiently and generates plots fast.
5. Integration with Divooka:
    - The library should be tightly integrated with Divooka's visual scripting environment and its vector/data grid processing facilities.
    - Functions should be represented as nodes with clear inputs and outputs, making it easy for users to build complex visualizations through node-based workflows.

1.2 Features

Divooka Native Types Integration

Provide DataGrid extensions for certain plot.types to allow easier plotting especially finance and time series data. Provide extensions to accept DateTime[] directly as input or Time as XAxis version for many charts.

2. Implementation

Integration (embedding) of already-available tools (GNUPlot and Python matplotlib and seaborn) is allowed and likely necessary at the start (though ideally we implement everything natively in .Net Core), but try to keep things below 20Mb and make sure everything we include as dependency is heavily exploited for value!

Key feature is configurable style and data-driven plotting, charting, and (geographical) mapping. A key design challenge is to figure out an efficient and relatively consistent configuration/default theme and make sure we do deliver that 600 (unique) chart types (not including minor variants)!

To ensure a robust and efficient development process, the implementation of our custom plotting and charting library will be divided into three stages. Each stage will focus on specific goals and improvements, ensuring that we deliver a high-quality product incrementally.

2.1 Stage 1: Native & Low-Level Drawing

Objective: To develop the foundational library using ScottPlot and SkiaSharp, providing most chart types while minimizing dependency size.

2.2 Stage 2: Integration

Objective: To enhance the library by integrating GNUPlot, matplotlib, and seaborn through Python.Net, standardizing plot types, and improving drawing capabilities.

Tasks:

1. Integration Setup:
    - Integrate Python.Net to enable the use of Python libraries.
    - Set up the necessary configurations for GNUPlot and matplotlib/seaborn.
2. Enhanced Plot Types:
    - Expand the range of available chart types by incorporating capabilities from GNUPlot and matplotlib/seaborn.
    - Standardize the appearance and behavior of plots across different implementations.
3. Advanced Features:
    - Implement advanced plotting features such as heatmaps, 3D plots, and financial charts.
    - Ensure seamless integration with the existing API.
4. Performance Optimization:
    - Optimize performance by leveraging the strengths of each integrated library.
    - Ensure efficient data handling and rendering.
5. Testing and Documentation:
    - Conduct extensive testing to validate integration and performance.
    - Update documentation to reflect new features and usage examples.

Outcome: A more comprehensive plotting library with enhanced plot types and standardized drawings, leveraging the power of GNUPlot, matplotlib, and seaborn.

2.3 Stage 3: Consolidation

Objective: To optimize the overall library implementation, reduce dependencies, ensure cross-platform robustness, and solidify graphical styles.

Tasks:

1. Optimization:
    - Refine the implementation to improve performance and reduce resource usage.
    - Streamline code and eliminate redundancies.
2. Dependency Management:
    - Assess and minimize dependencies to reduce the overall footprint of the library.
    - Ensure that the library remains lightweight and efficient.
3. Cross-Platform Support:
    - Ensure that the library functions seamlessly across Windows, macOS, and Linux.
    - Address any platform-specific issues and optimize compatibility.
4. Graphical Styles:
    - Standardize graphical styles and themes to provide a consistent look and feel.
    - Enhance customization options for users.
5. Final Testing and Documentation:
    - Conduct rigorous testing to ensure stability and robustness.
    - Finalize documentation and provide comprehensive guides and examples.

Outcome: A fully optimized, cross-platform plotting library with reduced dependencies, consistent graphical styles, and robust performance, ready for widespread use and integration with Divooka.

2.4 Technical Notes

For static report generation, at the moment the overall framework is just to generate each individual charts/displays first, then combine them using canvas positions to form the final report. In the future we might provide advanced automatic layouts and templates, but it's still based on combining charts/displays.

