

JavaScript   
Stock Charting + Technical Analysis Library

Version 9.2013

Contents

[About 6](#_Toc360547884)

[Data 6](#_Toc360547885)

[Package 7](#_Toc360547886)

[The Files 7](#_Toc360547887)

[sample.html 7](#_Toc360547888)

[stx-multi.html 7](#_Toc360547889)

[sample\_style.css 7](#_Toc360547890)

[SAMPLEDATA.js 7](#_Toc360547891)

[SAMPLEINTRADAY.js 7](#_Toc360547892)

[stxUtilities.js 7](#_Toc360547893)

[stxMarket.js 8](#_Toc360547894)

[stxStudies.js 8](#_Toc360547895)

[stxChart.js 8](#_Toc360547896)

[stxKernelOs.js 8](#_Toc360547897)

[stxI18N.js 8](#_Toc360547898)

[stxModulus.js 8](#_Toc360547899)

[Third Party Libraries 8](#_Toc360547900)

[excanvas.js 8](#_Toc360547901)

[Intl.js 9](#_Toc360547902)

[Getting Started 9](#_Toc360547903)

[Interface Usage 10](#_Toc360547904)

[Panning 10](#_Toc360547905)

[Zooming 10](#_Toc360547906)

[Drawing 10](#_Toc360547907)

[Panels 10](#_Toc360547908)

[Periodicity 10](#_Toc360547909)

[Implementation 11](#_Toc360547910)

[Internals 12](#_Toc360547911)

[Data Engine 13](#_Toc360547912)

[Streaming 14](#_Toc360547913)

[Extensibility 15](#_Toc360547914)

[Injection API: Prepend and Append 15](#_Toc360547915)

[Change Callbacks 16](#_Toc360547916)

[DOM Extension 16](#_Toc360547917)

[Customizing Market Defaults 16](#_Toc360547918)

[Internationalization 17](#_Toc360547919)

[Creating Custom Indicators 17](#_Toc360547920)

[Examples 19](#_Toc360547921)

[What's New 20](#_Toc360547922)

[Version 7-2013 20](#_Toc360547923)

[Version 5-2013 21](#_Toc360547924)

[Version 3-2013 22](#_Toc360547925)

[Version 2-2013 22](#_Toc360547926)

[Native Methods 23](#_Toc360547927)

[Chart 23](#_Toc360547928)

[Return to “Home” 23](#_Toc360547929)

[Create X Axis 23](#_Toc360547930)

[Create Y Axis 24](#_Toc360547931)

[Create Volume Chart (panel, not overlay) 24](#_Toc360547932)

[Initialize Display 24](#_Toc360547933)

[Update Price Label (HR) with Crosshair Values 24](#_Toc360547934)

[Set Chart Periodicity 25](#_Toc360547935)

[Mouse Move 25](#_Toc360547936)

[Compute Consolidated Quote 25](#_Toc360547937)

[Run Ticks 26](#_Toc360547938)

[Display the Chart 26](#_Toc360547939)

[Draw 26](#_Toc360547940)

[Data Set - Create 26](#_Toc360547941)

[Draw Current Price 26](#_Toc360547942)

[Data – Append Quotes (Streaming) 27](#_Toc360547943)

[Panels 27](#_Toc360547944)

[Panels – Calculate Position 27](#_Toc360547945)

[Panel - Create 27](#_Toc360547946)

[Panel – Stack Panels 27](#_Toc360547947)

[Panels - Draw 28](#_Toc360547948)

[Crosshairs 28](#_Toc360547949)

[Create Crosshairs 28](#_Toc360547950)

[Set Crosshair Colors 28](#_Toc360547951)

[Display Crosshairs 28](#_Toc360547952)

[UnDisplay Crosshairs 29](#_Toc360547953)

[Set Measurement Display 29](#_Toc360547954)

[Drawing 29](#_Toc360547955)

[Abort Current Drawing Operations 29](#_Toc360547956)

[Create a Vector 29](#_Toc360547957)

[Draw a Vector 30](#_Toc360547958)

[Draw Vectors 30](#_Toc360547959)

[User Interaction Events 30](#_Toc360547960)

[Mouse Up 30](#_Toc360547961)

[Mouse Down 30](#_Toc360547962)

[Touch – Single Click 31](#_Toc360547963)

