MOCK MAGIC
GUIDE - VERSION 1.0.0

INTRODUCTION

Mock Magic is a zero-code mock data generator for Unity. It produces realistic fake data - names, emails, addresses, time series, streaming feeds, and 50+ other data types - directly from an Editor window. No scripting required.

Open **Tools** → **Mock Magic** → **Generator**, pick a data type, and copy the results. That's it.

Mock Magic covers three domains of test data:

- **Simple data** - individual values like names, emails, phone numbers, GUIDs, lorem ipsum, and more (51 types in total).
- **Series data** - patterned datasets for charts, graphs, and tabular testing (14 mathematical patterns with configurable noise, labels, and multi-series support).
- **Streaming data** - continuous real-time data emission with configurable rhythms, usable in both Edit Mode and Play Mode.

All generated data can be exported to **Text**, **CSV**, **JSON**, or **C# code** and copied to the clipboard with a single click. For developers who need runtime data generation, Mock Magic also offers a clean scripting API.

CONTENTS

KEY FEATURES

Feature	Description
51 data types	Personal info, contact details, locations, business data, dates, text, numbers, technical identifiers, and more
14 series patterns	Random, Linear, Exponential, Logarithmic, Sine, Cosine, TrendingUp, TrendingDown, Seasonal, Stepped, Sparse, RandomWalk, BellCurve, Sawtooth
17 label presets	Months, Weekdays, Quarters, Years, Products, Regions, Teams, Departments, Countries, Cities, Colors, Letters, and more
4 streaming rhythms	Constant, Random, Burst, Drip - with full timing control
4 export formats	Text, CSV, JSON, C# code - one-click clipboard copy
Visual Editor Window	Four-tab interface: Simple, Series, Streaming, Export - no code needed
Edit Mode & Play Mode	Streaming works in both modes - test without entering Play Mode
Chart Guru integration	Chart Guru includes a built-in MockChartDataSource component powered by Mock Magic

SWIFT CHARTS PARITY

Mock Magic's data structures are designed to map directly to charting frameworks like Apple's Swift Charts and Chart Guru. A MockDataPoint with Label, Value, and SeriesName translates one-to-one to a BarMark, LineMark, or AreaMark. The 14 series patterns cover every standard chart scenario - from simple bar charts (Random, Stepped) to financial dashboards (RandomWalk) to scientific plots (BellCurve, Sine). Generate the data in Mock Magic, export it, and use it directly.

GETTING STARTED

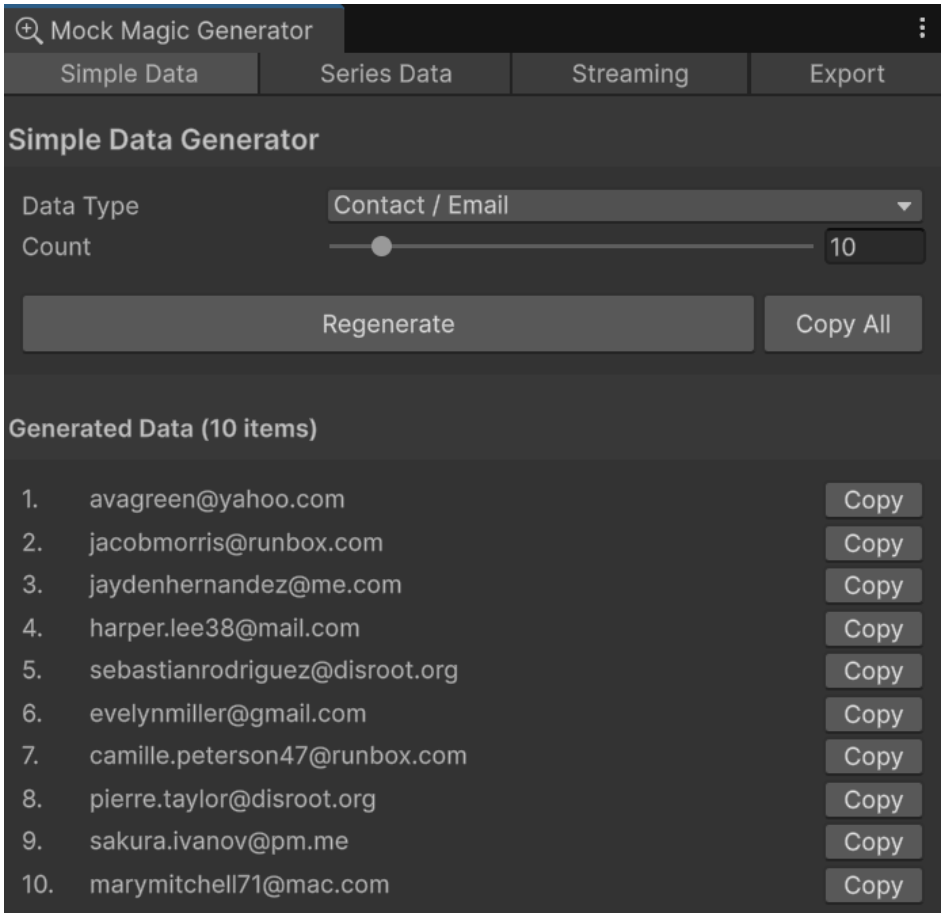
Installation

1. Install MockMagic from the Package Manager.
2. Dependencies are resolved automatically.
3. Verify installation: open **Tools** → **Mock Magic** → **Generator** from the Unity menu bar.

Your First Mock Data - 30 Seconds

1. Open **Tools** → **Mock Magic** → **Generator**.
2. The **Simple Data** tab is selected by default.
3. Click the **data type dropdown** - a categorized menu appears with groups like Personal, Contact, Location, and more.
4. Select **Contact** → **Email**.
5. Click **Copy All** to copy everything to your clipboard.

That's it. You now have 10 fake email addresses ready to paste wherever you need them.



The Four Tabs

The Mock Magic Generator window has four tabs, each serving a distinct purpose:

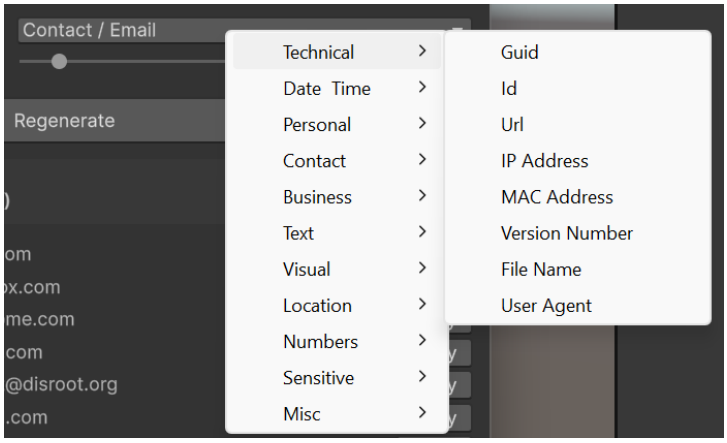
Tab	Purpose
Simple Data	Generate individual fake values - names, emails, dates, colors, UUIDs, and 45+ more types
Series Data	Generate patterned datasets for charts and graphs - trending, seasonal, bell curves, etc.
Streaming	Emit mock data continuously in real-time - test live dashboards and feeds
Export	Format and copy generated data as Text, CSV, JSON, or C# code

SIMPLE DATA

The Simple Data tab generates individual fake values or batches of values from 51 built-in data types. Every value is generated from realistic embedded datasets - names are multicultural, emails have plausible domains, addresses follow real formats.