On the surface, it looks like we'll need a way to provide sufficient customization without overwhelming configurations. In reality (per Houdini's design and original Gospel envision) There is a way to do that on the language/GUI level (using property panel, a feature I will roll out in the first full version), but we haven't got to materialize it yet.

For the purpose of the plotting library, we should just focus on providing as many configurations as appropriate, and don't worry about how it looks with "ConfigureXXX" yet.

3. Plot Types

Our plotting library aims to cover a wide range of charting needs, balancing simplicity and comprehensiveness. The following plot types should be included:

Basic Plots

- Line Plot: Ideal for visualizing trends over time.
- Bar Chart: Useful for comparing categorical data.
- Scatter Plot: Effective for displaying relationships between variables.
- Pie Chart: Suitable for showing proportions of a whole.
- Histogram: Perfect for representing the distribution of data.

Advanced Plots

- Heatmap: Displays data density or intensity across a two-dimensional space.
- 3D Plot: Allows for visualization of data in three dimensions.
- Financial Charts: Includes candlestick and OHLC charts for financial data analysis.
- Geospatial Plots: Enables plotting data on maps for geospatial analysis.

Specialized Plots

- Box Plot: Used for depicting data distributions and identifying outliers.
- Violin Plot: Combines aspects of box plots and density plots to show data distributions.
- Network Graph: Useful for visualizing relationships between entities.
- Surface Plot: Represents three-dimensional data on a two-dimensional plane.
- Population Pyramid
- Gant Chart
- Word Frequency Map

1. Sankey Diagram:
    - Visualizes the flow of resources or data between different nodes.
    - Useful for showing energy flows, material flows, or decision processes.
2. Radar Chart (Spider Chart):
    - Compares multiple variables across different categories.
    - Ideal for performance analysis, skill assessment, or comparing products.
3. Bubble Chart:
    - Similar to a scatter plot but includes a third variable represented by the size of the bubbles.
    - Useful for visualizing relationships between three variables.
4. Treemap:
    - Displays hierarchical data using nested rectangles.
    - Effective for showing proportions within categories, such as file sizes or budget allocations.
5. Sunburst Chart:
    - Similar to a treemap but uses a radial layout.
    - Useful for visualizing hierarchical data in a more compact and visually appealing way.
6. Chord Diagram:
    - Represents relationships between different entities in a circular layout.
    - Useful for visualizing connections or flows between groups.
7. Violin Plot:
    - Combines aspects of box plots and density plots.
    - Useful for comparing distributions between different groups.
8. Streamgraph:
    - Similar to an area chart but emphasizes the flow and changes over time.
    - Useful for visualizing changes in data streams over time.
9. Network Graph:
    - Displays relationships and connections between nodes.
    - Ideal for social network analysis, computer networks, and biological networks.
10. Funnel Chart:
    - Visualizes stages in a process or sales pipeline.
    - Useful for identifying potential bottlenecks or drop-off points in a process.
11. Waterfall Chart:
    - Shows the cumulative effect of sequentially introduced positive or negative values.

    - Useful for financial analysis, such as visualizing profit and loss statements.
12. Heatmap:
    - Represents data density or intensity using color gradients.
    - Ideal for showing patterns or correlations in data matrices.
13. Doughnut Chart:
    - Similar to a pie chart but with a central hole.
    - Useful for showing proportions of a whole with a clearer comparison between categories.
14. Polar Area Chart:
    - Similar to a radar chart but with segments that represent values.
    - Useful for displaying cyclic data like time series or seasonal trends.
15. Contour Plot:
    - Represents three-dimensional data on a two-dimensional plane using contour lines.
    - Useful for geographical data, weather patterns, and scientific data.
16. Calendar Heatmap:
    - Displays data over time in a calendar format.
    - Useful for visualizing activities, events, or trends over days, weeks, and months.
17. Hexbin Plot:
    - Similar to a scatter plot but bins the data points into hexagonal bins.
    - Useful for visualizing the density of data points in large datasets.
18. Parallel Coordinates Plot:
    - Represents multivariate data with parallel axes.
    - Useful for visualizing relationships between multiple variables.
19. Stacked Area Chart:
    - Similar to an area chart but stacks multiple data series.
    - Useful for showing the cumulative contribution of different categories over time.
20. Box Plot:
    - Visualizes the distribution of data through quartiles.
    - Useful for identifying outliers and comparing distributions across categories.

