INTRODUCTION

Chart Guru is a professional 2D charting solution for Unity, built for polished dashboards, HUDs, runtime interfaces, editor tools, analytics screens, live monitoring displays, and data-rich game UI.

It helps teams create beautiful, animated charts through designer-friendly workflows or a clean Swift Charts-inspired C# authoring style. The package brings the Swift Charts mental model to Unity in approachable terms: choose marks, bind meaningful values, configure axes and styling, add annotations and interaction, then generate readable Unity-ready code when needed.

If you already know Swift Charts, the near one-to-one mental model lets you reuse that knowledge instead of learning an unrelated chart grammar. If you do not, the same semantic API and carefully chosen defaults provide a dependable path to professional dashboards, industrial interfaces, and enterprise tools.

At a glance

Chart families	Bar, line, area, point, scatter, bubble, pie, donut, radar, heatmap, candlestick, sparklines, mini charts, indicators, and more.
Authoring paths	Create charts without code, design them visually, generate reusable code, or write charts directly when full control is needed.
Unity UI targets	Use charts in Canvas/uGUI screens, UI Toolkit layouts, UI Builder workflows, UXML/USS interfaces, runtime scenes, editor windows, inspectors, HUDs, and dashboards.
Swift Charts migration	Use a Swift Charts-style C# workflow, paste SwiftUI Chart snippets through the importer, and generate equivalent Unity chart code.
Editor tooling	Includes a searchable gallery, WYSIWYG designer, live previews, code generation, data mapping, theme editing, scene creation tools, sample scenes, and reusable workflows for teams.
Flexible data	Work with manual data, generated mock data, reusable data assets, C# lists and arrays, CSV, JSON, Google Sheets, web endpoints, finance-style OHLC data, ring buffers, and custom feeds.
Live and interactive	Stream values, append or replace data, bind external feeds, update monitoring dashboards, and support hover, selection, gestures, scrolling, panning, zooming, viewport windows, and range brushing.
Presentation quality	Use legends, axes, labels, annotations, gradients, shadows, glow, palettes, rounded corners, line interpolation, symbols, color scales, reusable themes, smooth transitions, and chart morphing.
Performance focus	Designed for GPU-accelerated rendering, Unity render pipeline compatibility, stable runtime updates, low-allocation refresh paths after warmup, and responsive animation.
Samples and extras	Includes 40+ example scenes, UI Toolkit samples, and the full Mock Magic quick-test and fake-data creation tool.
Integration output	Refresh live chart textures for companion tools, with optional composed text overlays for titles, axes, legends, and data labels. The optional LED Studio integration uses this output to drive LED boards from live charts.

Built for serious Unity UI

Chart Guru is intended for games, simulations, internal tools, command centers, business dashboards, education apps, financial displays, and asset-store-quality editor extensions where charts need to look finished and behave like part of the product.

CONTENTS

CHART LANGUAGE USED IN THIS GUIDE

A chart has a few named regions and building blocks. Knowing these terms makes the Inspector and API much easier to understand.

Term	Meaning
Chart area	The whole rectangle reserved for the chart, including title, subtitle, plot, axes, legend, labels, background, and border.
Plot area	The inner data region where bars, lines, points, areas, heatmap cells, and reference rules are drawn.
Mark	One visual data item or part of a series. Examples: BarMark, LineMark, AreaMark, PointMark, SectorMark, RectangleMark, RuleMark, RadarMark, and CandleStickMark.
Axis	A value or category guide. Cartesian charts normally use an X axis and a Y axis. Pie, donut, radar, and compact charts usually hide axes.
Scale and domain	The scale converts data values to positions. The domain is the range of data values shown, such as 0 to 100 or Jan to Dec.
Tick and grid line	A tick is a labeled value on an axis. A grid line extends a tick across the plot area to help compare values.
Series	A group of related marks, such as Sales and Expenses. Series keys drive grouping, stacking, colors, and legend entries.
Legend	The label list that explains series colors, symbols, or categories.
Title and subtitle	Text attached to the chart area, typically used for the main message and context such as period, units, or filter.
Data label and annotation	A data label shows a value near a mark. An annotation is richer explanatory text attached to a mark or reference line.

Beginner rule of thumb: choose bars for comparing categories, lines or areas for trends, points for relationships, pie or donut charts for simple part-to-whole stories, heatmaps for matrix intensity, and candlesticks for OHLC financial data.

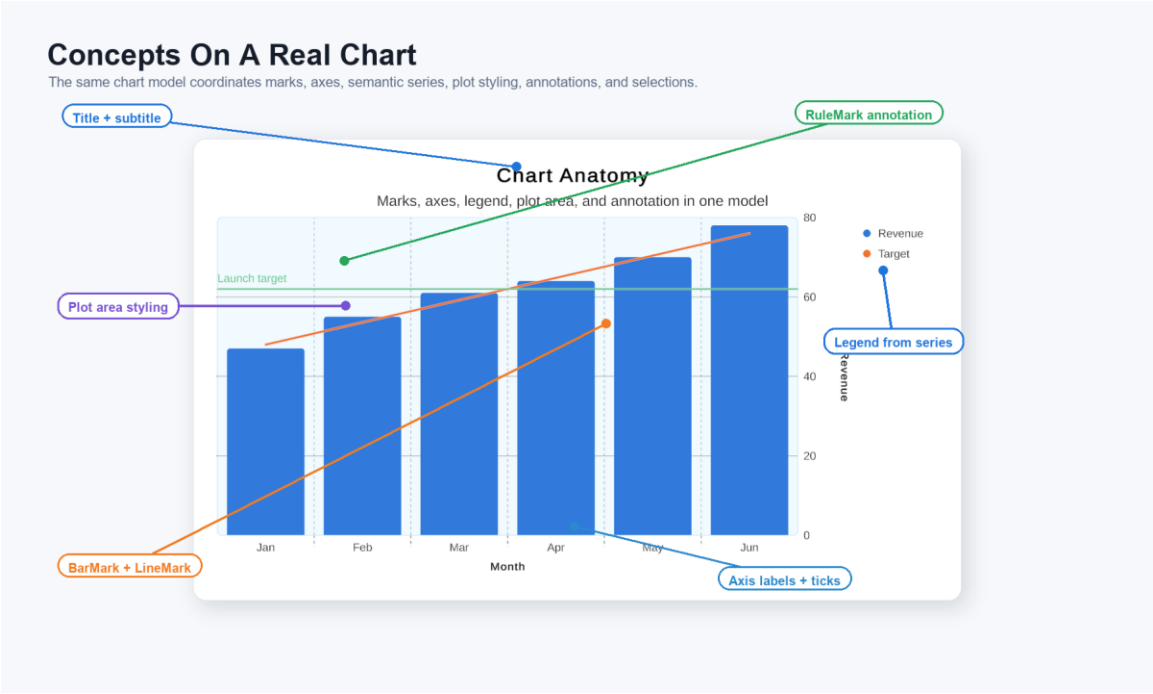


Figure 1. Chart anatomy: title, subtitle, plot area, axes, legend, marks, annotations, and encoded series are separate parts of the same Chart model.

INSTALLATION AND QUICK START

REQUIREMENTS

- Unity 2022.3 or newer.
- Burst 1.8.0 or newer, Collections 1.2.4 or newer, Mathematics 1.2.0 or newer, Newtonsoft JSON 3.2.2, and TextMeshPro 3.0.9 or newer. These are declared in the package manifest.
- A Canvas is recommended for the most common runtime UI path. UI Toolkit is also supported through *ChartGuruElement*.

INSTALL THE PACKAGE

1. Import Chart Guru from the Unity Asset Store.
2. Let Unity resolve the dependencies declared in package.json.
3. Open Tools > Chart Guru > Chart Gallery to confirm the package is installed and to browse live examples.
4. Open Tools > Chart Guru > Create Simple Chart when you want Chart Guru to create a sample Canvas and chart GameObject for you.

FASTEST FIRST CHART

5. Create a Canvas and add an empty child GameObject.
6. Add ChartGuru > Chart Graphic to the child GameObject.
7. Add ChartGuru > Data Sources > Mock Chart Data Source to the same GameObject.
8. Choose a Pattern such as TrendingUp, set Series Count and Points Per Series, and click Generate if needed.
9. Change Chart Type, theme, axes, labels, and legend settings on ChartGraphic. The chart updates in Edit Mode and Play Mode.

BUILD YOUR FIRST CHART

INSPECTOR PATH: NO CODE

Use the Inspector path when you want a fast dashboard, HUD, prototype, or designer-owned chart. ChartGraphic owns rendering and chart settings. A data source component feeds marks into it.

- Use *MockChartDataSource* for instant generated data.
- Use *ManualChartDataSource* when you want to type values directly into the Inspector.
- Use file or web data sources when a chart should follow CSV, JSON, Google Sheets, HTTP, Yahoo Finance, or high-throughput live samples.

CODE PATH: FULL CONTROL

Use the code path when chart data is produced by gameplay, simulation, analytics, editor tooling, or custom runtime systems.

```
using ChartGuru;
using UnityEngine;

public sealed class MonthlySalesChart : MonoBehaviour
{
    private ChartGraphic _graphic;

    private void Start()
    {
        float[] values = { 65f, 89f, 45f, 78f, 92f, 55f };
        string[] labels = { "Jan", "Feb", "Mar", "Apr", "May", "Jun" };

        ChartBuilder builder = Chart.Create();
        for (int i = 0; i < values.Length; i++)
        {
            BarMark bar = new BarMark(i, values[i])
                .CornerRadius(8f)
                .ForegroundStyle(PlottableValue.Value("Month", labels[i]));
            bar.CategoryLabel = labels[i];
            builder.Add(bar);
        }

        Chart chart = builder
            .ChartTitle("Monthly Sales")
            .ChartSubtitle("Current fiscal year")
            .ChartXAxis(x => x.Label("Month"))
            .ChartYAxis(y => y.Label("Sales").TickFormat("F0"))
            .ChartLegend(LegendPosition.Hidden)
            .Animation(Animation.Default.Delay(0.05f))
            .Build();

        _graphic = ChartCanvasHelper.RenderToRectTransform(
            chart, GetComponent<RectTransform>(), true, "Monthly Sales");
    }

    private void OnDestroy()
    {
        ChartCanvasHelper.RemoveChart(_graphic);
    }
}
```

HORIZONTAL BARS

Horizontal bars are not a separate ChartType. They are regular BarMark data with horizontal orientation. In the Inspector, use Bar Orientation. In code, either assign BarLayout.Horizontal explicitly or bind a numeric X value with a categorical Y value as shown below.

```
BarMark bar = new BarMark(
    PlottableValue.Value("Sales", value),
    PlottableValue.Value("Category", "Long category name"))
    .CornerRadius(6f);
```

CORE CONCEPTS

MARKS

A mark is the smallest drawable unit in a chart. The chart type often follows the mark type, but Chart Guru can also compose compatible cartesian marks in one chart, such as bars plus rule marks or points plus a line.