Using the Simple Data Tab

1. **Data type dropdown** - Click to open a categorized menu. Types are grouped into 11 categories for easy discovery.
2. **Count slider** - Set how many values to generate (1–100).
3. **Regenerate** - Click to generate a new batch. Data also regenerates automatically when you change the type or count.
4. **Copy All** - Copies all generated values to the clipboard, separated by newlines.
5. **Preview list** - A scrollable list showing all generated values, numbered. Each item has its own **Copy** button for copying a single value.



Data Type Reference
All 51 data types, organized by category:

Personal

Type	Example Output
Name	Sarah Johnson
FirstName	Marcus
LastName	Patel
Age	34
Honorific	Dr.
Username	tech_sarah42
Password	kR7#mP2xLq9&

Contact

Type	Example Output
Email	marcus.patel@outlook.com
Phone	+1 (425) 738-2914

Location

Type	Example Output
Address	1847 Oak Ave
FullAddress	1847 Oak Ave, Seattle, WA 98101
City	Toronto
State	California
Country	Japan
ZipCode	90210
Coordinate	(47.6062, -122.3321)
Latitude	47.6062
Longitude	-122.3321

Business

Type	Example Output
Company	CloudPulse Technologies
JobTitle	Senior Product Designer
ProductName	Premium Wireless Headphones
Price	$49.99
Rating	4.3

Date & Time

Type	Example Output
Date	2025-08-14
Time	14:32:07
DateTime	2025-08-14 14:32:07
Timestamp	1723641127

Text

Type	Example Output
Word	quantum
Sentence	The quick algorithm processed every node efficiently.
Paragraph	*(Multiple sentences of lorem ipsum-style text)*
LoremIpsum	Lorem ipsum dolor sit amet consectetur...

Numbers

Type	Example Output
Integer	42
Float	73.284
Percentage	87.45
Currency	1249.99
Boolean	True

Visual

Type	Example Output
Color	RGBA(0.42, 0.78, 0.15, 1.0)
HexColor	#6BC72A

Type	Example Output
Emoji	🎮

Technical

Type	Example Output
Guid	a3f8b2c1-d4e5-4f6a-8b9c-0d1e2f3a4b5c
Id	1 *(auto-incrementing)*
Url	https://www.cloudpulse.io
Domain	techpulse.io
IpAddress	192.168.42.17
MacAddress	A4:2B:8C:D3:E5:F1
UserAgent	Mozilla/5.0 (Windows NT 10.0; Win64; x64)...
FileName	report_2024.pdf
VersionNumber	2.14.3

Sensitive

Type	Example Output
CreditCard	4532-0151-2843-9671
Ssn	428-51-7293
Isbn	978-3-16-148410-0

Miscellaneous

Type	Example Output
Animal	Red Fox
Hashtag	#gamedev

Note: All generated data is fake. Credit card numbers pass the Luhn check but are not real. ISBN check digits are valid. Passwords use Fisher-Yates shuffling with guaranteed uppercase, lowercase, digit, and special character.

Tips

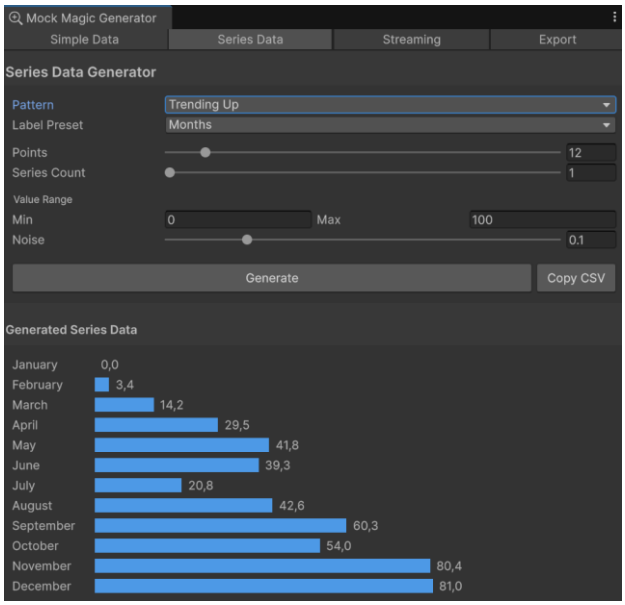
- **Combine types for realistic records.** Generate 10 names, 10 emails, and 10 cities separately, then pair them together for a mock user database.
- **Use Copy All for bulk pasting.** The clipboard output is newline-separated - paste directly into spreadsheets, text files, or code editors.
- **Adjust the count slider** for anything from a single value (quick placeholder) to 100 items (stress testing).

SERIES DATA

The Series Data tab generates patterned numerical datasets - the kind of data you feed into charts, graphs, dashboards, and data visualizations. Each dataset consists of labeled data points with values that follow a mathematical pattern.

Using the Series Data Tab

1. **Pattern** - Select from 14 mathematical patterns (see Pattern Reference below).
2. **Label Preset** - Choose how data points are labeled on the X-axis: months, weekdays, quarters, numeric indices, and more.
3. **Points** - Set how many data points per series (3–100).
4. **Series Count** - Generate multiple series at once (1–10). Useful for grouped/stacked charts or comparisons.
5. **Min / Max** - Define the value range for generated data.
6. **Noise** - Add randomness to the pattern (0 = perfectly clean, 0.5 = very noisy). Noise makes data look more realistic.
7. **Generate** - Click to produce the dataset. A visual bar preview appears below.
8. **Copy CSV** - Copies the generated data in CSV format to the clipboard.



Visual Preview

After generating, the Series Data tab displays an inline horizontal bar chart. Each data point shows its label, a colored bar proportional to the value, and the numeric value. For multi-series data, series are separated by name headers.

The Data Point Structure
Every generated series data point is a MockDataPoint with five fields:

Field	Description
Label	The X-axis category label (e.g., "Jan", "Q1", "North")
Value	The primary Y-axis value
SecondaryValue	An optional secondary value, useful for range/band charts
SeriesIndex	Which series this point belongs to (0-based)
SeriesName	The name of the series (e.g., "Revenue", "Series 1")

Scripting: Access these fields directly - point.Label, point.Value, point.SeriesName, etc.

