Trading4Pro Chart v2 Library API

Table of Contents

How to Use Chart Library v2	2
Demo Version Limitations	4
Chart Instance Options	5
Custom Color Schemes	8
Customizing Toolbar	11
Custom Menus	18
Chart API	19
General API	21
Individual Chart API	23
Technical Analysis	28
Assets and Schedules	41
Candle Data Management	43
Events API	46

How to Use Chart Library v2

Make sure you include the chart library on the web page:

```
<script src="chart.min.js"></script>
```

Create a chart instance:

```
<div id="chartDiv" style="width: 800px; height: 600px"></div>
<script type="text/javascript">
     let chart = new T4PChart('chartDiv', options);
     /* ... your code here ... */
</script>
```

It is also possible to use the ESM version of the library (if necessary). To do so, place the import statement inside a script element with the type attribute set to "module":

```
<script type="module">
     import T4PChart from './chart.esm.js';
     let chart = new T4PChart('chartDiv', options);
     /* ... your code here ... */
</script>
```

Also it is possible to pass the div element variable to the Chart constructor:

```
<script type="module">
     import T4PChart from ‘./chart.esm.js';
     let container = document.getElementById('chartDiv');
     let chart = new T4PChart(container, options);
     /* ... your code here ... */
</script>
```

After that, you can access the chart API via the *chart* variable:

```
chart.asset = 'FXCM:EURCAD'; // Or any other asset ID format
chart.type = 'candles';
chart.timeframe = 3600;
console.log(chart.symbol); // Expected output: EURUSD
```

You will find detailed chart API documentation in the Chart API section.

Demo Version Limitations

The demo version of the chart library provides limited functionality compared to the full version. The following restrictions apply:

Chart Type

Only *bars* and *line* chart types are available.

Timeframes

Only the following timeframes are supported:

- 1 Minute (60 seconds)
- 1 Day (86,400 seconds)

Indicators

Only the following indicators can be added:

- Exponential Moving Average (EMA)
- ZigZag

Drawings

Only the following drawing tools are available:

- Gann Fan
- Fibonacci Arcs

Chart Layout

Only two layout options are supported:

- 0 – one single chart
- 2 – three charts split horizontally

Trading Positions

Only stop loss can be set. Take profit is not available.

Chart Instance Options

As you can see from the previous example, you can provide the chart instance with an options object. Here is the list of supported options:

menu

Defines the main toolbar elements and allows custom menu elements to be added (see the Customizing Toolbar chapter).

Default value: ['asset', 'timeframe', 'type', 'indicators', 'drawings', 'layout', 'clean', 'language', 'theme']

assets

Defines various settings for each asset (see Assets and Schedules).

Default value: Map(0)

timeframes

Sets available timeframes (in seconds).

Default value: {
'1 Tick': 1,
'1 Minute': 60,
'5 Minutes': 300,
'15 Minutes': 900,
'30 Minutes': 1800,
'1 Hour': 3600,
'4 Hours': 14400,
'1 Day': 86400,
'1 Week': 604800,
}

themes

Adds custom color schemes (see Custom Color Schemes).

Default value: {}

theme

Specifies the default color scheme.

The default value is set to the browser's preferred color scheme setting. Built-in themes: light, dark.

languages

Sets the list of allowed languages.

Default value: ['en', 'de', 'es', 'fr', 'pt', 'it', 'ru', 'tr', 'ar', 'ms', 'id', 'th', 'vi', 'zh', 'zh-Hant', 'ja', 'ko']
Supported languages:

- en - English;
- de - German;
- es - Spanish;
- fr - French;
- pt - Portuguese;
- it - Italian;
- ru - Russian;
- tr - Turkish;
- ar - Arabic;
- ms - Malay;
- id - Indonesian;
- th - Thai;
- vi - Vietnamese;
- zh - Simplified Chinese;
- zh-Hant - Traditional Chinese;
- ja - Japanese;
- ko - Korean.

language

Defines the default application language.

The default value is set to the browser's preferred language setting.

timezone

Defines the default application timezone.

Default value: 'UTC'

layout

Sets the default application layout scheme (from 0 to 9).

Default value: 0

charts

Sets the default settings for charts (up to 8 charts for layout = 8). Each chart contains four parameters:

asset – chart asset identifier. Default value: 'EURUSD';
timeframe – chart timeframe value (in seconds). *Default value: 3600;*
type – chart type. Supported values: 'candles', 'bars', 'line', 'area'. Default value: 'candles';
scale – chart scale (from 0.1 to 2). Default value: 1.

Default value: [] – which means every chart will use default parameters.

Note: the latest specified set of properties will be considered the default.

volume

Specifies if the volume chart is shown (requires volume data).

Default value: true

restore

Defines which data should be restored after the chart is reloaded.

Default value: ['layout', 'theme', 'language', 'timezone']
Supported items:

- *layout* - restores the chart layout configuration, including assets, timeframes, charts' types and scales;
- *theme* - restores the last used color scheme;
- *language* - restores the last selected language;
- *timezone* - restores the last selected timezone;
- *indicators* - restores all indicators on the chart;
- *drawings* - restores all drawings on the chart;

priceMenu

Defines the price axis menu (see the Custom Menus chapter).

Default value: null (menu is disabled)

contextMenu

Defines the chart's context menu (see Custom Menus chapter).

Default value: ['paste']

Custom Color Schemes

It is possible to define a custom theme by specifying it in the *themes* option of the chart during initialization. Here is an example showing the implementation of a custom theme.

```
let chart = new T4PChart('chartContainer', { themes: {
     custom: {
          menuBg: '#3b3d45',
          menuBorder: '#53585f',
          menuElemBg: '#3b3d45',
          menuElemIcon: '#babec7',
          menuElemText: '#babec7',
          menuElemHoverBg: '#3b3d45',
          menuElemHoverIcon: '#00b0ea',
          menuElemHoverText: '#00b0ea',
          menuElemActiveBg: '#00b0ea',
          menuElemActiveIcon: '#ffffff',
          menuElemActiveText: '#ffffff',
          submenuDelimiter: '#53585f',
          submenuElemBg: '#3b3d45',
          submenuElemIcon: '#babec7',
          submenuElemText: '#babec7',
          submenuElemActiveBg: '#3b3d45',
          submenuElemActiveIcon: '#00b0ea',
          submenuElemActiveText: '#00b0ea',
          contextToolbarIcon: '#babec7',
          contextToolbarHoverIcon: '#eceff6',
          contextToolbarActiveIcon: '#00b0ea',
          popupBg: '#3b3d44',
          popupMobileBg: '#3b3d44',
          popupBorder: '#53585f',
          popupShadow: '0px 4px 12px 0px #25262b',
          popupText: '#babec7',
```

```
            popupHeaderBg: '#1e1f22',
            popupHeaderText: '#babec7',
            popupHeaderClose: '#babec7',
            formInputBg: '#3b3d44',
            formInputBorder: '#5b606b',
            formInputHoverBorder: '#757c8a',
            formInputText: '#babec7',
            formInputPlaceholder: '#5b606b',
            formInputActiveBg: '#00b0ea',
            formInputActiveText: '#ffffff',
            formInputHighlight: '#d5dbe6',
            formSeparator: '#babec7',
            chartBg: '#25262b',
            chartFrame: '#00b0ea',
            candleRising: '#7ac522',
            candleFalling: '#e64b3b',
            candleShadow: '#babec7',
            rateLineRising: 'rgba(122, 197, 34, 0.5)',
            rateLineFalling: 'rgba(230, 75, 59, 0.5)',
            line: '#1cb5ee',
            areaLine: '#1cb5ee',
            areaBg: 'rgba(28, 181, 238, 0.1)',
            axisLine: '#53585f',
            axisText: '#babec7',
            grid: '#555555',
            dataBg: '#25262b80',
            dataBorder: '#53585f',
            dataHoverBorder: '#f3f5f8',
            dataText: '#babec7',
        }
} });
chart.theme = 'custom';
```

Note that all object keys are optional: unspecified colors will be taken from the *light* built-in theme.