Mark	Used for
BarMark	Vertical or horizontal bars, ranges, grouped bars, stacked bars, and mini bars.
LineMark	Trend lines and connected point series.
AreaMark	Filled trends, stacked areas, range bands, and confidence bands.
PointMark	Point plots, scatter plots, and bubble-like encodings when symbol size is mapped.
SectorMark	Pie and donut slices.
RectangleMark	Heatmap cells and rectangular ranges.
RuleMark	Horizontal or vertical reference lines and threshold markers.
RadarMark	Radar or spider chart spokes and filled shapes.
CandleStickMark	Open, high, low, close financial candles.

BUILDER

The builder is a fluent way to assemble a Chart model. You add marks, configure the chart, then call Build. The same Chart model can be rendered in Canvas, UI Toolkit, editor previews, or custom code.

```
Chart chart = Chart.Create()
    .Add(new LineMark(0f, 42f))
    .Add(new LineMark(1f, 48f))
    .ChartTitle("Trend")
    .ChartXAxis(x => x.Label("Time"))
    .ChartYAxis(y => y.Label("Value"))
    .Build();
```

SEMANTIC ENCODING

Semantic encoding means you describe what a mark represents instead of hard-coding every color. Chart Guru can then assign consistent colors, create legend entries, and group marks into series.

```
bar.ForegroundStyle(PlottableValue.Value("Product", "Electronics"));

// Use explicit color only when you need direct control.
bar.ForegroundStyle(Color.red);
```

SERIES, STACKING, AND LEGENDS

Marks with the same SeriesKey belong together. ForegroundStyleKey controls color assignment. For multi-series charts, use semantic foreground style values or set SeriesKey and ForegroundStyleKey yourself. Stacking is controlled with MarkStackingMethod: Unstacked, Standard, Normalized, or Center.

Figure 2. Stacking options: unstacked, standard, normalized, and center modes let the same bar dataset show grouped series, absolute totals, 100% composition, or centered totals around the baseline.

CHART TYPES AND WHEN TO USE THEM

Chart type	Best use	Notes
Bar	Compare values across categories.	Use BarLayout.Horizontal for long category names. Supports rounded corners, ranges, grouping, and stacking.
Line	Show change over time or ordered X values.	Supports Linear, Monotone, CatmullRom, Cardinal, Natural, and Step interpolation modes.
Area	Show trend plus magnitude.	Can fill below a line or between ranges. Supports stacking and centered streamgraph style.
Point / Scatter / Bubble	Show individual observations or relationships.	Use symbols and symbol size scale for additional dimensions.
Pie / Donut	Show simple part-to-whole composition.	Use sparingly. Works best with a small number of slices and clear labels.
Radar	Compare several attributes per entity.	Useful for profiles, skill charts, and multi-axis comparison.
HeatMap	Show intensity across a two-dimensional grid.	Built from RectangleMark and color scales.
Candlestick	Show financial OHLC data.	Uses open, high, low, close columns or CandleStickMark values.
Sparkline / MiniBar / MiniPie / MiniDonut / Indicator	Embed compact values in HUDs, dashboards, tables, or inspector panels.	Compact types hide axes and reduce padding by default.

CHOOSING A CHART TYPE

- Use bar charts for ranking and category comparison.
- Use line charts for a trend where order matters.
- Use area charts when accumulated magnitude or range matters.
- Use scatter or bubble charts when both X and Y are measured values.
- Use heatmaps when a matrix pattern matters more than individual numbers.
- Use compact charts when the chart is a supporting signal, not the whole focus.

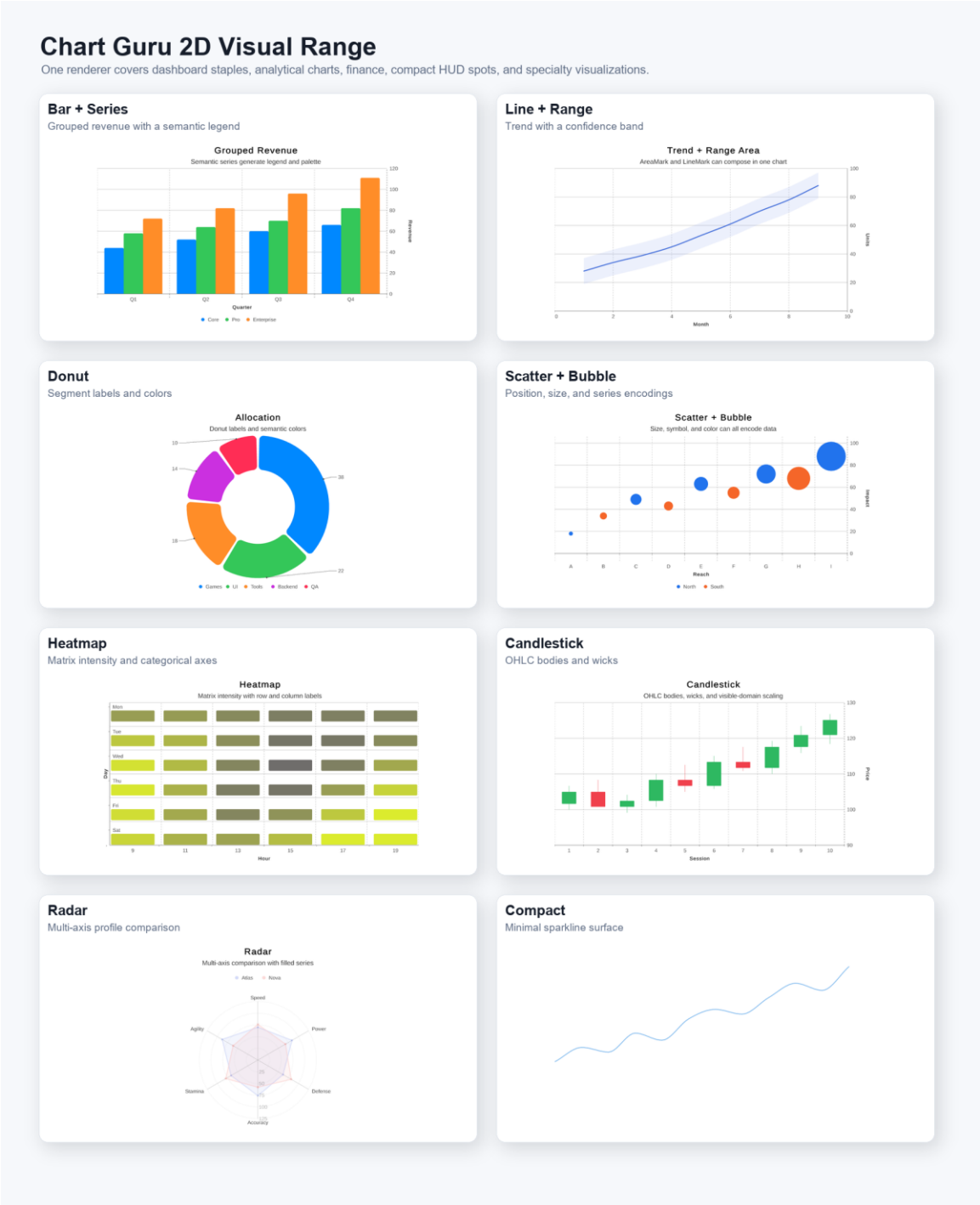


Figure 3. 2D chart type showcase: grouped bars, line and area trends, donut composition, scatter and bubble, heatmap, candlestick, radar, and compact dashboard charts.

DATA SOURCES

Data sources are components that build marks and apply them to a target chart. They work in Edit Mode and Play Mode where appropriate, and most data source inspectors expose Refresh or Generate actions for quick iteration.

Source	Use it for
MockChartDataSource	Instant generated data for prototypes and demos. Patterns include Random, Linear, Exponential, Logarithmic, Sine, Cosine, TrendingUp, TrendingDown, Seasonal, Stepped, Sparse, RandomWalk, BellCurve, Sawtooth, and GaussianNoise.
ManualChartDataSource	Hand-entered labels, values, series, colors, and reference rules directly in the Inspector.
CsvChartDataSource	CSV, TSV, semicolon-separated files, TextAssets, StreamingAssets, inline text, or absolute paths.
JsonChartDataSource	JSON files or inline JSON, including nested data via root selectors.
GoogleSheetsChartDataSource	A shared Google Sheets URL converted to chart-ready tabular data.
WebChartDataSource	HTTP CSV or JSON endpoints with headers, polling, retry, and backoff.
YahooFinanceChartDataSource	Stock and financial time-series data for line or candlestick charts.
RingBufferChartDataSource	High-throughput live samples, appended efficiently and flushed once per frame.