Pattern Reference

Pattern	Description
Random	Uniformly random values between min and max. Good for bar charts and scatter data.
Linear	Steady progression from min to max. A clean diagonal line.
Exponential	Slow start that accelerates sharply upward. Growth curves and compound metrics.
Logarithmic	Fast initial rise that flattens out. Diminishing returns, learning curves.
Sine	Smooth wave oscillating around the midpoint. Cyclic patterns, audio-like data.
Cosine	Same as Sine but phase-shifted by 90°. Starts at the peak.
TrendingUp	General upward movement with random walk noise. Revenue growth, user acquisition.
TrendingDown	General downward movement with random walk noise. Decline metrics, churn.
Seasonal	Sine wave combined with a linear upward trend. Monthly sales with yearly seasonality.
Stepped	Staircase pattern with flat plateaus. Pricing tiers, plan upgrades.
Sparse	70% zero values, 30% random non-zero spikes. Event data, error logs.

Pattern	Description
RandomWalk	Cumulative random steps from the midpoint. Stock prices, financial data.
BellCurve	Normal distribution shape centered at the midpoint. Survey results, test scores.
Sawtooth	Repeating ramp-up pattern. Periodic resets, charge/discharge cycles.

Label Preset Reference

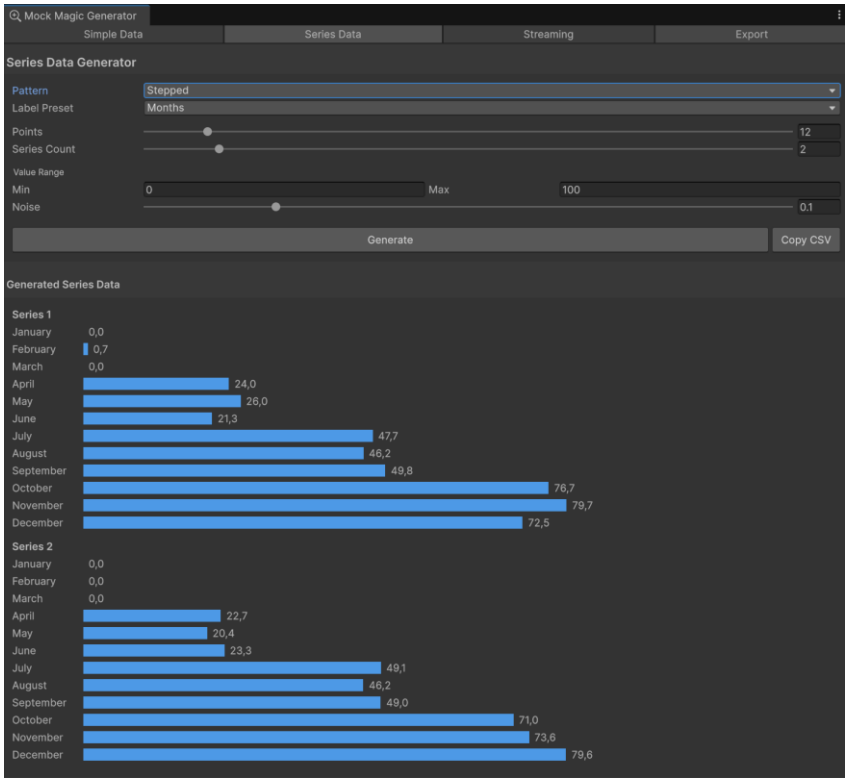
Preset	Labels	Example
Numeric	1, 2, 3, 4...	Simple numbered sequence
Months	January, February, March...	Full month names (wraps around)
MonthsShort	Jan, Feb, Mar...	Abbreviated months
Quarters	Q1, Q2, Q3, Q4	Quarterly labels (wraps around)
Years	2020, 2021, 2022...	Year sequence (configurable start)
Weekdays	Monday, Tuesday...	Full weekday names
WeekdaysShort	Mon, Tue, Wed...	Abbreviated weekdays
Products	Product A, Product B...	Generic product names
Regions	North, South, East, West...	Geographic regions
Categories	Category 1, Category 2...	Generic categories
Teams	Team Alpha, Team Beta...	Team names
Departments	Sales, Marketing, Engineering...	Business departments
Countries	USA, China, Japan...	Country names
Cities	New York, London, Tokyo...	City names
Colors	Red, Blue, Green...	Color names
Letters	A, B, C, D...	Alphabetic labels
Custom	*(User-defined)*	Provide your own label array

Multi-Series Data

Set **Series Count** to more than 1 to generate multiple series sharing the same labels. Each series follows the selected pattern independently but uses the same label axis. This is ideal for:

- **Grouped bar charts** - comparing categories across series
- **Multi-line charts** - overlaying trends
- **Stacked area charts** - showing composition over time

Series are automatically named "Series 1", "Series 2", etc. unless custom names are provided.



Understanding Noise

The **Noise** slider (0–0.5) adds controlled randomness to any pattern:

- **0** - Perfectly clean mathematical pattern. Useful for demonstrations or when you need exact shapes.
- **0.05–0.15** - Subtle variation. Data looks realistic without losing the pattern shape. This is the sweet spot for most chart testing.
- **0.3–0.5** - Heavy noise. The underlying pattern is still visible but data looks rough and organic.

All values are clamped to the min/max range after noise is applied, so noise never produces out-of-bounds values.

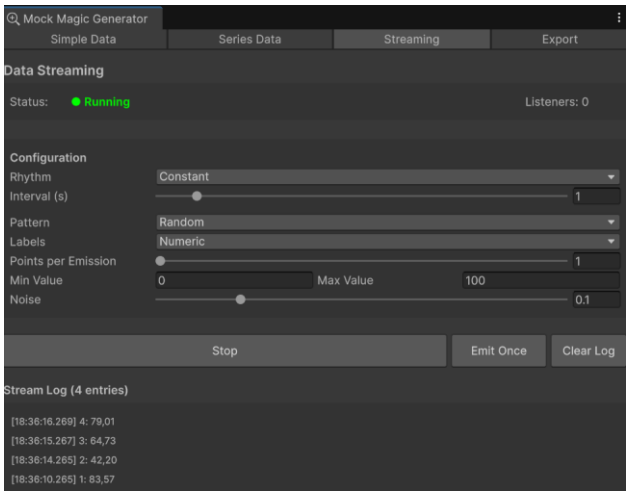
STREAMING DATA

The Streaming tab provides continuous real-time mock data emission. Instead of generating a static dataset, it emits new data points at configurable intervals - perfect for testing live dashboards, real-time charts, monitoring UIs, or any system that consumes a data feed.