Customizable Plots (Charts)

- Provide users with the ability to create custom plots by extending existing types or defining new ones.
- Allow for the addition of annotations, custom markers, and interactive elements.

More advanced charts can be considered as templates with data-driven behaviors:

- Mind Graph
- Global Map
- Speed Dial
- Fantasy Map
- Custom World Map

DSL - Application Mockup

An extension of basic plotting and specialized charts, and a mix of static report and Zora Drawing package - define GUI control components (as static renders) and allow very basic application mock up use. When done well this can be combined with Zora.MobileAppBuilder to serve as a dedicated render delegate for live preview purpose (i.e. without running the host or do HTML gen and we are able to see what the app will look

like e.g. in PV1 Neo). Then it's more like a WYSISYG app builder - but the beauty here is that we are not bothering mouse interaction for building, and it's an offline renderer - great replacement for Affinity Design for GUI design. A full sized layout options would borrow designs from Godot because its layout settings is quite easy to use.

DSL - Text Based Plotting (Dhole)

Like GNUPlot and Mermaid, but we will NOT be using those directly.

4. Conclusion

By developing a custom plotting and charting library tailored to our graph-native visual scripting environment, Methodox Technologies, Inc. can provide a powerful, intuitive tool for data visualization. This library will not only enhance the capabilities of our Divooka platform but also offer our users a seamless and productive experience. The proposed requirements and plot types ensure that we meet the diverse needs of our users while maintaining simplicity and flexibility.

This RFC also serves as a quality-benchmark: pushing to max quality in a single package, demonstrating production-level feature-completeness requirements and serve as reference to the expected quality whenever we make new releases.

References

Existing good plotting tools below - All look very good, just not suited for small scripts.

- OxyPlot
- LiveChartsv2: https://github.com/beto-rodriguez/LiveCharts2 (with map.support)
- https://scottplot.net/faq/compare/
- Seaborn: High level API, statistical focused
- https://www.kdnuggets.com/visualizing-data-statology-primer
- https://github.com/alandefreitas/matplotplusplus
  https://stackoverflow.com/questions/4283731/plotting-package-for-c

See doc below for some needed plot type or chart categories. Let's implement them all[^2]!

[^2]: When it comes to integration, preferably raw C#, then embedded CLI like GNUPlot, last resort is embedded Python - unless we move the schedule of embedded python earlier and make it official soon

- Plotly
- PowerBI
- ScottPlot
- GNU Plot
- Mermaid MD
- Matplotlib
- D3.js
- https://gojs.net/
- https://js.cytoscape.org/
- MSDN Chart types reference for Excel : https://support.microsoft.com/en-us/office/available-chart-types-in-office-a6187218-807e-4103-9e0a-27cdb19afb90

- https://gojs.net/latest/

Notice apparently there is a lot of variants/noise in terms of plotting providers/libraries. But those are NOT even core technology on its own - curious people put money and effort into developing some triviality and call it a product.

Appendix

A. Usage Studies

In this section we take notes on how things are done in certain existing platforms to borrow ideas.

Excel: Population Pyramid

https://youtu.be/txSSvUqllKU?si=lRYC4TJZrkl2jZxQ

Textual visual description of steps: https://populationeducation.org/how-to-build-a-population-pyramid-in-excel-step-by-step-guide/ Obviously that's too many steps. I claim even with data cleanibg it should take no more than 3 steos (3 mouse/keyboard clicks)

Excel: Add Arbitrary Additional Text Labels

Impossible.