MOCK DATA SOURCE

MockChartDataSource is the recommended starting point. Choose a pattern, label preset, point count, series count, value range, and noise factor. It can generate once or stream continuously with Constant, Random, Burst, or Drip rhythm.

```
MockChartDataSource mock = gameObject.AddComponent<MockChartDataSource>();
mock.Pattern = SeriesPattern.TrendingUp;
mock.SeriesCount = 3;
mock.PointsPerSeries = 12;
mock.LabelPreset = LabelPreset.MonthsShort;
mock.GenerateOnce();
mock.StartStreaming();
```

FILE AND WEB MAPPING

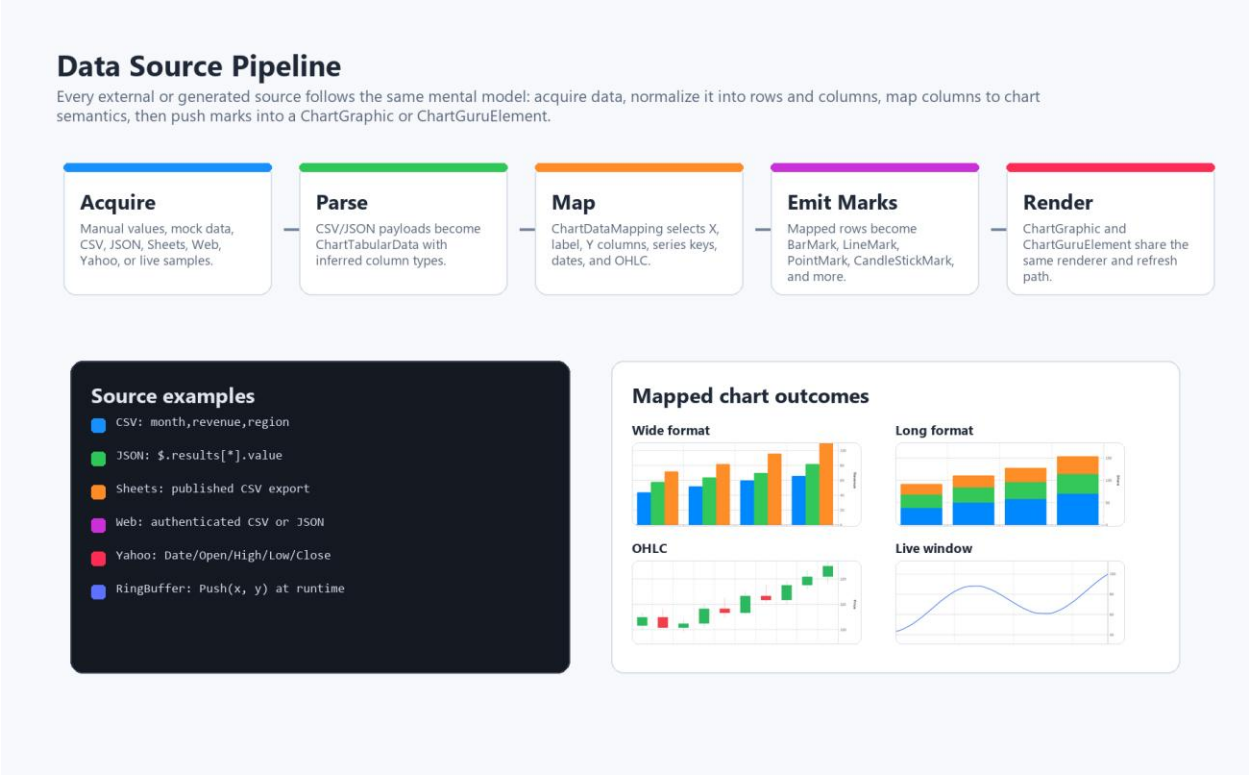
CSV, JSON, Sheets, Web, and Yahoo sources parse data into rows and columns, then use *ChartDataMapping* to decide which columns become X values, Y values, labels, series, and OHLC candles.

- XColumn becomes the X axis. If empty, row index is used.
- LabelColumn becomes the category or data label when present.
- YColumns are the plotted numeric values. WideFormat treats each Y column as its own series.
- SeriesByColumn pivots long-format data into multiple series.
- OpenColumn, HighColumn, LowColumn, and CloseColumn create candlestick marks.

- MaxRows limits how many parsed rows are consumed.

DATA SOURCE MENTAL MODEL

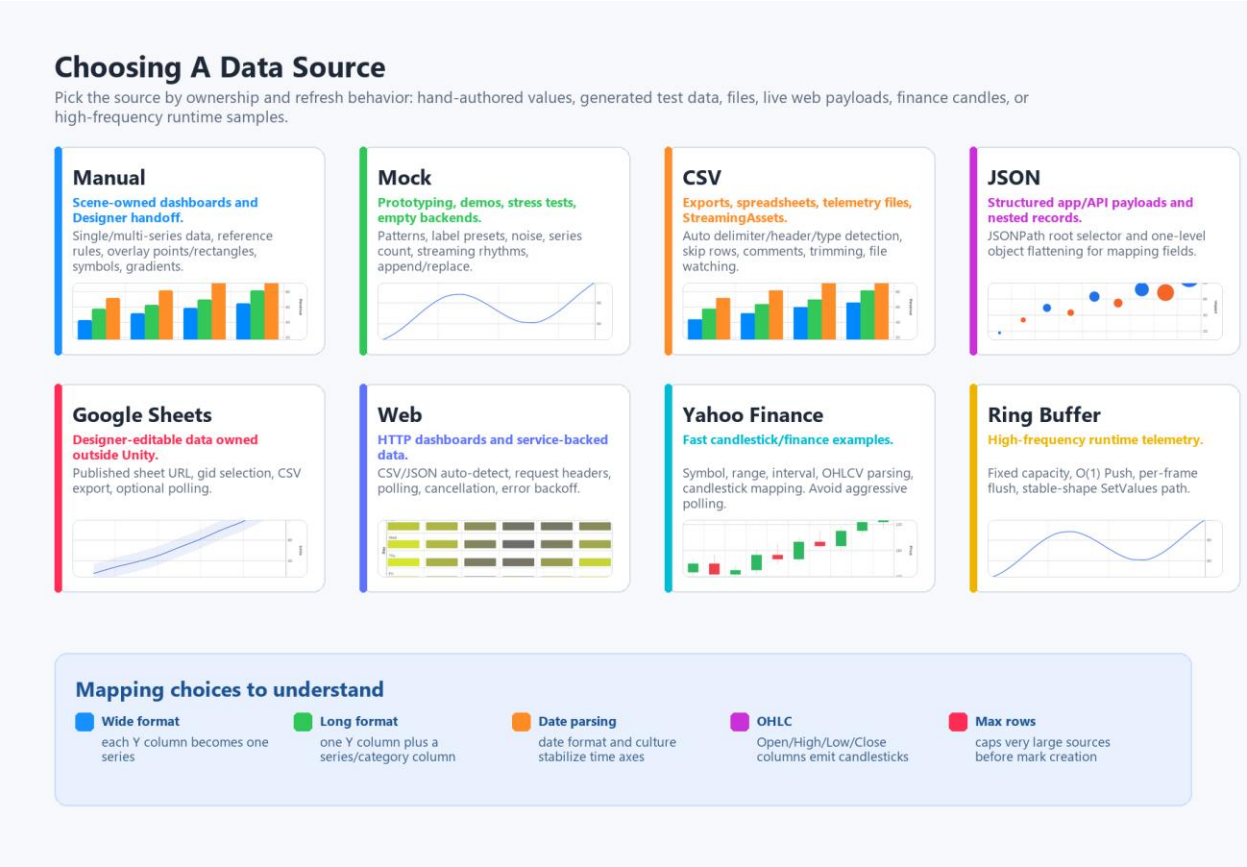
A data source is not just a loader. It answers four questions: where values come from, how raw text or generated samples become rows and columns, how those columns map to chart semantics, and when the chart should refresh. Once that model is clear, CSV, JSON, Sheets, Web, Yahoo, Manual, Mock, and RingBuffer sources all become variations of the same pipeline.



Data source overview. Sources acquire or generate values, normalize them into tabular data, map columns to chart semantics, and emit marks for the shared renderer.

CHOOSING A DATA SOURCE

Choose the source by ownership and refresh behavior. Use Manual when the scene owns the data, Mock when the real backend is not ready, CSV/JSON for files and exports, Sheets for externally edited tabular data, Web for service-backed dashboards, Yahoo Finance for quick OHLC examples, and RingBuffer for high-frequency live samples.



Data source chooser. Each source targets a different ownership and refresh pattern while sharing the same mapping and rendering model.

INDIVIDUAL DATA SOURCES

ManualChartDataSource is best for scene-owned data and Designer handoff. It supports single-series and multi-series values, reference rules, overlay points, overlay rectangles, symbols, colors, gradients, and reusable styling fields.

MockChartDataSource is the recommended starting point for design, demos, and stress tests. It generates patterns with configurable labels, series counts, noise, value ranges, gradients, and streaming rhythms, so a chart can be designed before a production backend exists.

CsvChartDataSource reads separated-value text from a TextAsset, project path, absolute path, StreamingAssets path, or inline string. It can auto-detect the delimiter, header row, and column types, but the inspector can lock those choices down with delimiter, header mode, comment prefix, skip rows, trimming, and file watching options.

JsonChartDataSource is for structured payloads. Use Root Selector when the useful records live inside a larger API response. Nested objects flatten one level with dot notation, which keeps fields such as ohlc.open or metrics.revenue available for normal mapping.

GoogleSheetsChartDataSource converts a published Google Sheets URL into a CSV export URL. Use it when designers or analysts own a sheet outside Unity. The sheet must be published or otherwise reachable by the player/editor, and the gid selects which worksheet tab is read.

WebChartDataSource polls an HTTP or HTTPS endpoint and parses CSV or JSON responses through the same mapping system. It supports request headers, JSON root selection, CSV options, polling intervals, cancellation, edit-mode preview, and exponential backoff on transient failures.

YahooFinanceChartDataSource is a convenience wrapper for the public Yahoo Finance chart endpoint. It fills symbol, range, interval, OHLCV parsing, and candlestick mapping automatically, making it useful for examples and prototypes. For production market dashboards, prefer a paid market-data provider through WebChartDataSource and avoid aggressive polling.

RingBufferChartDataSource is the live-data path for high-frequency runtime samples. Producers push Y or X/Y values into a fixed-capacity buffer; the chart receives a per-frame flush and can use the stable-shape SetValues path instead of rebuilding marks every sample.