Streaming works in both **Edit Mode** and **Play Mode**. You don't need to enter Play Mode to test a live data feed.

Using the Streaming Tab

1. **Status indicator** - Shows "● Running" (green) when streaming or "○ Stopped" (gray) when idle. Also displays the current listener count.
2. **Rhythm** - Select how data is emitted over time (see Rhythm Reference below). The controls below update dynamically based on the selected rhythm.
3. **Pattern** - The series pattern used for generated values (same 14 patterns as the Series tab).
4. **Labels** - The label preset for emitted data points.
5. **Points per Emission** - How many data points are generated per emission (1–10).
6. **Min / Max Value** - Value range for generated data.
7. **Noise** - Randomness factor (0–0.5).
8. **Start / Stop** - Toggle button to begin or end streaming.
9. **Emit Once** - Fire a single emission manually without starting continuous streaming.
10. **Clear Log** - Clear the stream log.
11. **Stream Log** - A scrollable list of timestamped entries showing each emitted data point. Newest entries appear at the top. Maximum 50 entries.



Rhythm Reference

Rhythm	Behavior	Controls
Constant	Emits at a fixed interval. Predictable, metronome-like.	**Interval** slider (0.1–10s)
Random	Emits at random intervals within a range. More organic feel.	**Min Interval** and **Max Interval** sliders
Burst	Emits a rapid burst of data, then pauses. Simulates batch updates.	**Burst Interval**, **Burst Count** (2–20), **Delay Between Bursts** (1–10s)
Drip	Emits single values one at a time. Slow, steady feed.	**Interval** slider (0.1–10s)

Edit Mode vs. Play Mode

	Edit Mode	Play Mode
How it works	Uses EditorApplication.update with timeSinceStartup tracking	Uses StartCoroutine with WaitForSeconds
When to use	Quick testing without entering Play Mode	Full runtime testing with MonoBehaviour lifecycle
Stream Log	Available	Available

Both modes produce identical output. Edit Mode streaming is particularly useful when prototyping - start a stream, observe the log, and stop it without ever pressing Play.

Label Continuity

Labels continue sequentially across emissions. If the first emission produces "Jan, Feb, Mar" and the next emission produces 3 more points, they will be labeled "Apr, May, Jun" - not "Jan, Feb, Mar" again. This makes streaming data behave realistically over time.

Listener Model

External systems receive stream data by registering as listeners. The **Status indicator** in the Streaming tab shows the current listener count. The MockMagic Editor window itself is always registered as a listener (that's how the Stream Log works). Other listeners -

such as charts, scripts, or monitoring systems - can register via the scripting API (see Section 8).

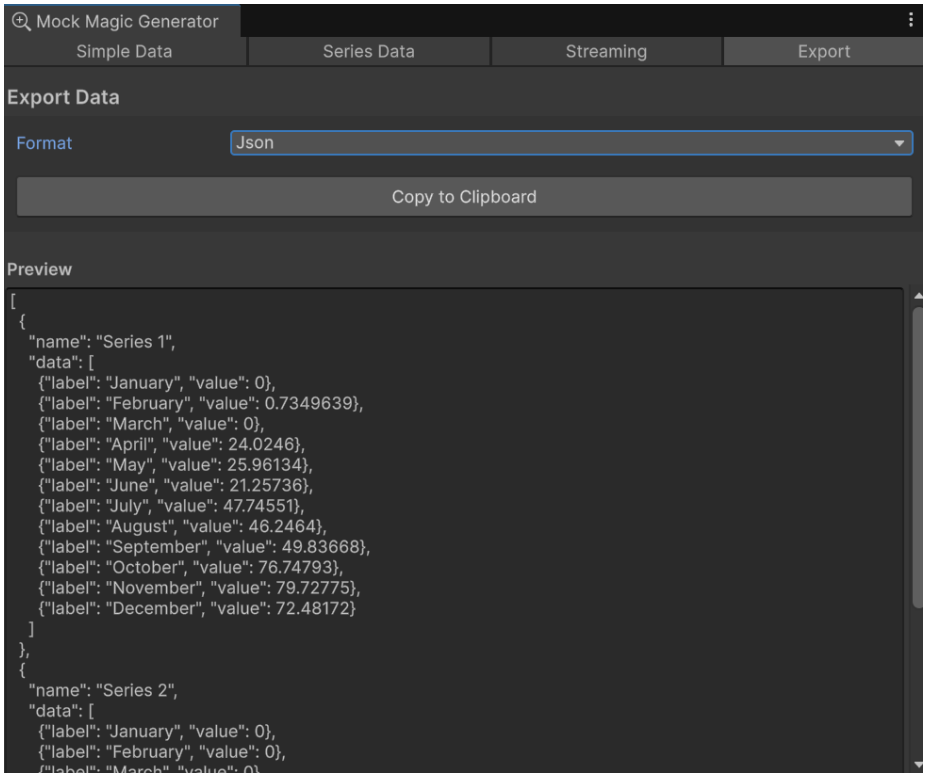
EXPORTING DATA

The Export tab formats your generated data for use outside MockMagic. Whether you need to paste test data into a spreadsheet, feed it to an API, or embed it directly in C# code, the Export tab handles the conversion and copies the result to your clipboard.

Using the Export Tab

1. **Format dropdown** - Select the output format: Text, CSV, JSON, or C# Code.
2. **Variable Name** - *(Shown only for C# Code format.)* Set the variable name used in the generated C# array declarations.
3. **Copy to Clipboard** - Click to copy the formatted output.
4. **Preview** - A scrollable text area showing the full formatted output. Updates automatically when you change the format or variable name.

If no data has been generated yet, the tab shows a help message: "Generate some data first."



Data Priority
The Export tab automatically uses the most recent data you generated, in this priority order:

1. **Multi-series data** (from the Series tab with Series Count > 1)
2. **Single series data** (from the Series tab with Series Count = 1)
3. **Simple data** (from the Simple Data tab)

FORMAT COMPARISON

Text
Human-readable, tab-separated output. Good for quick inspection and pasting into documents.
Simple data:
1. Sarah Johnson
2. Marcus Patel
3. Yuki Tanaka
Series data:
Label Value
———————————————————
Jan 65.00
Feb 89.00
Mar 45.00

CSV
Standard comma-separated values with headers. Import directly into Excel, Google Sheets, databases, or any CSV-compatible tool.
Simple data:
Index,Value
0,Sarah Johnson
1,Marcus Patel
2,Yuki Tanaka
Series data:
Label,Value,SecondaryValue,SeriesIndex,SeriesName
Jan,65.00,0.00,0,Revenue
Feb,89.00,0.00,0,Revenue
Mar,45.00,0.00,0,Revenue

JSON
Structured JSON output. Use for API testing, configuration files, or data exchange between systems.
Simple data:

["Sarah Johnson","Marcus Patel","Yuki Tanaka"]

Series data (single):

[

{"Label":"Jan","Value":65.0,"SecondaryValue":0.0,"SeriesIndex":0,"SeriesName":"Revenue"}
,

{"Label":"Feb","Value":89.0,"SecondaryValue":0.0,"SeriesIndex":0,"SeriesName":"Revenue"}
,

{"Label":"Mar","Value":45.0,"SecondaryValue":0.0,"SeriesIndex":0,"SeriesName":"Revenue"
}
]

Series data (multi-series):

[
  {
    "name":"Revenue",
    "data":[

{"Label":"Jan","Value":65.0,"SecondaryValue":0.0,"SeriesIndex":0,"SeriesName":"Revenue"}
    ]
  },
  {
    "name":"Costs",
    "data":[

{"Label":"Jan","Value":42.0,"SecondaryValue":0.0,"SeriesIndex":1,"SeriesName":"Costs"}
    ]
  }
]

C# Code

Ready-to-paste C# array declarations. Set the **Variable Name** field to control the identifier name.

Simple data (variable name: users):

```
string[] users = new string[]
{
    "Sarah Johnson",
    "Marcus Patel",
    "Yuki Tanaka"
```

};
Series data (variable name: revenue):
string[] revenueLabels = new string[] { "Jan", "Feb", "Mar" };
float[] revenueValues = new float[] { 65f, 89f, 45f };
Multi-series data generates shared labels and separate per-series value arrays:
string[] labels = new string[] { "Jan", "Feb", "Mar" };
float[] revenueValues = new float[] { 65f, 89f, 45f };
float[] costsValues = new float[] { 42f, 51f, 38f };
[Screenshot: Export tab showing C# Code format with variable name "revenue" and the generated code in the preview]

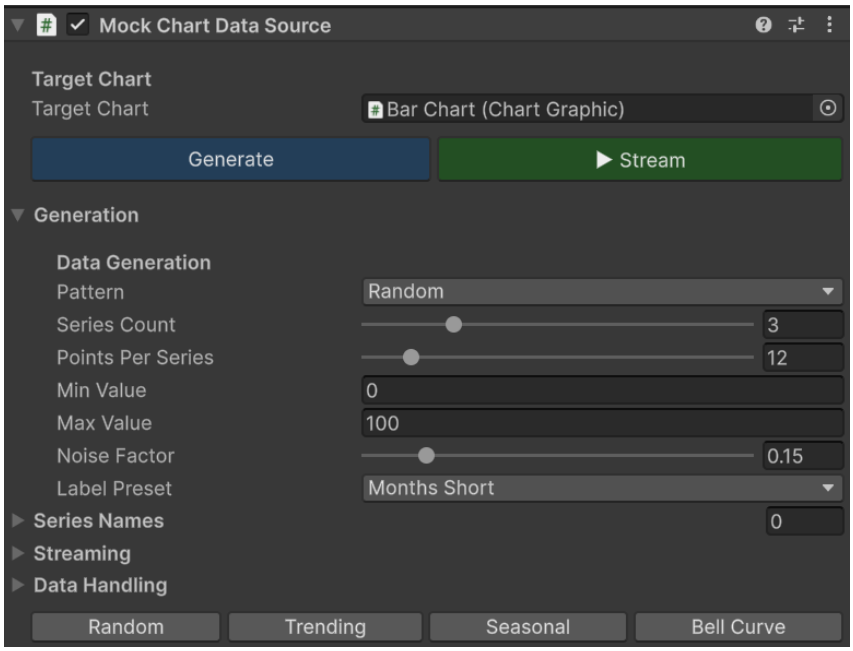
USING MOCKMAGIC WITH CHART GURU

Chart Guru ships with a built-in component called *MockChartDataSource* that is powered entirely by MockMagic. You do not need to open the MockMagic Generator window, export data, or write any code - just add the component to your chart and configure it in the Inspector.

Adding MockChartDataSource

There are two ways to add it:

1. **From the ChartGraphic Inspector** - When a ChartGraphic component has no data source attached, the Inspector shows an **"Add Mock Data Source"** button. Click it to add MockChartDataSource automatically.
2. **From the Add Component menu** - Select **Add Component → ChartGuru → Mock Chart Data Source**.



Inspector Layout

The *MockChartDataSource* Inspector has three foldout sections and action buttons:

Action Buttons

At the top of the component, two prominent buttons:

Button	Description
Generate	Generate a dataset once and apply it to the chart immediately
▶ Stream / ■ Stop	Start or stop continuous data streaming to the chart

A "Streaming active" info box appears while streaming is running.

Generation Foldout

Configure how one-shot data is generated:

Setting	Type	Default	Description
Pattern	SeriesPattern	TrendingUp	Mathematical pattern for values
Series Count	int	1	Number of data series (1–10)
Points Per Series	int	12	Data points per series (1–100)
Min Value	float	0	Minimum generated value
Max Value	float	100	Maximum generated value
Noise Factor	float	0.15	Randomness (0–1)
Label Preset	LabelPreset	MonthsShort	X-axis label source
Series Names	string[]	*(auto)*	Custom series names (shown when Series Count > 1)

Streaming Foldout

Configure continuous data emission:

Setting	Type	Default	Description
Generate On Enable	bool	true	Auto-generate data when the component is enabled
Auto Start	bool	false	Auto-start streaming when the component is enabled
Rhythm	StreamRhythm	Constant	Emission timing mode
Interval	float	1.0s	Base interval between emissions (0.05–30s)

Setting	Type	Default	Description
Min / Max Interval	float	0.5 / 2.0	*(Random rhythm only)* Interval range
Burst Count	int	5	*(Burst rhythm only)* Emissions per burst (1–20)
Burst Delay	float	3.0s	*(Burst rhythm only)* Pause between bursts (0.1–60s)
Points Per Emission	int	1	Data points per emission (1–20)

Data Handling Foldout

Configure how streamed data integrates with existing chart data:

Setting	Type	Default	Description
Mode	MockDataMode	Replace	How incoming data is handled
Max Points	int	50	Maximum data points to keep (shown for Append/Prepend modes)

The three data modes:

Mode	Behavior
Replace	Completely replaces the chart's data with each new emission
Append	Adds new points to the end; trims oldest points beyond Max Points
Prepend	Adds new points to the beginning; trims newest points beyond Max Points

Quick Pattern Buttons

At the bottom of the Inspector, four one-click buttons apply common configurations:

Button	Pattern	Notes
Random	Random	Quick random dataset
Trending	TrendingUp	Upward trend with noise
Seasonal	Seasonal	Also sets label preset to MonthsShort and points to 12
Bell Curve	BellCurve	Normal distribution shape

Each button immediately generates data and updates the chart.