Customizing Toolbar

It is possible to both specify default toolbar elements and define custom ones.

The set of toolbar elements is defined by the *menu* option of the options object passed to the chart constructor (see Chart Instance Options).

Here's the list of default toolbar elements:

- *asset* - a symbol search box to change the chart's asset;
- *timeframe* - a dropdown list that allows you to choose the active chart's timeframe;
- *type* - a dropdown list that allows you to select the active chart type (candles, bars, line, or area);
- *indicators* - a list of indicators to add;
- *drawings* - a list of drawings to add;
- *layout* - a dropdown list for choosing the multi-chart layout;
- *clean* - a menu that allows you to clear technical analysis on the chart;
- *language* - the application language selector;
- *theme* - a toggle button for switching between light and dark color schemes.

In order to define a custom menu element, it is necessary to pass the descriptor object as an element of the *menu* option array:

```
new Chart(chartDiv, {
menu: ['asset', 'timeframe', 'type', /* etc. */
     { // Custom button declaration
          type: 'button',
          icon: 'myIcon',
          title: 'Click Me',
          handler: () => console.log('Button clicked!')
     },
     { // Custom dropdown list declaration
          type: 'select',
          value: 'one', // The default value
          items: [
               { icon: 'iconOpt1', title: 'Option #1', value:
               'one' },
```

```
                { icon: 'iconOpt2', title: 'Option #2', value:
                'two' },
                { icon: 'iconOpt3', title: 'Option #3', value:
                'three' }
            ],
            handler: (value) => console.log('Option selected: ',
            value)
        },
]});
```

In the example above two custom elements are declared: a button and a dropdown list with three options to choose from.

Important: The *icon* parameter is the name of an SVG symbol. Make sure the corresponding SVG code is present inside the chart container DIV element. Custom elements use the *icon* value within the <use> tag:

```
<svg><use href="myIcon"></use></svg>
```

There are five types of custom elements:

- *button* - a simple button;
- *search* - a dynamic list of values which displays the list of results depending on the text entered;
- *select* - a dropdown list of fixed values to choose from;
- *submenu* - a dropdown list of items with separate handler functions for each item;
- *toggle* - a button which switches between a specified number of values.

Each element type has its own set of properties.

Button

compact
: Forces the element to be compact (title is not displayed).
Default value: false

icon
Sets the element icon.

title
Sets the element title (not displayed when *compact* is true).

handler - Function()
Specifies the click handler function.

Search

compact
Forces the element to be compact (title is not displayed).
Default value: false

icon
Sets the element icon.

title
Sets the element title (not displayed when *compact* is true).

value
Sets the default element value.

search
Sets the search handler function that must return an array of [value, title] rows. For example: [['BTCUSD', 'Bitcoin'], [‘ETHUSD’, ‘Ethereum’]]. Can be asynchronous.

handler - Function(value)
Specifies the value change handler function.

Select

compact
Forces the element to be compact (title is not displayed).
Default value: false

icon
Sets the element icon.

title
Sets the element title (not displayed when *compact* is true).

value
Sets the default element value (must be one of the values specified in *items* property).

items
Defines available options. Each options is described with an object: { icon: ‘optional_icon', title: ‘text', value: 'unique_value' }.

handler - Function(value)
Specifies the value change handler function.

Submenu

compact
Forces the element to be compact (title is not displayed).
Default value: false

icon
Sets the element icon.

title
Sets the element title (not displayed when *compact* is true).

items
Defines available submenu items. Each item is described with an object: { icon: ‘optional_icon', title: ‘text', handler: someFunction}.

Toggle

compact
Forces the element to be compact (title is not displayed).
Default value: false

value
Sets the default element value (must me one of the values specified in *options* property).

options
: Defines available options. Each options is described with an object: { icon: ‘optional_icon', title: ‘text', value: 'unique_value' }.

handler - Function(value)
: Specifies the value change handler function.

Here is an example of a custom element representing a submenu with four basic math operations. Choosing any operation opens a custom form where the user can enter A and B operand values. Once the form is submitted, an alert windows appears showing the result of the corresponding operation.

```
let mathElement = {
     type: 'submenu',
     icon: 'math',
     title: 'Math',
     items: [
          { icon: 'add', title: 'Add', handler: () => {
               chart.setForm(mathForm, {}, (form) => {
                    alert(form.a + form.b);
                    chart.setForm(null);
               });
          }},
          { icon: 'sub', title: 'Subtract', handler: () => {
               chart.setForm(mathForm, {}, (form) => {
                    alert(form.a - form.b);
                    chart.setForm(null);
               });
          }},
          { icon: 'mul', title: 'Multiply', handler: () => {
               chart.setForm(mathForm, {}, (form) => {
                    alert(form.a * form.b);
                    chart.setForm(null);
               });
          }},
```

```
          { icon: 'div', title: 'Divide', handler: () => {
               chart.setForm(mathForm, {}, (form) => {
                    alert(form.a / form.b);
                    chart.setForm(null);
               });
          }},
     ]
};
```

Where *mathForm* is a custom form descriptor:

```
let mathForm = {
     title: 'Enter A and B values',
     icon: 'mathIcon',
     areas: {
          txtA: 'Operand A',
          valA: { component: 'NumberInput', name: 'a', value:
          1, step: 0.1 },
          txtB: 'Operand B',
          valB: { component: 'NumberInput', name: 'b', value:
          1, step: 0.1 },
          footer: [{ component: 'ButtonSubmit' }, { component:
          'ButtonCancel' }],
     },
     form: {
          desktop: {
               columns: '1fr 1fr 1fr 1fr',
               template: "'txtA valA txtB valB' 'footer footer
               footer footer'"
          },
          portrait: {
               columns: '1fr 1fr',
               template: "'txtA valA' 'txtB valB' 'footer
               footer'"
```

```
          },
          landscape: {
               columns: '1fr 1fr 1fr 1fr',
               template: "'txtA valA txtB valB' 'footer footer
               footer footer'"
          }
     }
}
```

Custom Menus

There are two types of custom menus that the chart supports: the price axis menu and the context menu.

priceMenu
: Adds a special button on the price axis, allowing you to open a popup menu at a specific price level.

contextMenu
: A popup menu that appears when you right-click on any area of the chart. By default, it consists of a single element 'paste', designed to insert chart indicators and drawings previously copied to the clipboard.