MAPPING CONCEPTS

- Wide format means each Y column becomes its own series. This matches spreadsheet exports such as Month, Core, Pro, Enterprise.
- Long format means one Y column is split by a category or series column. This matches rows such as Quarter, Region, Revenue.
- XColumn controls the axis position. If it is empty, the row index becomes the X value. LabelColumn controls category/data labels when present.
- DateFormat and DateCulture stabilize time parsing when the source uses a non-invariant date format or a locale-specific month/day order.
- OpenColumn, HighColumn, LowColumn, and CloseColumn switch the emitted marks to candlesticks when all four are configured.
- MaxRows is a safety valve for very large sources. It caps rows before marks are produced, which is useful for previews and heavy files.

STYLING, LAYOUT, AND LABELS

THEMES

ChartTheme is a ScriptableObject that controls palette, text sizes, axis colors, grid colors, background, plot styling, shadows, glow, animation defaults, and text scaling. Assign a theme in the builder, ChartGraphic, ChartComponent, or ChartGuruElement. Mark opacity applies consistently to bars, points, sectors, and the other rendered mark families.

```
Chart chart = Chart.Create()
    .Theme(myTheme)
    .ChartTitle("Revenue")
    .ChartLegend(LegendPosition.Bottom)
    .Build();
```

CHART AREA AND PLOT AREA

Use the chart area settings for the outer background and border. Use plot area settings for the data region background, border, corner radius, and padding. This distinction matters when you want a card-like chart frame but a clean data plot inside it.

AXES AND SCALES

Axes can be automatic or explicitly configured. Use domains when the chart should keep a stable range. Use categorical or band axes for named categories, time axes for dates, logarithmic or symmetric-log scales for broad numeric ranges, and power scales when a non-linear transform is needed.

```
Chart chart = builder
    .ChartXAxis(x => x.Label("Month").TickCount(6))
    .ChartYAxis(y => y.Label("Revenue").Domain(0, 100).TickFormat("F0"))
    .ChartYScale(0, 100)
    .Build();
```

TITLES, LABELS, AND ANNOTATIONS

Use the chart title for the main message and subtitle for context. Data labels show values on marks. Annotations attach explanatory text to a mark or reference line. RuleMark is the preferred way to add targets, thresholds, or baselines.

```
RuleMark target = RuleMark.Horizontal(100f).LineStyle(new[] { 8f, 4f });
target.ForegroundStyle(Color.green);
target.Annotation(AnnotationPosition.TopLeading, "Target");
builder.Add(target);
```

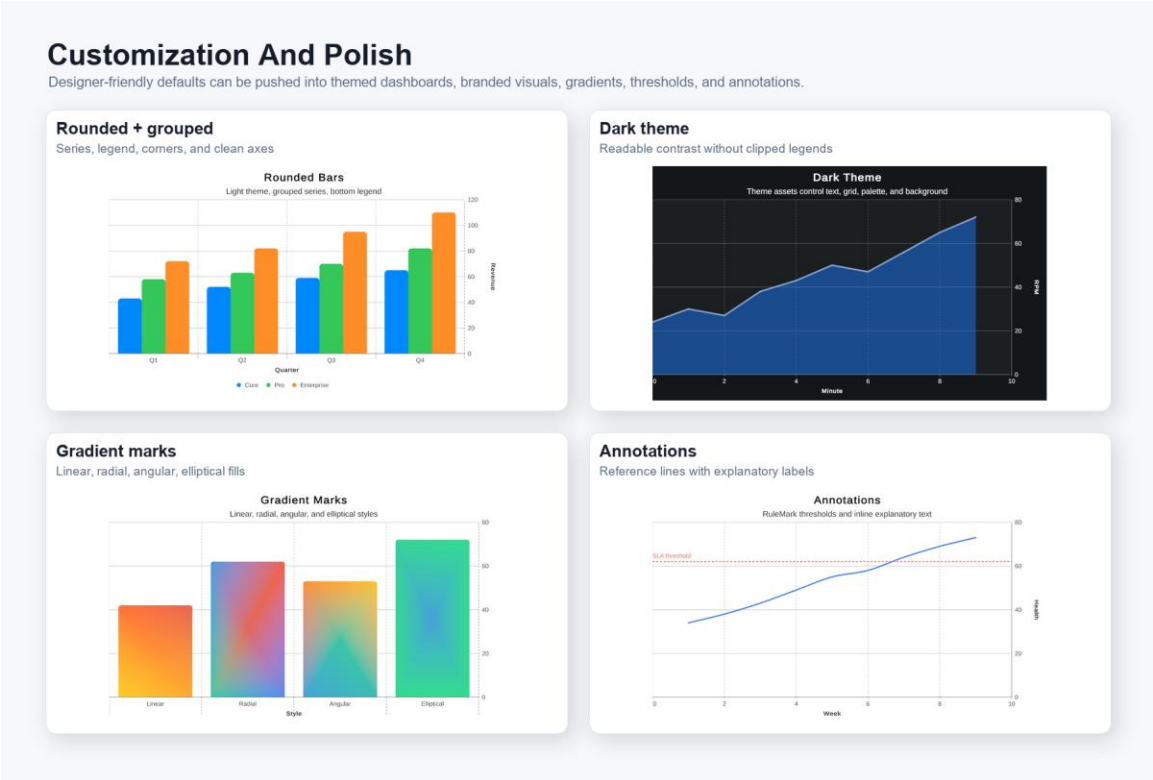


Figure 4. Customization gallery: rounded layouts, dark themes, gradient marks, and target annotations can be mixed without changing the data model.

INTERACTION, ANIMATION, AND LIVE UPDATES

SELECTION AND GESTURES

ChartGraphic implements pointer, hover, drag, and scroll interfaces. The Chart model exposes selected point, selected X or Y value, selected angle, selected ranges, hovered index, and selection indicator styles such as Highlight, Lollipop, and RuleMark.

- Use point selection for dashboards and drilldown.
- Use range selection for time windows and brushing.
- Use angle selection for pie and donut slices.
- Use scrollable axes and visible domains when only part of a large data set should be visible.

ANIMATION AND MORPHING

Animations can run on initial display, value changes, explicit replays, and morph transitions. Use Animation.Default, Linear, EaseIn, EaseOut, EaseInOut, Spring, SpringResponse, Bouncy, Smooth, Snappy, InterpolatingSpring, TimingCurve, or FromEasing, then add Delay, Speed, RepeatCount, or RepeatForever modifiers as needed. Morphing transitions between chart types at runtime and can use CrossFade, Scale, or Semantic styles. Semantic morphing is supported for Bar, MiniBar, Line, Sparkline, Area, Point, Bubble, Scatter, Pie, Donut, MiniPie, MiniDonut, and HeatMap. If either side of a requested semantic morph is outside that set, Chart Guru automatically resolves the transition to Scale so the morph remains visible.

```
chart.MorphTo(ChartType.Line, new MorphOptions
{
    Animation = Animation.EaseInOut(0.6f),
    Style = ChartMorphStyle.Semantic
});
chart.SetValues(newValues);      // fastest Y-only update when shape is unchanged
chart.SetData(newMarks);         // full replace when mark shape changes
```

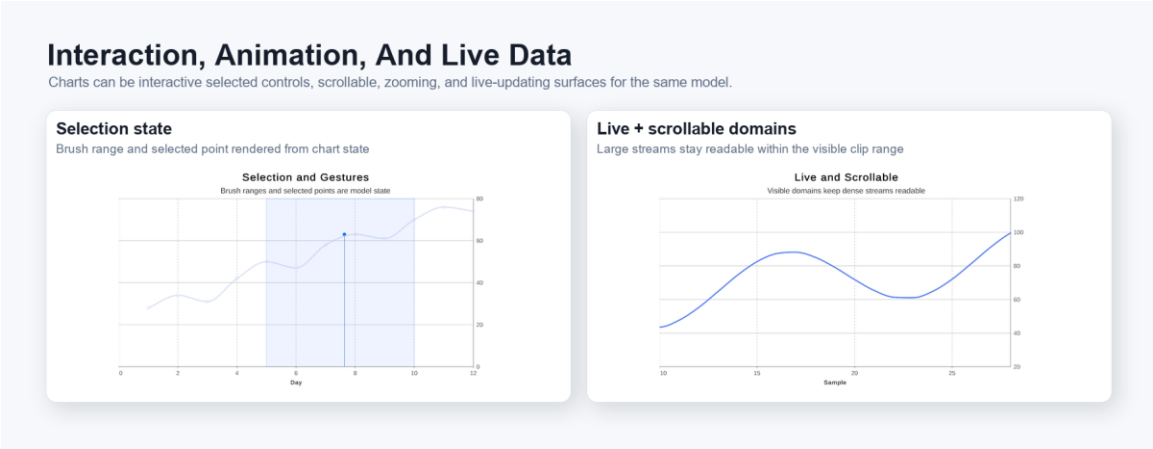


Figure 5. Interaction and live data: selection ranges, selected points, visible domains, scrolling, zooming, and streaming updates share the same chart model.

RENDERING PATHS

All 2D rendering paths share the same Chart model and renderer. Choose the host that matches where the chart appears.

Host	Use it when
ChartGraphic	The chart belongs in a uGUI Canvas, including Screen Space Overlay, Screen Space Camera, or World Space Canvas.
ChartComponent	You want a quick scene component for prototyping without building a full Canvas workflow.
ChartGuruElement	The UI is built with UI Toolkit, UXML, USS, or Unity data binding.
Chart Designer / Chart Gallery	You need an editor preview or a visual authoring surface.

UI TOOLKIT

ChartGuruElement is a VisualElement with UXML attributes for chart type, title, labels, axes, legend, marks, data source, and interaction. Use UIToolkitChartBinder when a scene component should feed a named element inside a UIDocument.

```
<ui:UXML xmlns:cg="ChartGuru.UIToolkit">
    <cg:ChartGuruElement
        chart-type="Line"
        chart-title="Sales Trend"
        legend-position="Bottom"
        show-data-labels="true"
        animate-on-start="true" />
</ui:UXML>
```

EDITOR TOOLS

CHART GALLERY

Open Tools > Chart Guru > Chart Gallery. The responsive gallery groups live preview cards by chart family and supports search, tags, multi-series previews, and stacked previews. Open a card in the Designer when you want to inspect or adapt the exact configuration.

CHART DESIGNER

Open Tools > Chart Guru > Chart Designer. The responsive WYSIWYG workspace combines live preview, aligned data tables, focused style and gradient controls, axes, legend, actions, validation, theme creation, and C# export. The same chart state can be created in the scene or continued through generated code.