Works in Edit Mode
MockChartDataSource is marked *[ExecuteAlways]*, meaning it works in both Edit Mode and Play Mode. You can:

- Click **Generate** to preview chart data without entering Play Mode
- Start **streaming** to see live chart updates in the Scene view
- Swap patterns and label presets with instant visual feedback

This makes it an excellent rapid prototyping tool - configure your chart's look and feel with realistic data before writing a single line of code.

Under the Hood
MockChartDataSource uses MockMagic's MockSeries class internally:

- **One-shot generation** calls MockSeries.Generate() (single series) or MockSeries.GenerateMultiSeries() (multi-series)
- **Streaming** calls MockSeries.GenerateLabels() and MockSeries.GenerateValues() to produce incremental data points
- Labels continue sequentially across emissions (same continuity behavior as MockMagic's Streaming tab)

Alternative Workflow: Export and Paste
If you prefer to use Chart Guru's ManualChartDataSource instead, you can:

1. Open the MockMagic Generator window (**Tools** → **Mock Magic** → **Generator**)
2. Generate series data in the **Series Data** tab
3. Switch to the **Export** tab and select **CSV** or **JSON**
4. **Copy to Clipboard**
5. Paste into your data source, configuration file, or script

This workflow gives you more control over the exact data, but requires an extra step compared to the integrated *MockChartDataSource* component.

SCRIPTING API

For developers who need mock data at runtime - automated testing, procedural content, live dashboards, or CI pipelines - MockMagic provides a clean static API. Every feature available in the Editor window is also available from code.

Assembly reference: Add a reference to the MockMagic assembly definition in your *.asmdef* file, or place your scripts in a folder without an assembly definition.
Namespace: using MockMagic;

Simple Data from Code

The Mock static class provides one method per data type, plus batch variants.

Single Values

```
using MockMagic;

string name = Mock.Name();           // "Sarah Johnson"
string email = Mock.Email();         // "marcus.patel@outlook.com"
string phone = Mock.Phone();         // "+1 (425) 738-2914"
int age = Mock.Age();                // 34
string address = Mock.FullAddress(); // "1847 Oak Ave, Seattle, WA 98101"
string guid = Mock.Guid();           // "a3f8b2c1-d4e5-4f6a-8b9c-0d1e2f3a4b5c"
float pct = Mock.Percentage();       // 87.45
bool flag = Mock.Boolean();          // true
string lorem = Mock.LoremIpsum(20);  // 20 words of lorem ipsum
UnityEngine.Color color = Mock.Color();// Random RGB color
```

Batch Values

Every single-value method has a batch counterpart that returns an array:

```
string[] names = Mock.Names(50);
string[] emails = Mock.Emails(100);
int[] ages = Mock.Ages(25);
string[] cities = Mock.Cities(10);
```

Generic Dispatch
Use Mock.Generate() when the data type is determined at runtime:

```
// Single value
string value = Mock.Generate(SimpleDataType.Email);

// Batch
string[] values = Mock.Generate(SimpleDataType.Email, 50);
```

Random Enum Values
Generate random values of any enum type:

```
MyEnum randomValue = Mock.Enum<MyEnum>();
MyEnum[] randomValues = Mock.Enums<MyEnum>(10);
```

ID Counter
Mock.Id*()* returns auto-incrementing integers. Reset with:

```
Mock.ResetIdCounter(0); // Next Id() call returns 0
```

Series Data from Code
The MockSeries static class generates patterned datasets.

Full-Featured Generation

```
using MockMagic;

MockDataPoint[] data = MockSeries.Generate(
    pattern: SeriesPattern.TrendingUp,
    count: 12,
    min: 0f,
    max: 100f,
    noise: 0.15f,
    labelPreset: LabelPreset.MonthsShort
);

// Each point has: data[i].Label, data[i].Value, data[i].SeriesName, etc.
```

Multi-Series Generation

```
MockDataPoint[][] multiData = MockSeries.GenerateMultiSeries(
    pattern: SeriesPattern.Seasonal,
    seriesCount: 3,
    pointCount: 12,
    min: 0f,
    max: 100f,
```

```
    noise: 0.1f,
    labelPreset: LabelPreset.MonthsShort,
    seriesNames: new[] { "Revenue", "Costs", "Profit" }
);

// multiData[0] = Revenue series, multiData[1] = Costs series, etc.
```

Convenience Methods

Short-form methods with sensible defaults:

```
MockDataPoint[] data = MockSeries.Linear();       // 12 points, 0–100, low noise
MockDataPoint[] data = MockSeries.Sine();         // Sine wave
MockDataPoint[] data = MockSeries.TrendingUp();   // Upward trend
MockDataPoint[] data = MockSeries.TrendingDown(); // Downward trend
MockDataPoint[] data = MockSeries.Seasonal();     // Seasonal pattern
MockDataPoint[] data = MockSeries.BellCurve();    // Normal distribution
MockDataPoint[] data = MockSeries.Random();       // Random values
MockDataPoint[] data = MockSeries.Stepped();      // Staircase
MockDataPoint[] data = MockSeries.Sparse();       // 70% zeros
MockDataPoint[] data = MockSeries.RandomWalk();   // Financial-style random walk
MockDataPoint[] data = MockSeries.Exponential();  // Exponential growth
```

Raw Values Only

If you only need the float array without labels:

```
float[] values = MockSeries.GenerateValues(
    SeriesPattern.Sine, count: 50, min: -1f, max: 1f, noise: 0f
);
```

Custom Labels

```
string[] labels = MockSeries.GenerateLabels(
    LabelPreset.MonthsShort, count: 12, startIndex: 0
);
// ["Jan", "Feb", "Mar", ..., "Dec"]
```

Streaming from Code

For continuous real-time data, use the MockStreaming static convenience class or the MockStream MonoBehaviour singleton directly.