Both menus are customizable and are defined by the *priceMenu* and *contextMenu* parameters of the options object passed to the chart constructor (see Chart Instance Options).

To define a custom menu element, you need to pass the descriptor object as an element of the menu *options* array:

```
new Chart(chartDiv, {
contextMenu: ['paste', // Default 'paste' menu option
(applicable only for contextMenu, not for priceMenu)
     { // Custom context menu option
          icon: 'myIcon', // Optional
          title: 'Open Google',
          handler: () => window.open('https://google.com')
     }
]});
```

Important: The *icon* parameter is the name of an SVG symbol. Make sure the corresponding SVG code is present inside the chart container DIV element. Custom elements use the *icon* value within the <use> tag:

```
<svg><use href="myIcon"></use></svg>
```

Chart API

The Chart API consists of two main parts: the general API and the individual chart API. The general API provides access to the application-level methods and properties, such as setting the language, changing the color scheme, etc. The individual chart API provides full control over separate charts (the application layout can have multiple charts).

Here is example code demonstrating the usage of the general and individual chart-level APIs:

```
chart.theme = 'dark'; // General API: set current color scheme
to 'dark'


chart.language = 'zh'; // General API: set application
language to Simplified Chinese


console.log(chart.active); // General API: display the index
of the currently active (selected) chart


chart[0].asset = 'FXCM:EURUSD'; // Chart level API: set the
first chart's asset to EURUSD (asset ID = 'FXCM:EURUSD')


chart[1].indicators.add('Fractals'); // Chart level API: add
Fractals indicator (with default parameters) to the second
chart


console.log(chart.timeframe); // Chart level API (!): display
active (selected) chart's timeframe
```

Pay attention to the last line in the example. Although it does not address a specific chart, by default it addresses the active chart. This line of code is equivalent to:

```
console.log(chart[chart.active].timeframe);
```

Important: When addressing a chart by index, make sure it exists. Use *chart.length* to determine how many charts the currently selected layout has:

```
// Iterate through all charts and display their symbols
for (let i = 0; i < chart.length; i++) {
     console.log(chart[i].symbol);
}
```

General API

chart.active - read/write
Sets or gets the currently active (selected) chart index.

Default value: 0

chart.theme - read/write
Sets or gets the application color scheme.

Default value: depends on the browser's preferred color scheme setting.
Built-in themes: light, dark.

chart.language - read/write
Sets or gets the application language.

Default value: depends on the browser's preferred language setting.

chart.timezone - read/write
Sets or gets the application timezone used to display date/time values on charts. Keep in mind that the *timezone* value must be a valid timezone IANA/Olson TZ string such as: 'Australia/Sydney', 'America/Los_Angeles', 'Europe/Rome' etc.

Default value: 'UTC'

chart.layout - read/write
Sets or gets the layout scheme for multiple charts.

Default value: 0
Supported values:

- *0* - one single chart;
- *1* - two charts split horizontally;
- *2* - three charts split horizontally;
- *3* - two charts split vertically;
- *4* - three charts split vertically;
- *5* - four charts split vertically;
- *6* - four charts, table 2x2;
- *7* - six charts, table 3x2.
- *8* - eight charts, table 4x2.
- *9* - six charts, one primary and five small charts;

chart.events - read only

Provides access to the application events API (see Events API). Allows handling of various chart events such as candle requests, layout changes, indicator/drawing modifications etc.

Available methods:

- *events.add(event, handler)* - adds an event listener for a specific chart event;
- *events.remove(event, handler)* - removes a specific event listener.

The *events* object also contains all supported event types (see Events API).

chart.assets - read only

Provides access to assets API (see Assets and Schedules).

Available methods:

- *assets.set(id, data)* - specifies asset properties (see Assets and Schedules).
  id - asset identifier (usually the symbol is used);
  data - object containing asset properties:
  - *symbol* - asset symbol;
  - *name* - asset display name;
  - *decimals* - number of digits after the decimal point (price precision);
  - *schedule* - defines trading hours.
- *assets.get(id) - get asset properties.*
- *assets.find([where])* - retrieves assets that satisfy the condition implemented by the *where* function. If no *where* is provided, it returns a list of all assets available;
- *assets.length* - returns the total amount of assets currently available.

chart.rates(assetId, timestamp, rates, [volume])

Updates the corresponding candle or creates a new one based on the provided *rates* for the specified asset (see Candle Data Management).

Important: This process depends on the asset's schedule and can result in duplicate candles or other issues if the schedule is set incorrectly.

Individual Chart API

chart.asset OR
chart[index].asset - read/write

Sets or gets the active chart asset ID OR the asset ID of the chart at the specified index.

Default value: 'EURUSD'

chart.symbol OR
chart[index].symbol - read only

Returns the active chart asset symbol OR the asset symbol of the chart at the specified index.

Identical to: *chart.assets.get(chart.asset).symbol*

chart.name OR
chart[index].name - read only

Returns the active chart asset name OR the asset name of the chart at the specified index.

Identical to: *chart.assets.get(chart.asset).name*

chart.decimals OR
chart[index].decimals - read only

Returns the active chart asset decimals value OR the asset decimals of the chart at the specified index.

Identical to: *chart.assets.get(chart.asset).decimals*

chart.schedule OR
chart[index].schedule - read only

Returns the active chart asset schedule OR the asset schedule of the chart at the specified index.

Identical to: *chart.assets.get(chart.asset).schedule*

chart.timeframe OR
chart[index].timeframe - read/write

Sets or gets the timeframe of the active chart OR the chart at the specified index.

Default value: 3600
Available timeframes by default: {
'1 Tick': 1,
'1 Minute': 60,
'5 Minutes': 300,
'15 Minutes': 900,
'30 Minutes': 1800,
'1 Hour': 3600,
'4 Hours': 14400,
'1 Day': 86400,
'1 Week': 604800,
}

chart.type OR
chart[index].type - read/write

Sets or gets the active chart type OR the type of the chart at the specified index.

Default value: 'candles'
Supported values: 'candles', 'bars', 'line', 'area'.

chart.position OR
chart[index].position - read/write

Sets or gets the active chart's position on the timeline OR the position of the chart at the specified index. The position is defined as the timestamp of the point located 10% from the chart's right edge on the time axis.

chart.scale OR
chart[index].scale - read/write

Sets or gets the active chart horizontal scale OR the scale of the chart at the specified index.