SWIFTUI CHART IMPORTER

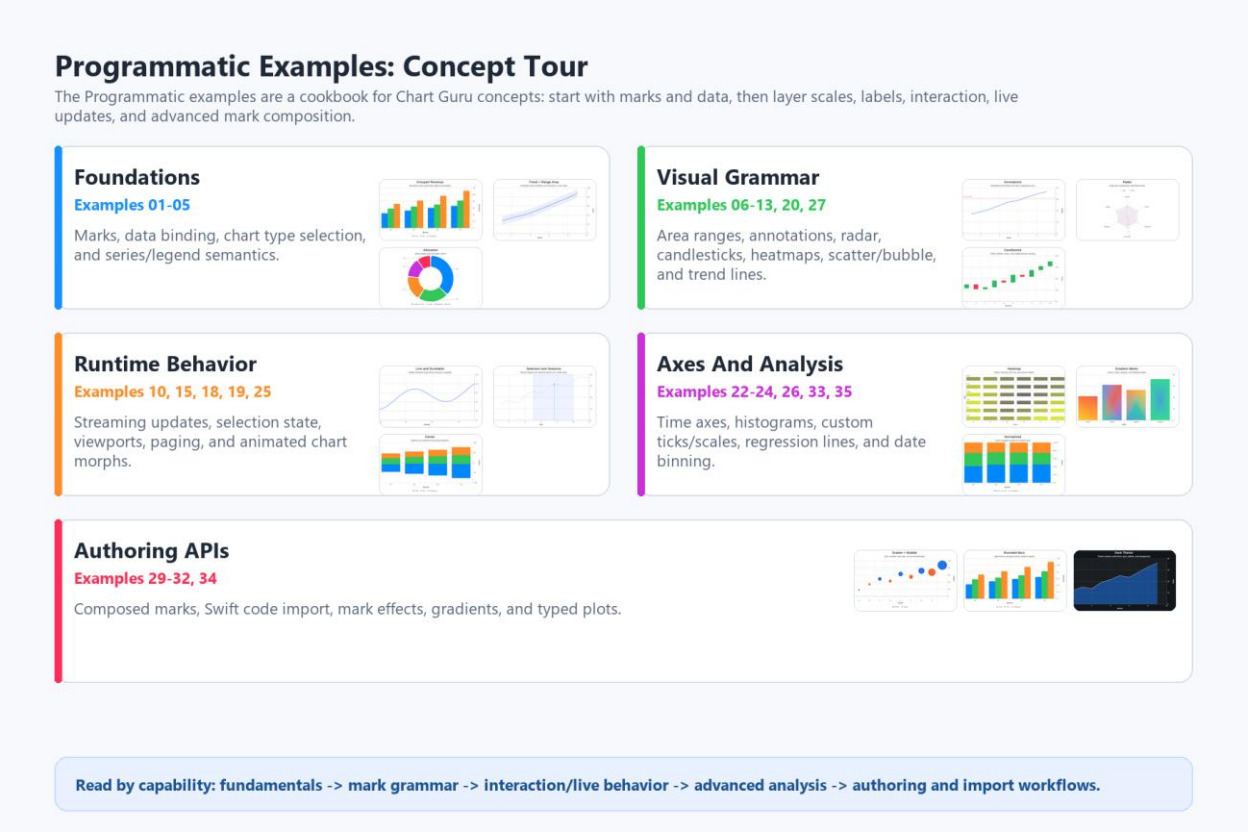
The importer translates common Swift Charts snippets into Chart Guru builder code. It recognizes BarMark, LineMark, AreaMark, PointMark, SectorMark, RuleMark, semantic foreground style, symbols, symbol sizes, interpolation, custom scales, legends, function-style ideas, vectorized plot patterns, and common ForEach/enumerated loops.

CREATE SIMPLE CHART AND UXML TEMPLATE

Use Tools > Chart Guru > Create Simple Chart to create a ready-to-run scene chart. Use Assets > Create > ChartGuru > UXML Chart Template to create a starter UI Toolkit chart template.

PROGRAMMATIC EXAMPLES AND CONCEPTS

The Programmatic examples are the fastest way to understand Chart Guru as a chart language rather than a collection of chart prefabs. Read them as a sequence: define marks, bind data, add semantic encodings, tune axes and labels, then layer runtime behavior such as selection, scrolling, streaming, and morphing.



Programmatic examples grouped by concept instead of by chart type.

HOW TO READ THE PROGRAMMATIC SCENES

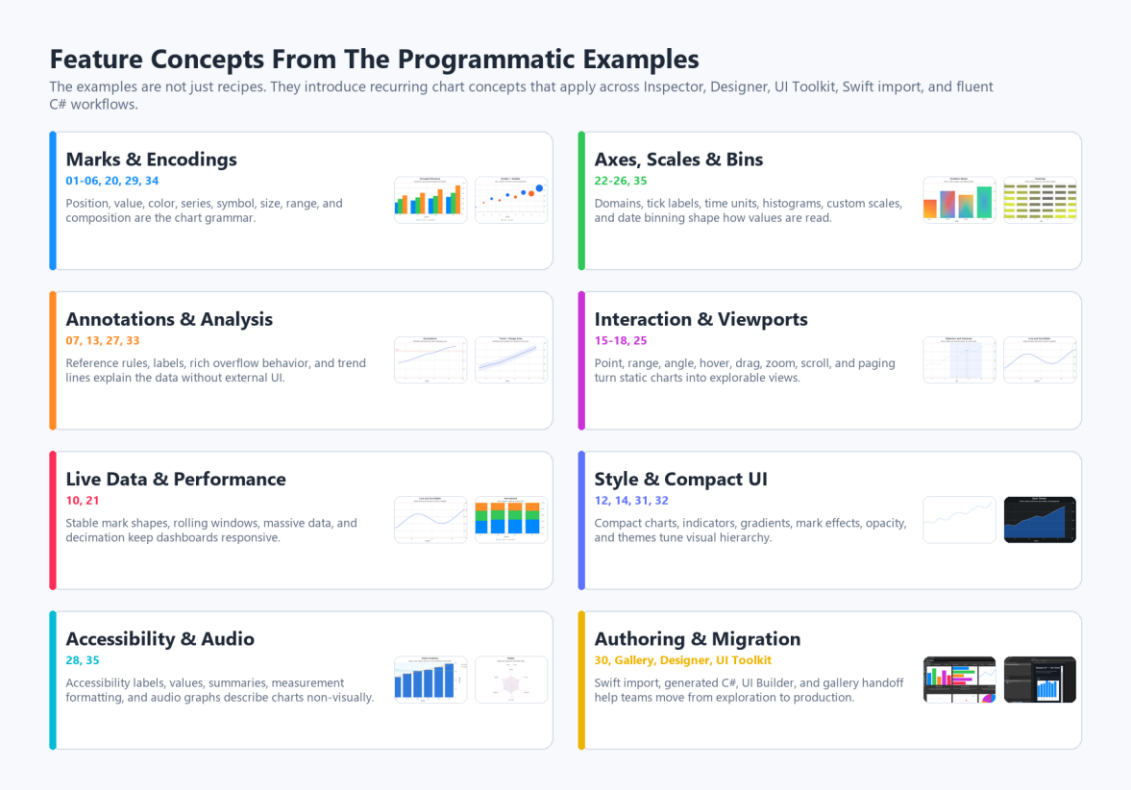
- Start with simple bar, 2D data binding, line chart, pie/donut, and multi-series scenes to learn marks, labels, series, and legend models.
- Use area, range, annotations, rule marks, axis customization, heatmap, scatter/bubble, and radar scenes to see how marks encode position, region, size, color, and context.
- Use live, streaming, viewport, animation, transition, paging, and morphing scenes to understand how charts update while preserving interaction state.

- Use Swift code, typed plots, gradients, mark effects, and composed marks when moving from simple samples into reusable chart grammar and migration workflows.

The examples deliberately overlap concepts. A chart type is only the starting point; the important part is how the example combines marks, data sources, axes, styling, interaction state, and rendering targets.

FEATURE CONCEPTS GUIDED BY THE EXAMPLES

The Programmatic examples are also a concept map. Use them to learn how features fit together, not only how to type a builder call. The same concepts apply whether the chart is made in the Inspector, Designer, Gallery, UI Toolkit, Swift import, or fluent C#.



Feature concept atlas. Programmatic examples grouped by the recurring concepts they teach across authoring workflows.

CONCEPTS TO LOOK FOR WHILE READING EXAMPLES

- Marks and encodings: BarMark, LineMark, AreaMark, SectorMark, PointMark, RectangleMark, RuleMark, and composed marks describe what is drawn. Encodings such as X, Y, ForegroundStyle, Symbol, size, label, and series describe what the data means.
- Chart type versus mark grammar: chart type chooses the default visual family, but marks still carry the semantic intent. This is why composed charts can combine bars, lines, points, rules, overlays, and annotations in one chart.
- Axes and scales: domains, ticks, labels, time units, custom scale transformers, range padding, and binning decide how values are read. These concepts matter as much as the visual chart type.
- Annotations and analysis: reference rules, trend lines, rich labels, overflow behavior, and accessibility descriptions turn a chart from a picture into an explanation.
- Interaction and viewports: selection, brushing, angle selection, hover, drag, zoom, scrolling, paging, and visible domains create explorable charts without adding separate UI controls.
- Live data and performance: use stable mark shapes, SetValues, ring buffers, rolling windows, LOD, and decimation when data changes frequently or arrives at high volume.
- Styling and compact UI: themes, gradients, mark effects, opacity, symbol choices, compact charts, indicators, and data labels tune information hierarchy for dashboards and repeated UI cells.
- Accessibility and sonification: measurement formatting, per-mark labels/values, chart summaries, trend descriptions, and audio graphs help charts communicate beyond the visual rendering.

When a feature is unclear, find the closest Programmatic example first, then decide whether to reproduce it in code, adapt it in the Designer, import a matching Swift Charts snippet, or express it as a ChartGuruElement in UI Toolkit.