Using MockStreaming (Recommended)

```
using MockMagic;
using UnityEngine;

public class LiveDashboard : MonoBehaviour
{
    private void OnEnable()
    {
        // Configure the stream
        MockStreaming.Configure(
            pattern: SeriesPattern.RandomWalk,
            min: 0f,
            max: 100f,
            noise: 0.1f,
            pointsPerEmission: 1,
            labelPreset: LabelPreset.MonthsShort
        );

        // Register a listener
        MockStreaming.Register(OnDataReceived);

        // Start streaming
        MockStreaming.Start(interval: 1.0f, rhythm: StreamRhythm.Constant);
    }

    private void OnDisable()
    {
        MockStreaming.Unregister(OnDataReceived);
        MockStreaming.Stop();
    }

    private void OnDataReceived(MockDataPoint[] points)
    {
        foreach (MockDataPoint point in points)
        {
            Debug.Log($"{point.Label}: {point.Value}");
        }
    }
}
```

Using MockStream Directly

MockStream is a singleton MonoBehaviour with full property access:

```
MockStream stream = MockStream.Instance;
stream.Pattern = SeriesPattern.Sine;
stream.Rhythm = StreamRhythm.Burst;
stream.Interval = 0.5f;
stream.BurstCount = 5;
stream.BurstDelay = 3f;
stream.MinValue = 0f;
stream.MaxValue = 100f;
stream.Noise = 0.1f;
stream.PointsPerEmission = 3;
stream.LabelPreset = LabelPreset.WeekdaysShort;

stream.OnDataEmitted += OnDataReceived;
stream.OnStreamStarted += () => Debug.Log("Stream started");
stream.OnStreamStopped += () => Debug.Log("Stream stopped");

stream.StartStream();
// ... later ...
stream.StopStream();
```

Single Emission

Fire one emission without starting continuous streaming:

```
MockStreaming.EmitOnce();
// or
MockStream.Instance.EmitOnce();
```

Export from Code

The MockExport static class converts data to formatted strings.

Exporting Series Data

```
using MockMagic;

MockDataPoint[] data = MockSeries.TrendingUp();

string text = MockExport.Export(data, ExportFormat.Text);
string csv  = MockExport.Export(data, ExportFormat.Csv);
string json = MockExport.Export(data, ExportFormat.Json);
```

```
string code = MockExport.Export(data, ExportFormat.CSharpCode, variableName:
"sales");
```

Exporting Multi-Series Data

```
MockDataPoint[][] multiData = MockSeries.GenerateMultiSeries(
    SeriesPattern.Seasonal, 3, 12, 0f, 100f, 0.1f, LabelPreset.MonthsShort
);

string json = MockExport.Export(multiData, ExportFormat.Json);
```

Exporting Simple Data

```
string[] emails = Mock.Emails(20);
string csv = MockExport.Export(emails, ExportFormat.Csv);
```

Copy to Clipboard

```
MockExport.CopyToClipboard(data);              // MockDataPoint[]
MockExport.CopyToClipboard(multiData);         // MockDataPoint[][]
MockExport.CopyToClipboard(emails);            // string[]
MockExport.CopyToClipboard("custom string");   // Raw string
```

Feeding MockMagic Data into Chart Guru

```
using MockMagic;
using ChartGuru;
using UnityEngine;

public class MockChart : MonoBehaviour
{
    private ChartGraphic _chart;

    private void Start()
    {
        // Generate mock data
        MockDataPoint[] data = MockSeries.Generate(
            SeriesPattern.TrendingUp, 12, 0f, 100f, 0.15f, LabelPreset.MonthsShort
        );

        // Build a Chart Guru chart
        ChartBuilder builder = Chart.Create();
        for (int i = 0; i < data.Length; i++)
        {
```

```
            BarMark bar = new BarMark(i, data[i].Value);
            bar.CategoryLabel = data[i].Label;
            builder.Add(bar);
        }

        Chart chart = builder
            .chartXAxis(x => x.Label("Month"))
            .chartYAxis(y => y.Label("Value"))
            .Build();

        _chart = ChartCanvasHelper.RenderToRectTransform(
            chart, GetComponent<RectTransform>(), true, "MockChart"
        );
    }
}
```

API REFERENCE SUMMARY

Mock (Static Class)
Single Value Methods

Method	Returns	Description
Guid()	string	New GUID
Id()	int	Auto-incrementing integer
ResetIdCounter(int startValue)	void	Reset ID counter
Timestamp()	long	Unix timestamp (±1 year from now)
Date(int yearsBack, int yearsForward)	string	"yyyy-MM-dd"
Time()	string	"HH:mm:ss"
DateTime(int yearsBack, int yearsForward)	string	Combined date and time
Name()	string	"FirstName LastName"
FirstName()	string	Random first name
LastName()	string	Random last name
Email()	string	Realistic email address
Phone()	string	US format phone number
Address()	string	Street address
FullAddress()	string	Full address with city, state, ZIP
City()	string	City name
State()	string	US state name
Country()	string	Country name
ZipCode()	string	5-digit ZIP code

Method	Returns	Description
Company()	string	Company name
JobTitle()	string	Job title
LoremIpsum(int wordCount)	string	Lorem ipsum text
Word()	string	Random word
Sentence(int minWords, int maxWords)	string	Random sentence
Paragraph(int sentenceCount)	string	Multiple sentences
Color()	UnityEngine.Color	Random RGB color
ColorName()	string	Named color (e.g., "Crimson")
HexColor()	string	"#RRGGBB" hex color
Coordinate()	Vector2	(latitude, longitude)
Latitude()	float	-90 to 90
Longitude()	float	-180 to 180
Currency(float min, float max)	decimal	Rounded to 2 decimal places
Percentage()	float	0–100 with 2 decimal precision
Boolean()	bool	50/50 true/false
Integer(int min, int max)	int	Random integer
Float(float min, float max)	float	Random float
Url()	string	Realistic HTTPS URL
Domain()	string	Domain name with TLD
IpAddress()	string	"x.x.x.x"
MacAddress()	string	"XX:XX:XX:XX:XX:XX"
Username()	string	Realistic username

Method	Returns	Description
Password(int length)	string	Strong password (shuffled, guaranteed character classes)
CreditCard()	string	Luhn-valid fake card number
Ssn()	string	"xxx-xx-xxxx" format
Isbn()	string	Valid ISBN-13
ProductName()	string	Product name with optional adjective
Price(float min, float max)	string	"$xx.xx" format
StarRating()	float	1.0–5.0 (skewed toward higher)
VersionNumber()	string	"major.minor.patch"
FileName()	string	Realistic filename with extension
UserAgent()	string	Browser user agent string
Age()	int	18–85 (bell-curve weighted)
Animal()	string	Animal name
Hashtag()	string	"#tag" format
Honorific()	string	"Mr.", "Dr.", etc.
Emoji()	string	Unicode emoji
Enum<T>()	T	Random value of any enum type

Batch Methods

Every single-value method above has a corresponding batch variant returning an array. The naming convention adds an s suffix:
Guids(count), Ids(count), Timestamps(count), Dates(count), Times(count), DateTimes(count), Names(count), FirstNames(count), LastNames(count), Emails(count), Phones(count), Addresses(count), Cities(count), Countries(count), ZipCodes(count), Companies(count), JobTitles(count), Words(count), Sentences(count), Paragraphs(count), Colors(count), ColorNames(count), HexColors(count), Coordinates(count), Latitudes(count), Longitudes(count), Percentages(count), Booleans(count), Integers(count), Floats(count), Urls(count), IpAddresses(count), MacAddresses(count), Usernames(count), Passwords(count), CreditCards(count), Ssns(count), Isbns(count), States(count), Domains(count), FullAddresses(count), Ages(count), ProductNames(count), Prices(count), StarRatings(count), VersionNumbers(count), FileNames(count), UserAgents(count), Animals(count), Hashtags(count), Honorifics(count), Emojis(count), Enums<T>(count)