Default value: 1
Supported values: from 0.1 to 2.

chart.range OR
chart[index].range - read/write

Sets or gets the active chart's timeline range OR the timeline range of the chart at the specified index. The range is defined as an array of two timestamps, representing the start and end of the time axis.

chart.empty OR
chart[index].empty - read only

Returns *true* if the active chart OR the chart at the specified index does not contain any candles data. Otherwise returns *false.*

chart.candles OR
chart[index].candles - read only

Provides access to the active chart's candles API OR candles API of the chart at the specified index (see Candle Data Management). Allows setting or requesting candles data for the chart's symbol and timeframe.

Available methods:

- *candles.set(candles)* - sets or updates the candle data for the active chart with the provided array of candles. In some cases, can be used to preload candle data. Though normally, adding candles is implemented via handling the events.CANDLES_REQUEST event (see Candle Data Management);
- *candles.get(fromTimestamp, toTimestamp)* - retrieves candle data within the specified time range, from *fromTimestamp* to *toTimestamp;*
- *candles.get(timestamp, nBefore, nAfter)* - gets candle data around a specified timestamp, with *nBefore* candles before and *nAfter* candles after the timestamp;
- *candles.get()* - retrieves all candle data currently displayed on the chart.

chart.indicators OR
chart[index].indicators - read only

Provides access to the active chart's indicators API OR indicators API of the chart at the specified index (see Technical Analysis). Allows creating, modifying and deleting various technical indicators.

Available methods:

- *indicators.add(name, [params])* - adds a new indicator to the chart with the specified name. Optional *params* can be provided to configure the indicator settings (see details in Technical Analysis);
- *indicators.find([where])* - retrieves chart indicators that satisfy the condition implemented by the *where* function. If no *where* is provided, it returns a list of all indicators currently on the chart;
- *indicators.length* - returns the total amount of indicators currently applied to the chart.

chart.drawings OR
chart[index].drawings - read only

Provides access to the active chart's drawings API OR drawings API of the chart at the specified index (see Technical Analysis). Allows creating, modifying and deleting various technical analysis figures.

Available methods:

- *drawings.add(name, [params])* - adds a new drawing to the chart with the specified name. Optional *params* can be provided to configure the drawing styles (see details in Technical Analysis);
- *drawings.find([where])* - retrieves chart drawings that satisfy the condition implemented by the *where* function. If no *where* is provided, it returns a list of all drawings currently on the chart;
- *drawings.length* - returns the total amount of drawings currently added to the chart.

chart.positions OR
chart[index].positions - read only

Provides access to the active chart's trading positions API OR positions API of the chart at the specified index. Allows creating, modifying and deleting trading positions.

Available methods:

- *positions.add(params, [assetId])* - adds a new trading position to the chart for the current or specified asset;
- *positions.get([where])* - retrieves chart trading positions that satisfy the condition implemented by the *where* function. If no *where* is provided, it returns a list of all positions added to the chart;
- *positions.length* - returns the total amount of positions currently added to the chart.

Trading position parameters:

- *side* - Position side. Supported values: 'buy', 'sell', default value: 'buy'.
- *entry* - Entry price, default value: last candle close.
- *opened* - Entry timestamp (milliseconds since epoch), default value: last candle timestamp.
- *closed* - Close timestamp (milliseconds since epoch) or null for open positions, default value: null.
- *takeProfit* - Take-profit price. When specified, the TP line, profit tag are displayed, default value: null.
- *stopLoss* - Stop-loss price. When specified, the SL line, loss tag are displayed, default value: null.
- *title* - Text label shown near the entry, default value: ''.

- *result* - Position summary result text shown at the right tag, default value: ''.
- *profit* - Text displayed at the profit tag (*takeProfit* level), default value: ''.
- *loss* - Text displayed at the loss tag (*stopLoss* level), default value: ''.
- *colorEntry* - Entry line color, default value: '#00b0ea'.
- *colorProfit* - Profit line color, default value: '#2eb23b'.
- *colorLoss* - Loss line color, default value: '#e54c54'.

chart.clear([filter]) OR
chart[index].clear(filter)

Deletes all technical analysis tools previously added to the active chart OR to the chart at the specified index.

filter - optional parameter, supported values: ‘indicators’, ‘drawings’

Technical Analysis

Indicators

The indicators API is accessible via the *indicators* property of the chart (see more details in the Individual Chart API chapter).

When adding indicators, omitted parameters will be set to default values:

```
// Add MACD with default parameters to the first chart
chart[0].indicators.add('MovingAverageConvergenceDivergence');

// Add RSI with specified parameters to the active chart
chart.indicators.add('RelativeStrengthIndex', {
     n: 20,
     overbought: 80,
     oversold: 20,
     color: '#887720',
     bold: true,
     colorBg: '8877201f'
});

// Add Envelope indicator using chart.indicators type
chart.indicators.add(chart.indicators.Envelope);
```

Important: All color parameters must be valid CSS color string values!

In order to access a particular indicator's API, first in is necessary to obtain the indicator object:

```
// Get the first indicator of the last chart and display its
name
let indicator = chart[chart.length - 1].indicators.find()[0];
if (indicator != null) console.log(indicator.name);
```

```
// Get all indicators of the active chart and display their
parameters
let indicators = chart.indicators.find();
for (let indicator of indicators) {
     console.log(indicator.params);
}

// Delete the last added indicator on the active chart
chart.indicators.find()[chart.indicators.lenght - 1]?.drop();
```

API:

- *indicator.name* - full indicator name;
- *indicator.params* - contains all indicator settings and styles*;*
- *indicator.drop()* - deletes the indicator;

You can change values of the *indicator.params* object in order to change the indicator parameters.

Here is the list of all available indicators at the moment:

Aroon
name: 'Aroon'
settings:
- *n* - Length, default value: 15

styles:
- *colorUp* - Up line color, default value: '#ef7d00'
- *boldUp* - Up line bold, default value: false
- *colorDown* - Down line color, default value: '#006ab4'
- *boldDown* - Down line bold, default value: false

Average Directional Index
name: 'AverageDirectionalIndex'
settings:
- *n* - Length, default value: 14

styles:
- *colorADX* - ADX line color, default value: '#003f84'
- *boldADX* - ADX line bold, default value: false

- *colorPDI* - Plus DI line color, default value: '#f78d59'
- *boldPDI* - Plus DI line bold, default value: false
- *colorMDI* - Minus DI line color, default value: '#5bc488'
- *boldMDI* - Minus DI line bold, default value: false

Bollinger Bands

name: 'BollingerBands'

settings:

- *n* - Length, default value: 15
- *k* - StdDev, default value: 1

styles:

- *colorBasis* - Basis line color, default value: '#0e73b9'
- *boldBasis* - Basis line bold, default value: false
- *colorBorder* - Borders color, default value: '#00b0ea'
- *boldBorder* - Borders bold, default value: false
- *colorBg* - Background color, default value: 'rgba(0,176,235,0.2)'

Bulls Bears Power

name: 'BullsBearsPower'

settings:

- *n* - Length, default value: 15
- *showBull* - Display Bulls line, default value: true
- *showBear* - Display Bears line, default value: true
- *showCombined* - Display Combined line, default value: true

styles:

- *colorBull* - Bulls line color, default value: '#8bc344'
- *boldBull* - Bulls line bold, default value: false
- *colorBear* - Bears line color, default value: '#d55744'
- *boldBear* - Bears line bold, default value: false
- *colorCombined* - Combined line color, default value: '#550d8e'

Donchian Channel

name: 'DonchianChannel'

settings:

- *n* - Length, default value: 15

styles:

- *colorBasis* - Basis line color, default value: '#ee7d00'
- *boldBasis* - Basis line bold, default value: false
- *colorBorder* - Borders color, default value: '#ffcb00'
- *boldBorder* - Borders bold, default value: false
- *colorBg* - Background color, default value: 'rgba(255,202,0,0.2)'

Envelope

name: 'Envelope'

settings:

- *n* - Length, default value: 15
- *p* - Percentage, default value: 5

styles:

- *colorMiddle* - Middle line color, default value: '#ce2867'
- *boldMiddle* - Middle line bold, default value: false
- *colorBorder* - Borders, default value: '#f194b7'
- *boldBorder* - Borders bold, default value: false
- *colorBg* - Background color, default value: 'rgba(206,40,103,0.15)'

Exponential Moving Average

name: 'ExponentialMovingAverage'

settings:

- *n* - Length, default value: 15

styles:

- *color* - Line color, default value: '#eb609e'
- *bold* - Line bold, default value: false

Fractals

name: 'Fractals'

settings:

- *n* - Periods, default value: 2

styles:

- *colorMax* - Maxima arrows color, default value: '#8bc344'
- *fillMax* - fill maxima arrows instead of contour, default value: false
- *colorMin* - Minima arrows, default value: '#d55744'
- *fillMin* - fill minima arrows instead of contour, default value: false

Ichimoku Cloud

name: 'IchimokuCloud'

settings:

- *nConv* - Conversion, default value: 9
- *nBase* - Base, default value: 26
- *nSpan* - Lagging Span, default value: 52
- *nDisp* - Displacement, default value: 26

styles:

- *colorConv* - Conversion line color, default value: '#e20612'
- *boldConv* - Conversion line bold, default value: false
- *colorBase* - Base line color, default value: '#65b22e'
- *boldBase* - Base line bold, default value: false
- *colorLag* - Lagging line color, default value: '#006ab4'

- *boldLag* - Lagging line bold, default value: false
- *colorA* - Lead 1 color, default value: '#934b95'
- *boldA* - Lead 1 bold, default value: false
- *colorB* - Lead 2 color, default value: '#ef7d00'
- *boldB* - Lead 2 bold, default value: false
- *colorBgAB* - Background 1 color, default value: 'rgba(101,178,46,0.3)'
- *colorBgBA* - Background 2 color, default value: 'rgba(226,6,18,0.2)'

Momentum

name: 'Momentum'

settings:

- *n* - Length, default value: 15

styles:

- *color* - Line color, default value: '#eb609e'
- *bold* - Line bold, default value: false

Moving Average Convergence Divergence

name: 'MovingAverageConvergenceDivergence'

settings:

- *nFast* - Fast, default value: 12
- *nSlow* - Slow, default value: 26
- *nSignal* - Signal, default value: 9

styles:

- *colorHistogram* - Histogram color, default value: '#944b96'
- *colorMACD* - MACD line color, default value: '#65b22e'
- *boldMACD* - MACD line bold, default value: false
- *colorSignal* - Signal line color, default value: '#40c4ef'
- *boldSignal* - Signal line bold, default value: false

Parabolic SAR

name: 'ParabolicSAR'

settings:

- *min* - Acceleration factor start, default value: 0.02
- *inc* - Acceleration factor step, default value: 0.02
- *max* - Acceleration factor maximum, default value: 0.2

styles:

- *color* - Dots color, default value: '#016ab3'
- *bold* - Use circles, default value: false

Pivot Points

name: 'PivotPoints'

styles:

- *color* - Levels color, default value: '#016ab3'

- *bold* - Levels bold, default value: false

Relative Strength Index

name: 'RelativeStrengthIndex'

settings:

- *n* - Length, default value: 15
- *overbought* - Overbought, default value: 70
- *oversold* - Oversold, default value: 30

styles:

- *color* - Line color, default value: '#65b22e'
- *bold* - Line bold, default value: false
- *colorBg* - Background color, default value: 'rgba(174,208,131,0.2)'

Simple Moving Average

name: 'SimpleMovingAverage'

settings:

- *n* - Length, default value: 15

styles:

- *color* - Line color, default value: '#ee7d00'
- *bold* - Line bold, default value: false

Standard Deviation

name: 'StandardDeviation'

settings:

- *n* - Length, default value: 15

styles:

- *color* - Line color, default value: '#3ab5f9'
- *bold* - Line bold, default value: false

Stochastic Oscillator

name: 'StochasticOscillator'

settings:

- *k* - %K period, default value: 15
- *d* - %D period, default value: 3
- *s* - Smooth, default value: 3
- *upper* - Upper band, default value: 80
- *lower* - Lower band, default value: 20

styles:

- *colorK* - K line color, default value: '#e20613'
- *boldK* - K line bold, default value: false
- *colorD* - D line color, default value: '#ee7d00'
- *boldD* - D line bold, default value: false

- *colorBg* - Background color, default value: 'rgba(255,202,0,0.2)'

Williams Alligator

name: 'WilliamsAlligator'

settings:

- *nJaw* - Jaw Length, default value: 21
- *offsetJaw* - Jaw Offset, default value: 8
- *nTeeth* - Teeth Length, default value: 13
- *offsetTeeth* - Teeth Offset, default value: 5
- *nLips* - Lips Length, default value: 8
- *offsetLips* - Lips Offset, default value: 3

styles:

- *colorJaw* - Jaw line color, default value: '#aed083'
- *boldJaw* - Jaw line bold, default value: false
- *colorTeeth* - Teeth line color, default value: '#e20613'
- *boldTeeth* - Teeth line bold, default value: false
- *colorLips* - Lips line color, default value: '#006ab4'
- *boldLips* - Lips line bold, default value: false

Zig Zag

name: '*ZigZag*'

settings:

- *p* - Percentage, default value: 1

styles:

- *color* - Line color, default value: '#9552c1'
- *bold* - Line bold, default value: false

Drawings

The drawings API is accessible via the *drawings* property of the chart (see more details in the Individual Chart API chapter).

When adding drawings, omitted parameters will be set to default values:

```
// Add a vertical line at the current timestamp
chart[0].drawings.add(chart.drawings.VerticalLine,
     [{ t: Date.now(), v: 0 }], // Points data
     { bold: true } // Drawing parameters
);
```

Important: All color parameters must be valid CSS color string values!

Note that if points data is omitted then the standard procedure of adding points on the chart will be initiated.

In order to access a particular drawing's API, first in is necessary to obtain the drawing object:

```
// Get the first drawing of active chart and display its name and points
let drawing = chart.drawings.find()[0];
if (drawing != null) console.log(drawing.name);

// Get all Forecast drawings and display their points data
chart.drawings
     .find(drawing => drawing.name === 'Forecast')
     .forEach((drawing) => {
          console.log(drawing.points);
     });
```

API:

- *drawing.name* - full drawing name;
- *drawing.params* - contains all drawing settings and styles;
- *drawing.drop()* - deletes the drawing;

You can change values of the *drawings.params* object in order to change the drawing parameters.