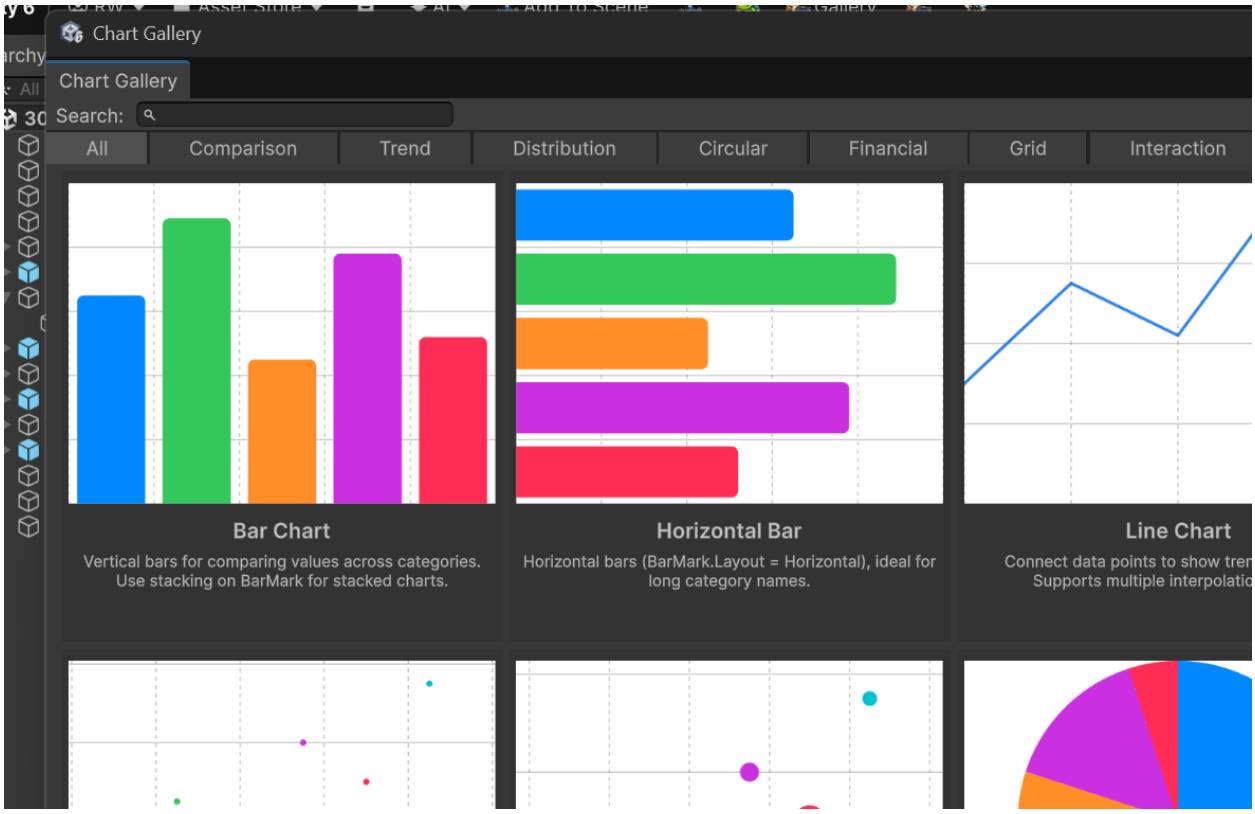
EXPANDED AUTHORING WORKFLOWS

Chart Guru can be authored from the Inspector, Gallery, Designer, fluent C#, Swift Charts snippets, and UI Toolkit. The best workflow depends on whether you are exploring a visual idea, building a scene chart, migrating Swift Charts code, or embedding charts into a UI Toolkit application.

These authoring paths share one chart model, so an idea can begin as a visual experiment, a familiar Swift Charts example, or declarative C# and still become maintainable production output without being rebuilt in another system.

GALLERY AS A CONCEPT BROWSER

Open Tools > Chart Guru > Chart Gallery to browse curated examples by category. The Gallery is useful before writing code because it shows compact, comparison, trend, distribution, circular, financial, grid, interaction, annotation, styled, and function examples with live preview renders.

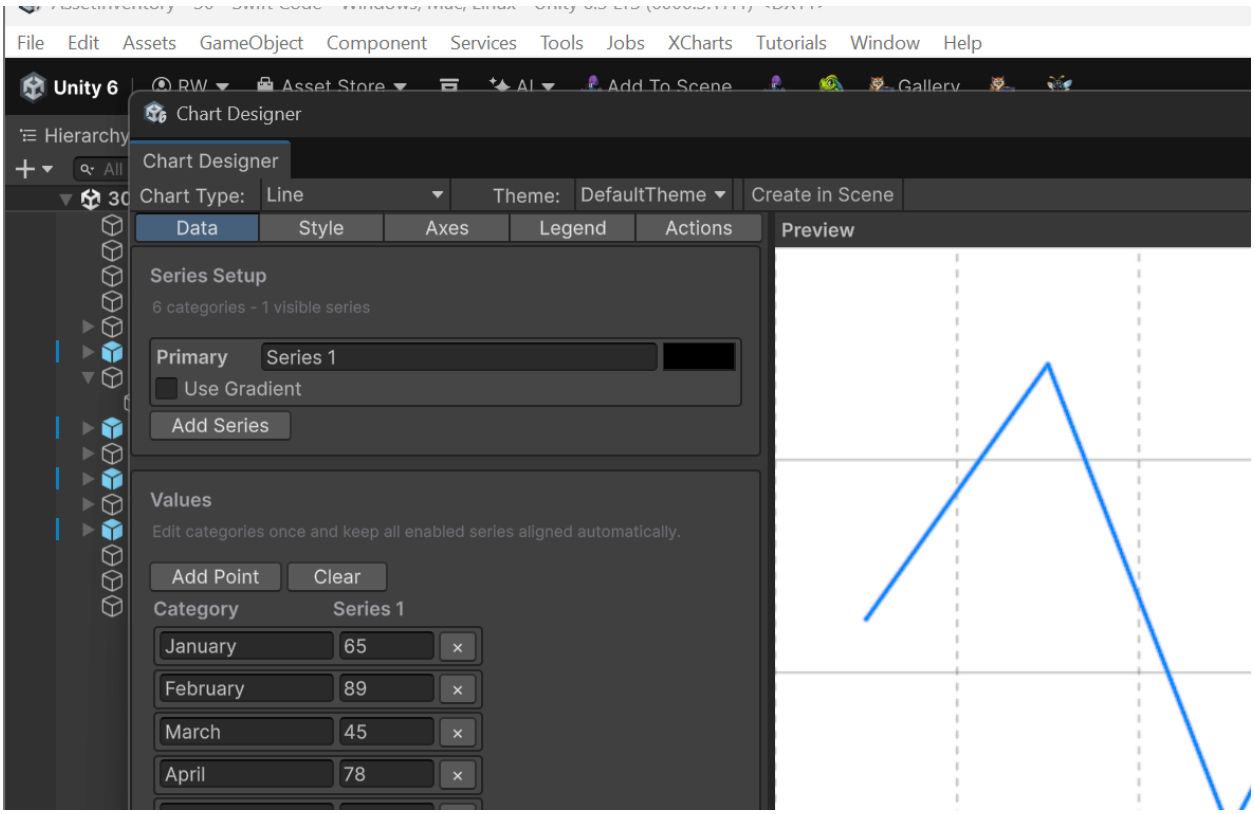


The Chart Gallery groups live chart previews by category and supports search, descriptions, multi-series previews, and stacked previews.

- Use Search when you know the concept name, such as trend, range, selection, or viewport.
- Use the category tabs when choosing a visual form for a problem: comparison, trend, distribution, circular, financial, grid, interaction, annotations, styled, function, or compact.
- Toggle multi-series and stacked previews before deciding whether a chart family behaves with multiple series before opening it in the Designer.
- Open a gallery card in the Designer when you want to inspect or adapt the exact chart configuration.

DESIGNER WORKFLOW

Open Tools > Chart Guru > Chart Designer for visual chart creation. The Designer is not a separate chart model; it edits the same chart state used by ChartGraphic, ChartGuruElement, and generated C#.



The Chart Designer edits data, style, axes, legends, actions, preview data, generated code, and save workflows from one editor window.

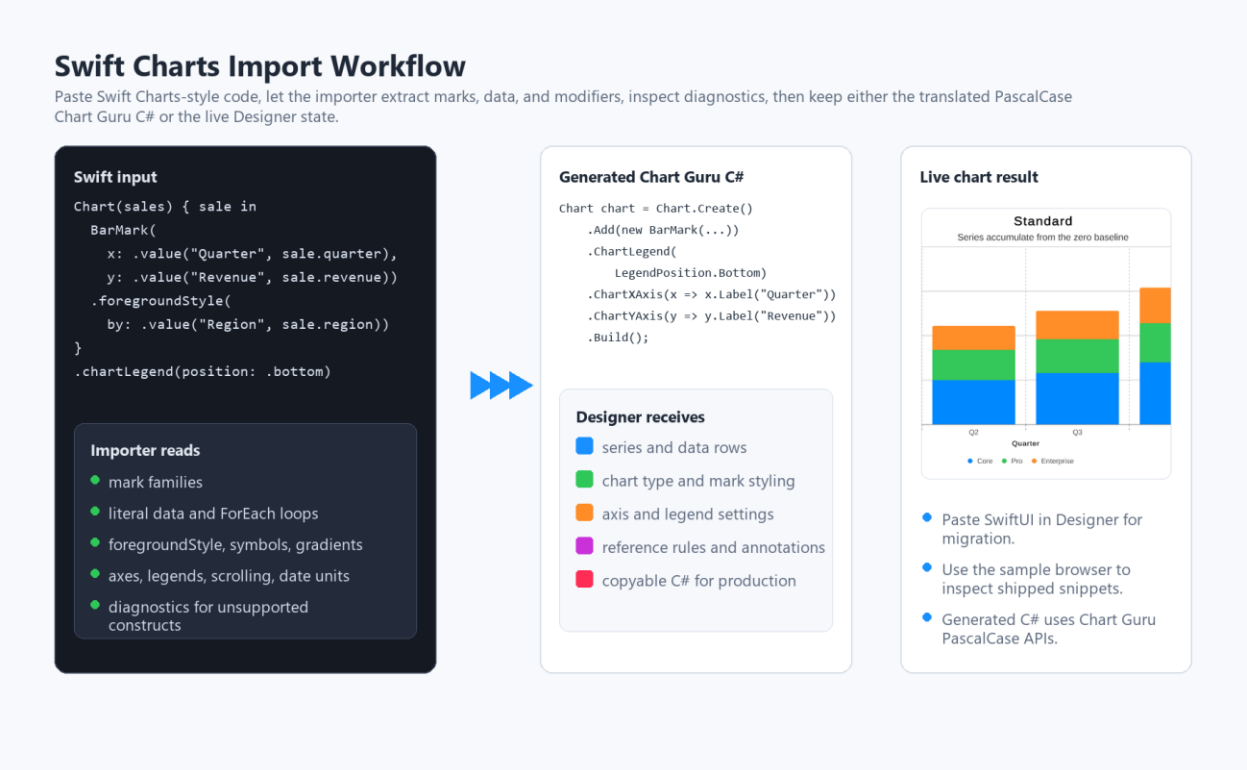
- Data controls define categories, values, series names, colors, and generated sample data for chart families that need special data shapes such as heatmaps or candlesticks.