Generic Dispatch

Method	Returns	Description
Generate(SimpleDataType type)	string	Generate one value of the specified type
Generate(SimpleDataType type, int count)	string[]	Generate multiple values of the specified type

---

MockSeries (Static Class)

Method	Returns	Description
Generate(SeriesPattern, int count, float min, float max, float noise, LabelPreset, int seriesIndex, string seriesName, string[] customLabels)	MockDataPoint[]	Generate a labeled series dataset
GenerateMultiSeries(SeriesPattern, int seriesCount, int pointCount, float min, float max, float noise, LabelPreset, string[] seriesNames, string[] customLabels)	MockDataPoint[][]	Generate multiple series sharing labels

Method	Returns	Description
GenerateValues(SeriesPattern, int count, float min, float max, float noise)	float[]	Generate raw pattern values (no labels)
GenerateLabels(LabelPreset, int count, int startIndex, int startYear, string[] customLabels)	string[]	Generate label strings from a preset

Convenience Methods

Method	Pattern	Default Points
Random()	Random	12
Linear()	Linear	12
Exponential()	Exponential	12
Sine()	Sine	12
TrendingUp()	TrendingUp	12
TrendingDown()	TrendingDown	12
Seasonal()	Seasonal	12
Stepped()	Stepped	12
Sparse()	Sparse	12
RandomWalk()	RandomWalk	12
BellCurve()	BellCurve	12

MockStream (MonoBehaviour Singleton)

Access via MockStream.Instance.

Properties

Property	Type	Description
Pattern	SeriesPattern	Pattern for generated values

Property	Type	Description
Rhythm	StreamRhythm	Emission timing mode
Interval	float	Base interval in seconds
MinInterval	float	Minimum interval (Random rhythm)
MaxInterval	float	Maximum interval (Random rhythm)
PointsPerEmission	int	Points generated per emission
BurstCount	int	Emissions per burst (Burst rhythm)
BurstDelay	float	Pause between bursts (Burst rhythm)
MinValue	float	Minimum generated value
MaxValue	float	Maximum generated value
Noise	float	Noise factor
LabelPreset	LabelPreset	Label source
IsRunning	bool	*(Read-only)* Whether streaming is active

Events

Event	Type	Description
OnDataEmitted	Action<MockDataPoint[]>	Fired each time data is emitted
OnStreamStarted	Action	Fired when streaming begins
OnStreamStopped	Action	Fired when streaming ends

Methods

Method	Returns	Description
StartStream()	void	Begin continuous streaming
StopStream()	void	Stop streaming
Toggle()	void	Toggle streaming on/off
EmitOnce()	void	Fire a single emission
Register(Action<MockDataPoint[]>)	void	Add a data listener
Unregister(Action<MockDataPoint[]>)	void	Remove a data listener
ClearListeners()	void	Remove all listeners
ListenerCount	int	*(Property)* Current listener count

MockStreaming (Static Convenience Class)
Wraps MockStream.Instance for ergonomic access without touching the singleton directly.

Method	Returns	Description
Configure(SeriesPattern, float min, float max, float noise, int pointsPerEmission, LabelPreset)	void	Configure all stream parameters
Start(float interval, StreamRhythm rhythm)	void	Start streaming
Stop()	void	Stop streaming
EmitOnce()	void	Fire a single emission
Register(Action<MockDataPoint[]>)	void	Add a data listener
Unregister(Action<MockDataPoint[]>)	void	Remove a data listener
IsRunning	bool	*(Property)* Whether streaming is active

MockExport (Static Class)
Series Data Export

Method	Parameters	Returns
Export(MockDataPoint[], ExportFormat, string variableName)	Single series + format	string
Export(MockDataPoint[][], ExportFormat, string variableName)	Multi-series + format	string
ToText(MockDataPoint[])	Single series	string
ToText(MockDataPoint[][])	Multi-series	string
ToCsv(MockDataPoint[], bool includeHeader)	Single series	string

Method	Parameters	Returns
ToCsv(MockDataPoint[][], bool includeHeader)	Multi-series	string
ToJson(MockDataPoint[], bool prettyPrint)	Single series	string
ToJson(MockDataPoint[][], bool prettyPrint)	Multi-series	string
ToCSharpCode(MockDataPoint[], string variableName)	Single series	string
ToCSharpCode(MockDataPoint[][], string variableName)	Multi-series	string

Simple Data Export

Method	Parameters	Returns
Export(string[], ExportFormat, string variableName)	String array + format	string
ToText(string[])	String array	string
ToCsv(string[], bool includeHeader)	String array	string
ToJson(string[])	String array	string
ToCSharpCode(string[], string variableName)	String array	string

Clipboard

Method	Parameters	Description
CopyToClipboard(MockDataPoint[])	Single series	Copy formatted data to system clipboard
CopyToClipboard(MockDataPoint[][])	Multi-series	Copy formatted data to system clipboard
CopyToClipboard(string[])	String array	Copy formatted data to system clipboard
CopyToClipboard(string)	Raw string	Copy raw string to system clipboard

---

Enums

SimpleDataType (51 values)

Guid, Id, Timestamp, Date, Time, DateTime, Name, FirstName, LastName, Email, Phone, Address, City, Country, ZipCode, Company, JobTitle, LoremIpsum, Sentence, Paragraph, Word, Color, HexColor, Coordinate, Latitude, Longitude, Currency, Percentage, Boolean, Integer, Float, Url, IpAddress, MacAddress, Username, Password, CreditCard, Ssn, Isbn, State

, Domain, FullAddress, Age, ProductName, Price, Rating, VersionNumber, FileName, User Agent, Animal, Hashtag, Honorific, Emoji

SeriesPattern (14 values)

Random, Linear, Exponential, Logarithmic, Sine, Cosine, TrendingUp, TrendingDown, Seasonal, Stepped, Sparse, RandomWalk, BellCurve, Sawtooth

LabelPreset (17 values)

Numeric, Months, MonthsShort, Quarters, Years, Weekdays, WeekdaysShort, Products, Regions, Categories, Teams, Departments, Countries, Cities, Colors, Letters, Custom

StreamRhythm (4 values)

Constant, Random, Burst, Drip

ExportFormat (4 values)

Text, Csv, Json, CSharpCode

MockDataMode (3 values - Chart Guru)

Replace, Append, Prepend

FAQ

I WOULD LIKE TO ADD AN ADDITIONAL TYPE

There is no generic concept yet for this but please contact me on Discord for such a request.

SUPPORT

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

Discord: https://discord.gg/uzeHzEMM4B