Here is the list of all available drawings at the moment:

ABCD Pattern
name: 'ABCDPattern'
points required: 4
styles:
- *color* - Lines color, default value: '#df1120'
- *bold* - Lines bold, default value: false

- *colorBg* - Background color, default value: 'rgba(130,138,155,0.2)'

Arrow

name: 'Arrow'
points required: 2

styles:

- *colorBg* - Background color, default value: 'rgba(221,23,41,0.2)'

Circle

name: 'Circle'
points required: 2
styles:

- *color* - Stroke color, default value: '#df1120'
- *bold* - Stroke bold, default value: false
- *colorBg* - Background color, default value: 'rgba(130,138,155,0.2)'

Fibonacci Arcs

name: 'FibonacciArcs'
points required: 2
styles:

- *colorLevel0236* - Level 0.236 color, default value: '#eb202d'
- *boldLevel0236* - Level 0.236 bold, default value: false
- *colorLevel0382* - Level 0.382 color, default value: '#64b12d'
- *boldLevel0382* - Level 0.382 bold, default value: false
- *colorLevel05* - Level 0.5 color, default value: '#eb5f9e'
- *boldLevel05* - Level 0.5 bold, default value: false
- *colorLevel0618* - Level 0.618 color, default value: '#ffca00'
- *boldLevel0618* - Level 0.618 bold, default value: false
- *colorLevel0786* - Level 0.786 color, default value: '#00afea'
- *boldLevel0786* - Level 0.786 bold, default value: false
- *colorLevel1* - Level 1 color, default value: '#eb202d'
- *boldLevel1* - Level 1 bold, default value: false
- *colorLevel1618* - Level 1.618 color, default value: '#006ab4'
- *boldLevel1618* - Level 1.618 bold, default value: false
- *colorLevel2618* - Level 2.618 color, default value: '#ed7c00'
- *boldLevel2618* - Level 2.618 bold, default value: false
- *colorBg* - Background color, default value: 'rgba(130,138,155,0.2)'

Fibonacci Retracement

name: 'FibonacciRetracement'
points required: 2
styles:

- *colorLevel0* - Level 0 color, default value: '#df1120'
- *boldLevel0* - Level 0 bold, default value: false
- *colorLevel0236* - Level 0.236 color, default value: '#eb202d'
- *boldLevel0236* - Level 0.236 bold, default value: false
- *colorLevel0382* - Level 0.382 color, default value: '#64b12d'
- *boldLevel0382* - Level 0.382 bold, default value: false
- *colorLevel05* - Level 0.5 color, default value: '#eb5f9e'
- *boldLevel05* - Level 0.5 bold, default value: false
- *colorLevel0618* - Level 0.618 color, default value: '#ffca00'
- *boldLevel0618* - Level 0.618 bold, default value: false
- *colorLevel0786* - Level 0.786 color, default value: '#00afea'
- *boldLevel0786* - Level 0.786 bold, default value: false
- *colorLevel1* - Level 1 color, default value: '#eb202d'
- *boldLevel1* - Level 1 bold, default value: false
- *colorLevel1618* - Level 1.618 color, default value: '#006ab4'
- *boldLevel1618* - Level 1.618 bold, default value: false
- *colorLevel2618* - Level 2.618 color, default value: '#ed7c00'
- *boldLevel2618* - Level 2.618 bold, default value: false
- *colorBg* - Background color, default value: 'rgba(130,138,155,0.2)'

Flag

name: 'Flag'
points required: 5
styles:

- *colorAdvance* - Advance color, default value: '#64b12d'
- *colorDecline* - Decline color, default value: '#eb202d'
- *opacity* - Background opacity, default value: 0.2

Forecast

name: 'Forecast'
points required: 2
styles:

- *colorSuccess* - Success color, default value: '#64b12d'
- *colorFailure* - Failure color, default value: '#eb202d'
- *opacity* - Background opacity, default value: 0.2

Gann Box

name: 'GannBox'
points required: 2
styles:

- *colorLevelH0* - Horizontal Level 0 color, default value: '#df1120'
- *boldLevelH0* - Horizontal Level 0 bold, default value: false
- *colorLevelV0* - Vertical Level 0 color, default value: '#df1120'

- *boldLevelV0* - Vertical Level 0 bold, default value: false
- *colorLevelH025* - Horizontal Level 0.25 color, default value: '#0f6cb1'
- *boldLevelH025* - Horizontal Level 0.25 bold, default value: false
- *colorLevelV025* - Vertical Level 0.25 color, default value: '#0f6cb1'
- *boldLevelV025* - Vertical Level 0.25 bold, default value: false
- *colorLevelH0382* - Horizontal Level 0.382 color, default value: '#ec7d21'
- *boldLevelH0382* - Horizontal Level 0.382 bold, default value: false
- *colorLevelV0382* - Vertical Level 0.382 color, default value: '#ec7d21'
- *boldLevelV0382* - Vertical Level 0.382 bold, default value: false
- *colorLevelH05* - Horizontal Level 0.5 color, default value: '#67b138'
- *boldLevelH05* - Horizontal Level 0.5 bold, default value: false
- *colorLevelV05* - Vertical Level 0.5 color, default value: '#67b138'
- *boldLevelV05* - Vertical Level 0.5 bold, default value: false
- *colorLevelH0618* - Horizontal Level 0.618 color, default value: '#1db1e8'
- *boldLevelH0618* - Horizontal Level 0.618 bold, default value: false
- *colorLevelV0618* - Vertical Level 0.618 color, default value: '#1db1e8'
- *boldLevelV0618* - Vertical Level 0.618 bold, default value: false
- *colorLevelH075* - Horizontal Level 0.75 color, default value: '#934e95'
- *boldLevelH075* - Horizontal Level 0.75 bold, default value: false
- *colorLevelV075* - Vertical Level 0.75 color, default value: '#934e95'
- *boldLevelV075* - Vertical Level 0.75 bold, default value: false
- *colorLevelH1* - Horizontal Level 1 color, default value: '#de1020'
- *boldLevelH1* - Horizontal Level 1 bold, default value: false
- *colorLevelV1* - Vertical Level 1 color, default value: '#de1020'
- *boldLevelV1* - Vertical Level 1 bold, default value: false
- *colorBg* - Background color, default value: 'rgba(130,138,155,0.2)'

Gann Fan

name: 'GannFan'
points required: 2
styles:

- *colorLine11* - Line 1/1 color, default value: '#df1120'
- *boldLine11* - Line 1/1 bold, default value: false
- *colorLine12* - Line 1/2 color, default value: '#0f6cb1'
- *boldLine12* - Line 1/2 bold, default value: false
- *colorLine13* - Line 1/3 color, default value: '#ec7d21'
- *boldLine13* - Line 1/3 bold, default value: false
- *colorLine14* - Line 1/4 color, default value: '#934e95'
- *boldLine14* - Line 1/4 bold, default value: false
- *colorLine18* - Line 1/8 color, default value: '#67b138'
- *boldLine18* - Line 1/8 bold, default value: false