- Style controls cover mark shape, interpolation, opacity, gradients, plot area styling, font choices, axis labels, and chart area layout.
- Axes controls expose domains, scale, tick count, formatting, label orientation, collision behavior, range padding, and origin alignment.
- Legend controls set position and alignment while actions controls expose animation, morphing, selection, and interaction-oriented preview behavior.
- Create in Scene instantiates a configured chart graphic in the active scene, while Show Code produces equivalent C# for production use.

SWIFT CHARTS IMPORT AND PARSING

The Swift Charts importer is intended for prototyping and migration. Paste Swift Charts-style code in the Designer or browse the shipped Swift samples in Example 30. The importer accepts real Swift framing such as variables, foregroundStyle, chartLegend, annotation, and BarMark/SectorMark/LineMark declarations while emitting clean Chart Guru C# using PascalCase APIs.

This makes the wider Swift Charts ecosystem a practical source of ideas: paste supported chart-focused code, inspect the translated result, and continue in idiomatic Chart Guru C#. Diagnostics keep partial or unsupported constructs visible rather than pretending that arbitrary Swift application logic can be converted automatically.

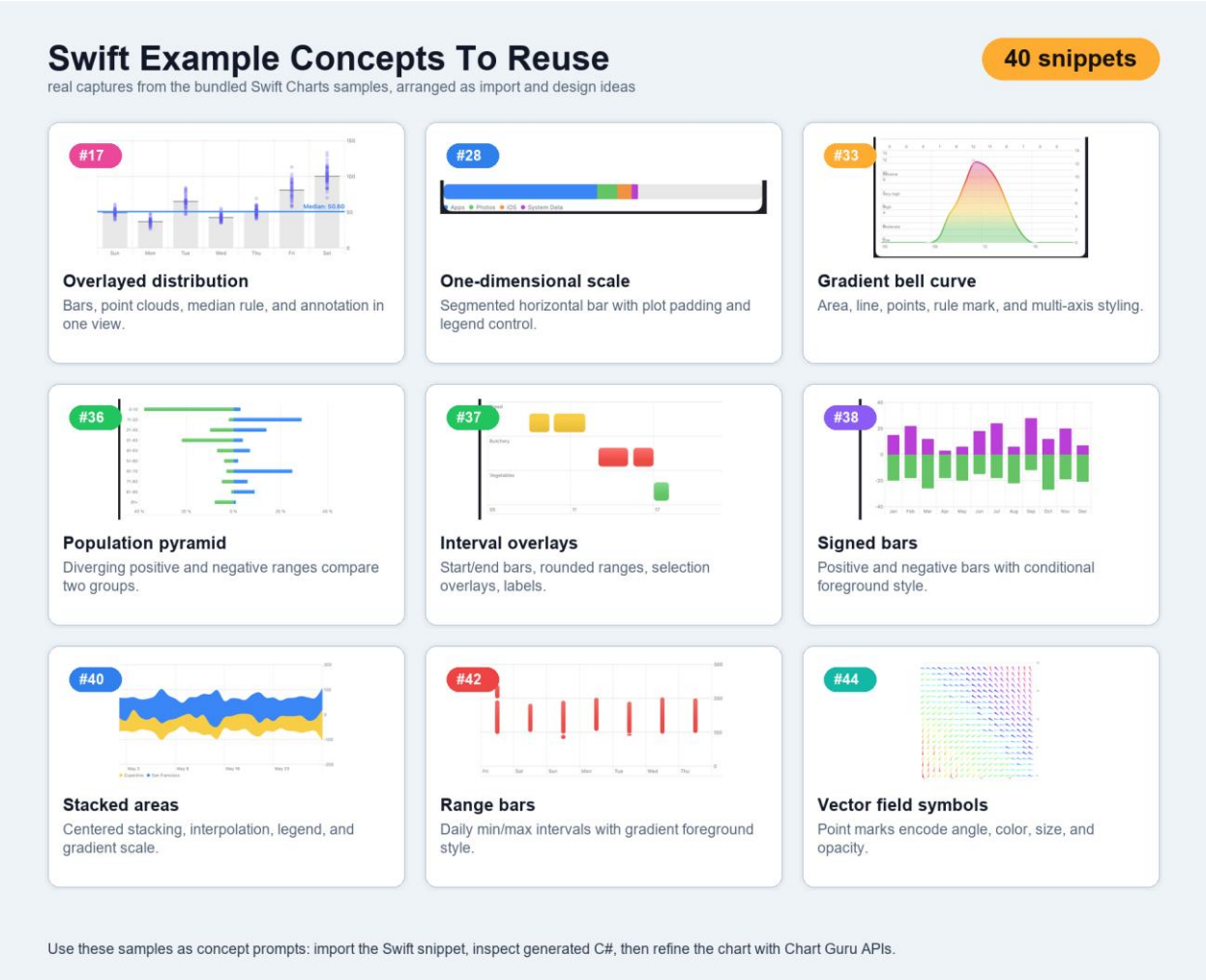


Swift Charts import parses supported marks, literal data, modifiers, axes, legends, and diagnostics, then produces Designer state and C#.

- Supported input is chart-focused Swift code with concrete marks and extractable literal data. Dynamic app logic should be translated manually or stubbed before import.
- Supported concepts include common marks, foreground style, gradients, symbols, stacking, legends, axis visibility and formatting, scroll position, visible domains, date units, annotations, reference rules, trendline ideas, function plots, and vectorized typed plot helpers.
- Diagnostics explain unsupported or partially supported constructs so the converted result can be reviewed instead of silently misrepresenting the original chart.
- Generated code keeps the Swift mark structure as close as practical while using idiomatic Chart Guru C# names and avoiding unnecessary default modifiers.

ADVANCED SWIFT EXAMPLE CONCEPTS

Example 30 includes 40 Swift Charts snippets that work as a concept atlas before you build production charts. Use the sample browser to inspect layered marks, custom axes, overlays, data loops, scale modifiers, function-style plot ideas, vectorized symbol fields, and dense encodings, then import the closest Swift structure and refine the generated Chart Guru C# where needed.

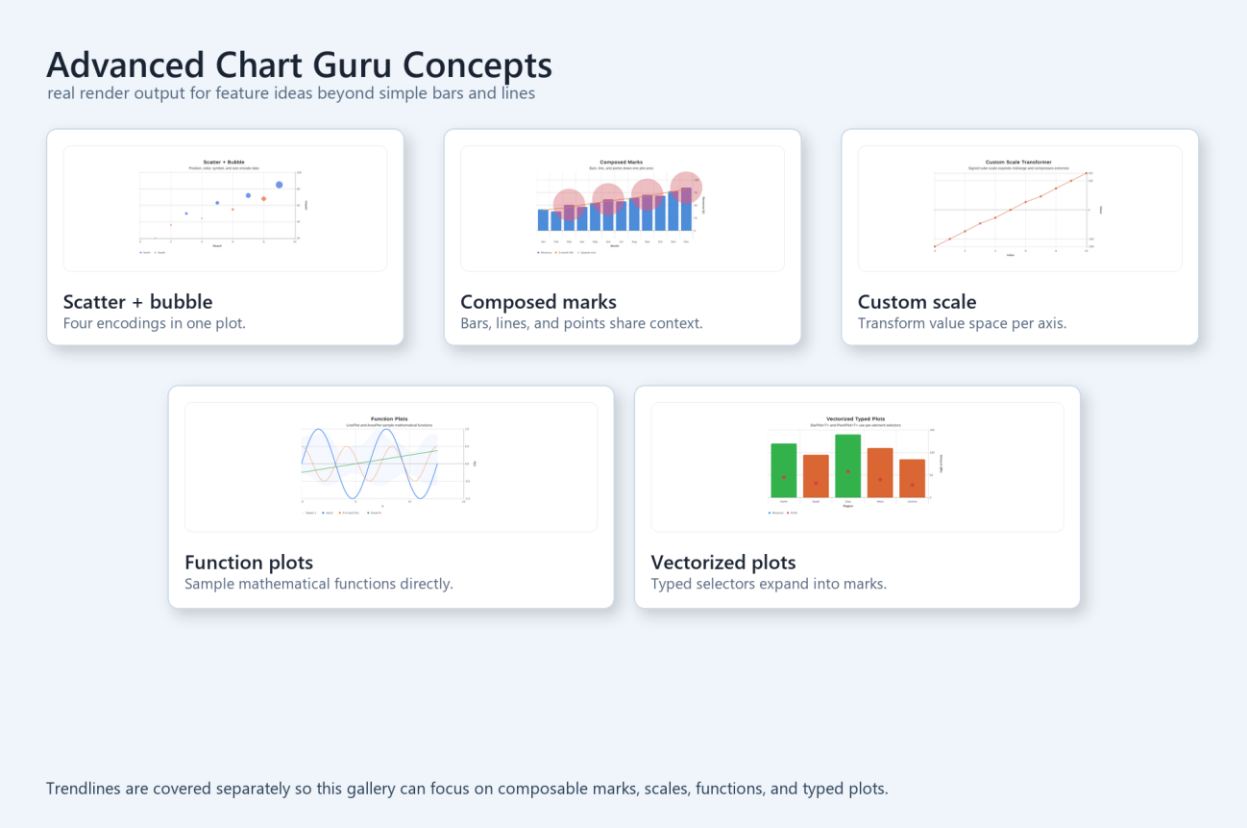


Advanced Swift example concepts from Example 30, including overlayed distributions, one-dimensional scales, multi-axis gradients, population pyramids, interval overlays, signed bars, stacked areas, range bars, vector-style point symbols, threshold rules, and lollipop selection.