- *colorBg* - Background color, default value: 'rgba(130,138,155,0.2)'

Head and Shoulders

name: 'HeadAndShoulders'
points required: 7
styles:
- *color* - Lines color, default value: '#df1120'
- *bold* - Lines bold, default value: false
- *colorBg* - Background color, default value: 'rgba(130,138,155,0.2)'

Horizontal Line

name: 'HorizontalLine'
points required: 1
settings:
- *level* - Price level, default value: calculated after point placement

styles:
- *color* - Line color, default value: '#df1120'
- *bold* - Line bold, default value: false

Line

name: 'Line'
points required: 2
styles:
- *color* - Line color, default value: '#df1120'
- *bold* - Line bold, default value: false

Parallel Channel

name: 'ParallelChannel'
points required: 3
styles:
- *color* - Lines color, default value: '#df1120'
- *bold* - Lines bold, default value: false
- *colorBg* - Background color, default value: 'rgba(130,138,155,0.2)'

Pitchfork

name: 'Pitchfork'
points required: 3
styles:
- *color* - Base line color, default value: '#df1120'
- *bold* - Base line bold, default value: false
- *colorMiddle* - Middle lines color, default value: '#fa7f54'
- *boldMiddle* - Middle lines bold, default value: false
- *colorBorder* - Border lines color, default value: '#ffd35b'

- *boldBorder* - Border lines bold, default value: false
- *colorBg* - Background color, default value: 'rgba(130,138,155,0.2)'

Text

name: 'Text'
points required: 2
settings:

- *text* - Text to display, default value: ''
- *color* - Text color, default value: '#df1120'
- *wrap* - Wrap text using boundaries, default value: true
- *bold* - Bold text, default value: false
- *fontSize* - Font size, default value: 12
- *colorBg* - Background color, default value: 'rgba(130,138,155,0)'
- *align* - Text alignment, default value: 'left', supported values: 'left', 'center', 'right'

Trade

name: 'Trade'
points required: 3
styles:

- *colorEntry* - Entry point color, default value: '#1cb5ee'
- *boldEntry* - Entry point bold, default value: false
- *colorProfit* - Take profit color, default value: '#65b230'
- *boldProfit* - Take profit bold, default value: false
- *colorLoss* - Stop loss color, default value: '#e20613'
- *boldLoss* - Stop loss bold, default value: false

Triangle

name: 'Triangle'
points required: 3
styles:

- *color* - Stroke color, default value: '#df1120'
- *bold* - Stroke bold, default value: false
- *colorBg* - Background color, default value: 'rgba(130,138,155,0.2)'

Vertical Line

name: 'VerticalLine'
points required: 1
styles:

- *color* - Line color, default value: '#df1120'
- *bold* - Line bold, default value: false

Assets and Schedules

It is possible (and highly recommended) to specify several parameters for each asset. This can be done by either passing an *assets* Map object into the options object used in the chart constructor (see Chart Instance Options) or via the chart application instance method *chart.assets.set* (see General API).

symbol
: Specifies the asset symbol used in the asset search (first toolbar element by default).

name
: Defines the asset display name (also used in asset search).

decimals
: Specifies the number of digits after the decimal point (price precision).

schedule
: Defines the asset's trading hours. Trading hours are used by the chart for interpolating timestamps outside loaded data and properly handling new rates when using the *chart.addRates* method.

  Each schedule contains the following properties:
  - *timezone* - the timezone in which trading hours are specified;
  - *days* - an array of seven days, with each day is represented as an array of time intervals (in minutes);
  - *offset* - the start time for daily candles (usually matches the trading session start time).

Important: Schedules for the assets must be properly set in order to build new candles correctly and avoid candle timeframe misalignment, which can lead to candle duplication in some cases.

Here is an example of setting assets data using both methods:

```
let chart = new T4PChart('chartDiv', {
     assets: new Map([
          ['BTCUSD', // Asset id, usually symbol is used
               {
                    symbol: 'BTCUSD',
```

```
                    name: 'Bitcoin',
                    decimals: 2
               }
               // Schedule is omitted which means a default
               one will be used
          ],
     ])
})

chart.assets.set('EURUSD', {
     symbol: 'EURUSD',
     name: 'Euro / US Dollar',
     decimals: 5,
     schedule: {
          timezone: 'America/New_York',
          days: [
               [], // Sunday, no trading hours
               [{from: 8 * 60, to: 24 * 60}],
               // Monday, session opens at 8:00 NY time
               [{from: 0, to: 24 * 60}], // Tuesday
               [{from: 0, to: 24 * 60}], // Wednesday
               [{from: 0, to: 24 * 60}], // Thursday
               [{from: 0, to: 17 * 60}],
               // Friday, session closes at 17:00 NY time
               [], // Saturday, no trading hours
          ],
          offset: 8 * 60 // Means daily candles will start at
          8:00 NY time
     }
});
```

The default schedule consists of seven days with 24-hour intervals { *from*: 0, *to*: 1440 }, an offset set to zero, and the timezone set to 'UTC'.

Candle Data Management

The data management API is accessible through the special methods of the chart application instance (see General API) and the *candles* property of individual charts (see Individual Chart API).

chart.rates(assetId, timestamp, rates, [volume])
: Updates the corresponding candle or creates a new one based on the provided *rates* for the specified asset.

For individual charts (keep in mind that the application supports a multi-chart layout):

chart.candles.set(candles) OR
chart[index].candles.set(candles)
: Sets or updates the candles data for the asset and timeframe of the active chart OR the chart at the specified index.

chart.candles.get(fromTimestamp, toTimestamp) OR
chart[index].candles.get(fromTimestamp, toTimestamp)
: Retrieves candle data within the specified time range, from *fromTimestamp* to *toTimestamp* for the active chart OR the chart at the specified index.

chart.candles.get(timestamp, nBefore, nAfter) OR
chart[index].candles.get(timestamp, nBefore, nAfter)
: Gets candle data around a specified timestamp, with *nBefore* candles before and *nAfter* candles after the timestamp for the active chart OR the chart at the specified index.

chart.candles.get() OR
chart[index].candles.get()
: Retrieves all candle data currently displayed on the active chart OR the chart at the specified index.

Important: Candle data is stored for asset/timeframe values, not for individual charts. If there are two or more charts with the same symbol and timeframe, all will be affected by setting candles for any of them.

The Recommended Way of Handling Candle Data

It is strongly recommended to load new candles only in response to the CANDLES_REQUEST event (see more about events in the Events API chapter).

```
chart.events.add(chart.events.CANDLES_REQUEST, async (symbol,
timeframe, timestamp, amount) => {
     if (timestamp == null) timestamp = Date.now();
     // Request necessary data from the candle server
     let response = await dataProvider.requestCandles(
          timestamp, amount
     );
     // Map data if necessary
     let candles = response.candles.map(c => ({
          timestamp: new Date(c.datetime).getTime(),
          open: c.open,
          high: c.high,
          low: c.low,
          close: c.close,
          volume: c.volume
     }));
     // Return candle data
     return candles;
});
```

Note that the *volume* property of the candle object is optional, while other fields are mandatory.

New rates are handled via the *chart.addRates()* method:

```
// Using some service that allows subscribing to new rates
ratesProvider.subscribe((rate) => {
     chart.rates(
          rate.assetId,
          rate.timestamp,
```

```
            // A single rate value might be used as well;
            however, if rates are being aggregated for
            optimization purposes, it is necessary to specify
            the whole range of values that occurred
            [rate.first, rate.min, rate.max, rate.last]
      );
});
```

Important: Do not forget to update all symbols present on the multi-chart layout. It is possible to track symbol changes via events.ASSET_CHANGE and events.LAYOUT_CHANGE events:

```
function subscribeToRates() {
      let assets = [];
      for (let i = 0; i < chart.length; i++) {
            if (assets.indexOf(chart[i].asset) === -1)
            assets.push(chart[i].asset);
      }
      // Pass the list of assets to receive updates for
      ratesProvider.subscribeToAssets(assets);
}

chart.events.add(chart.events.ASSET_CHANGE, subscribeToRates);
chart.events.add(chart.events.LAYOUT_CHANGE,
subscribeToRates);
```

In some rare cases you might want to directly set candles via the *chart.candles.set()* method. For example, if it is necessary to show some static data on the chart.

Important: Schedules for the assets must be properly set in order to build new candles correctly and avoid candle timeframe misalignment, which can lead to candle duplication in some cases (see details in Assets and Schedules).

Events API

Chart events allows you to subscribe to various events within the chart application, enabling you to handle user interactions, data requests, and changes to the chart's state.

To access the Events API, use the *chart.events* property (see General API). The Events API provides methods to add and remove event listeners for specific events.

Available methods:

- *chart.events.add(event, handler)* - Adds an event listener for a specific event.
- *chart.events.remove(event, handler)* - Removes a previously added event listener.

Example usage:

```
function onAssetChange(asset, index) {
     console.log(`Asset changed to ${asset} on the chart
     #${index + 1}`);
}

// Add an event listener for asset changes
chart.events.add(chart.events.ASSET_CHANGE, onAssetChange);

// Remove the event listener
chart.events.remove(chart.events.ASSET_CHANGE, onAssetChange);
```

Here is the list of supported events:

CANDLES_REQUEST

Triggered when the chart requires candle data for a specific symbol and timeframe. This event is crucial for integrating the chart with your data source.

Handler parameters: *(symbol, timeframe, timestamp, amount)*

- *symbol* - the symbol for which candles are requested;
- *timeframe* - the timeframe (in seconds) for the candles;
- *timestamp* - the timestamp around which data is requested. Can be null, indicating current data is needed;
- *amount* - the number of candles requested.

The handler should return a Promise that resolves to an array of candle objects. Each candle object should have the following properties:

- timestamp - the opening time of the candle;
- open - the opening price;
- high - the highest price;
- low - the lowest price;
- close - the closing price;
- volume - (optional) the trading volume during the candle period.

CANDLES_LOADED

Triggered after candle data has been successfully loaded into the chart.

Handler parameters: (*symbol, timeframe, minTimestamp, maxTimestamp*)

- *symbol* - the symbol for which candles were loaded;
- *timeframe* - the timeframe of the candles;
- *minTimestamp* - the earliest timestamp among the loaded candles;
- *maxTimestamp* - the latest timestamp among the loaded candles.

LAYOUT_CHANGE

Triggered when the chart layout is changed (e.g., switching from a single chart to a multi-chart layout).

Handler parameters: *(layout)*

- *layout* - the new layout index.

LANGUAGE_CHANGE

Triggered when the application language is changed.

Handler parameters: *(language)*

- *language* - the new language code.

THEME_CHANGE

Triggered when the color scheme (theme) of the chart is changed.

Handler parameters: *(theme)*

- *theme* - the new theme name.

TIMEZONE_CHANGE

Triggered when the application's timezone setting is changed.

Handler parameters: *(timezone)*

- *timezone* - the new timezone identifier.

ACTIVE_CHANGE

Triggered when the active (selected) chart is changed in a multi-chart layout.

Handler parameters: *(index)*

- *index* - the index of the new active chart.

ASSET_CHANGE

Triggered when the asset of a chart is changed.

Handler parameters: *(assetId, index)*

- *assetId* - the new asset ID;
- *index* - the index of the chart where the symbol changed.

TIMEFRAME_CHANGE

Triggered when the timeframe of a chart is changed.

Handler parameters: *(timeframe, index)*

- *timeframe* - the new timeframe (in seconds);
- *index* - the index of the chart where the timeframe changed.

TYPE_CHANGE

Triggered when the chart type is changed (e.g., from 'candles' to 'line').

Handler parameters: *(type, index)*

- *type* - the new chart type;
- *index* - the index of the chart where the type changed.

INDICATOR_ADD

Triggered when a new technical indicator is added to a chart.

Handler parameters: *(indicator, index)*

- *indicator* - the indicator object that was added;
- *index* - the index of the chart where the indicator was added.

INDICATOR_CHANGE

Triggered when an existing indicator parameters are changed.

Handler parameters: *(indicator, index)*

- *indicator* - the indicator object that was updated;
- *index* - the index of the chart where the indicator was updated.

INDICATOR_DROP

Triggered when a technical indicator is removed from a chart.

Handler Parameters: *(indicator, index)*

- *indicator* - the indicator object that was removed;
- *index* - the index of the chart where the indicator was removed.

DRAWING_ADD

Triggered when a new drawing (technical analysis figure) is added to a chart.

Handler parameters: *(drawing, index)*

- *drawing* - the drawing object that was added;
- *index* - the index of the chart where the drawing was added.

DRAWING_CHANGE

Triggered when an existing drawing parameters are changed.

Handler parameters: *(drawing, index)*

- *drawing* - the drawing object that was updated;
- *index* - the index of the chart where the drawing was updated.

DRAWING_DROP

Triggered when a drawing (technical analysis figure) is removed from a chart.

Handler parameters: *(drawing, index)*

- *drawing* - the drawing object that was removed;
- *index* - the index of the chart where the drawing was removed.

POSITION_ADD

Triggered when a new trading position is added to a chart.

Handler parameters: *(position, index)*

- *position* - the position object that was added;

- *index* - the index of the chart where the position was added.

POSITION_CHANGE

Triggered when an existing trading position parameters are changed.

Handler parameters: *(position, index)*

- *position* - the position object that was updated;
- *index* - the index of the chart where the position was updated.

POSITION_DROP

Triggered when a trading position is removed from a chart.

Handler parameters: *(drawing, index)*

- *position* - the position object that was removed;
- *index* - the index of the chart where the position was removed.