- Layer marks: combine AreaMark, LineMark, PointMark, RectangleMark, and RuleMark so gradients, peaks, thresholds, and selections describe the same data.
- Compose axes: translate top/bottom X axes, leading/trailing Y axes, custom labels, grid lines, and labelled bands into ChartXAxis, ChartYAxis, and axis builder calls.
- Use overlays: pair chartOverlay and ChartProxy patterns with RuleMark, lollipop symbols, and annotations for tap or drag inspection.
- Encode dense data: use foreground style, symbol size, opacity, interpolation, stacking, start/end ranges, function plots, and vectorized typed plots to carry extra dimensions without adding separate charts.
- Refine custom Swift ideas: snippets with app-specific ChartSymbolShape code are useful starting points; translate the chart structure first, then finish the symbol, mark, function/vectorized plot, or renderer customization in Chart Guru.

SUPPORTED SWIFT CONSTRUCTS AND MODIFIERS

Marks: BarMark, LineMark, AreaMark, PointMark, RectangleMark, RuleMark, SectorMark, candlestick/range concepts, and composed mark groups.
Data loops: Chart(data), ForEach, enumerated(), grouped/nested loops, and bundled sample-data resolution.
Encodings and modifiers: x/y/start/end channels, foreground style, symbol, symbol size, opacity, interpolation, line style, corner radius, gradients, and legends.
Axes and scales: chartXScale, chartYScale, domains/ranges, top/bottom and leading/trailing axes, AxisMarks, grid lines, ticks, labels, custom-scale equivalents, and modifyInferredDomain ideas.
Analysis and interaction: reference rules, annotations, chartOverlay/ChartProxy patterns, lollipop inspection, accessibility labels/values, trendline/regression concepts, mathematical function plots, and vectorized typed plot helpers.

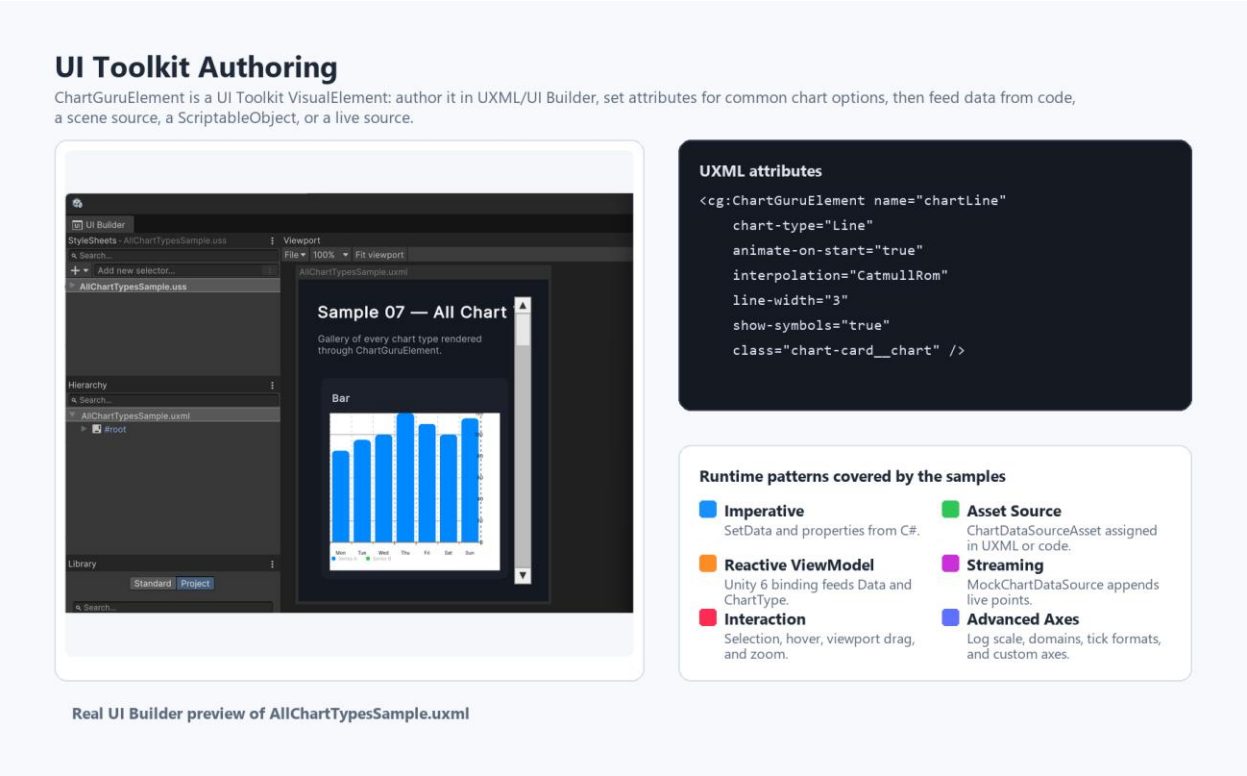


Advanced Chart Guru concepts: scatter/bubble encodings, composed marks, custom scales, mathematical function plots, and vectorized typed plots.

TrendLinePlot regression modes from Example 33: linear, polynomial, exponential, logarithmic, and power fits with sampled regression overlays.

UI TOOLKIT DESIGNER AND UXML AUTHORING

ChartGuruElement is a VisualElement with UXML attributes for chart type, title, labels, axes, legend, marks, data source, and interaction. Use UI Toolkit/Scriptable charts as scene component model feeds a render element inside a UIDocument.



UI Builder preview of a ChartGuruElement UXML sample plus the UXML/runtime patterns used by the UI Toolkit examples.

- Use Assets > Create > ChartGuru > UXML Chart Template to create a ready-to-edit ChartGuruElement template.
- Use UXML attributes for stable visual configuration such as chart type, axis visibility, labels, interpolation, symbols, colors, interaction, legend state, viewport, zoom, animation, and visible domain lengths.
- Use USS for surrounding layout and UI composition. Chart rendering itself comes from ChartGuruElement and the shared chart Guru renderer.
- Study the UI Toolkit examples for imperative setup, asset-backed data, scene data source, runtime updates, SetValues streaming data, interaction, theming, advanced axes, and external data source binding.

PERFORMANCE GUIDANCE

- Prefer SetValues when only Y values change and the mark count stays the same.
- Hide dense data labels on large charts. Text is pooled, but thousands of visible labels are still harder to read and render.
- Use compact charts for small dashboard signals instead of full axes and legends.

- Use RingBufferChartDataSource for frequent live samples instead of rebuilding a chart every event.
- Keep RenderTexture size close to the displayed chart size. Oversized charts cost memory and fill rate.
- Let LOD and decimation handle very dense line or candlestick data where possible.
- Reuse ChartGraphic and Chart instances when updating dashboards instead of destroying and recreating UI objects.

TROUBLESHOOTING

Problem	Check
Chart is blank	Confirm the GameObject has ChartGraphic or another host, has a valid size, and has marks from code or a data source.
No data appears from a source	Confirm the data source targets the correct chart, then click Generate or Refresh. For file/web sources, inspect ChartDataMapping.
Axes look wrong	Check chart type, bar orientation, domain, scale type, category labels, and whether the chart type hides axes by design.
Legend is missing	A legend needs series or foreground style keys. Use semantic ForegroundStyle values or set LegendPosition to a visible value.
Labels overlap	Reduce visible labels, rotate axis labels, shorten category names, use a wider chart, or use label collision settings.
Live chart allocates or stutters	Prefer SetValues or RingBufferChartDataSource, reduce label count, disable overlapping animations, and avoid recreating chart objects every frame.
UI Toolkit chart does not update	Confirm ChartGuruElement is bound to the intended data source or binder, and that the data source raises DataChanged after refreshing.

API QUICK REFERENCE

CHART TYPES

2D ChartType values: Bar, Line, Point, Area, Pie, Donut, Radar, Bubble, Scatter, HeatMap, Candlestick, Sparkline, MiniBar, MiniPie, MiniDonut, Indicator.

COMMON BUILDER CALLS

```
Chart.Create()
Chart.Create<T>(data, item => new BarMark(...))
builder.Add(mark)
builder.ChartType(ChartType.Bar)
builder.Theme(theme)
builder.ChartTitle("Title").ChartSubtitle("Subtitle")
builder.ChartXAxis(axis => axis.Label("X"))
builder.ChartYAxis(axis => axis.Label("Y"))
builder.ChartXScale(min, max).ChartYScale(min, max)
builder.ChartLegend(LegendPosition.Bottom)
builder.ChartScrollableAxes(ScrollableAxes.Horizontal)
builder.ChartXVisibleDomain(length)
builder.ChartForegroundStyleScale(dictionaryOrColorScale)
builder.ChartSymbolScale(dictionary)
builder.ChartSymbolSizeScale(new ClosedRange<float>(4f, 24f))
builder.ShowDataLabels()
builder.Animation(Animation.Default)
builder.NoAnimation()
builder.Build()
```

COMMON RUNTIME CALLS

```
chart.SetValues(values);
chart.SetData(marks);
chart.AddMark(mark);
chart.ClearMarks();
chart.MorphTo(ChartType.Pie);
chart.ClearSelection();

ChartGraphic graphic = ChartCanvasHelper.RenderToRectTransform(chart, rect, true, "Chart");
ChartCanvasHelper.RemoveChart(graphic);
```

SUPPORT

Web: https://www.wetzold.com/tools

Discord: https://discord.gg/uzeHzEMM4B