[Touch – Double Click 31](#_Toc360547964)

[Touch – Move 31](#_Toc360547965)

[Touch – Start 31](#_Toc360547966)

[Touch – End 31](#_Toc360547967)

[Non Injectable Methods 32](#_Toc360547968)

[Chart 32](#_Toc360547969)

[Chart - Initialize 32](#_Toc360547970)

[Chart – load master data 32](#_Toc360547971)

[Chart – Resize Chart 32](#_Toc360547972)

[Zoom In 32](#_Toc360547973)

[Zoom Out 32](#_Toc360547974)

[Chart – Resize Canvas 32](#_Toc360547975)

[Chart – Correct if off edge 33](#_Toc360547976)

[Create Canvas Watermark 33](#_Toc360547977)

[Resolve Y location 33](#_Toc360547978)

[Back out Y location 33](#_Toc360547979)

[Back out X location 33](#_Toc360547980)

[Chart - 33](#_Toc360547981)

[Chart – Remove overlay 34](#_Toc360547982)

[Chart – Display Sticky Message 34](#_Toc360547983)

[Chart – Compute Length 34](#_Toc360547984)

[Chart – Locate Tick Position 34](#_Toc360547985)

[Chart – Compute Pixel Given A Price 34](#_Toc360547986)

[Chart – Compute Price Given A Pixel 34](#_Toc360547987)

[Chart – Compute Pixel Given A Tick 34](#_Toc360547988)

[Chart – Compute Tick Given a Pixel 35](#_Toc360547989)

[Chart – Return tick from date 35](#_Toc360547990)

[Chart – Return date from Tick 35](#_Toc360547991)

[Chart – Set Adjusted Prices 35](#_Toc360547992)

[Chart – Set Chart Type 35](#_Toc360547993)

[Chart – Hide Dates in X Axis 35](#_Toc360547994)

[Panels 36](#_Toc360547995)

[Create Panel Watermark 36](#_Toc360547996)

[Panel – Resize 36](#_Toc360547997)

[Panel – Down 36](#_Toc360547998)

[Panel - Up 36](#_Toc360547999)

[Panel - Solo 36](#_Toc360548000)

[Panel - Close 36](#_Toc360548001)

[Crosshairs 37](#_Toc360548002)

[Crosshairs - Hide 37](#_Toc360548003)

[Crosshairs - Show 37](#_Toc360548004)

[Drawing 37](#_Toc360548005)

[Drawing – Plot a Line, Segment, or Ray 37](#_Toc360548006)

[Drawing – Construct vector 37](#_Toc360548007)

[Drawing – Delete Highlighted Drawing 37](#_Toc360548008)

[Drawing – Delete Vector 37](#_Toc360548009)

[Drawing – Highlight Vector 38](#_Toc360548010)

[Drawing – Un-highlight Vector 38](#_Toc360548011)

[Drawing – Clear All 38](#_Toc360548012)

[CSS / Styling 38](#_Toc360548013)

[CSS – Get CSS Color 38](#_Toc360548014)

[CSS – Get Font Size 38](#_Toc360548015)

[CSS – Set canvas color, width, alpha 38](#_Toc360548016)

[CSS – Set Font 39](#_Toc360548017)

[CSS – Get Class Style 39](#_Toc360548018)

# About

This HTML5 library provides developers and web designers with a complete HTML 5 stock charting solution. The library is compatible with Safari, Firefox, Chrome and Internet Explorer versions 8 and later. It is also touch aware, allowing mobile users on tablets or smart phones to access all of the charting features. Built in features include interactive chart navigation, drawing tools and a full suite of technical indicators.

**IMPORTANT!** For users of previous versions please refer to the “What's New” section at the end of this document for instructions on how to migrate your application to this new version.

The HTML5 API is a professional grade API that can be used to build both basic charts as well as advanced charts that can be used in professional workstations. The API is extensive and provides both high level convenience functions as well as low level access that provides developers with fine levels of control. Please feel to contact us for support or questions about the API. We are always happy to provide sample code for any particular charting problem you are trying to solve.

The library comes with several sample implementations that you can use depending on your needs and experience:

stx-demo.html – An off the shelf stock charting widget complete with menuing, themes, preference storage. Use this to quickly integrate professional quality stock charts into your website with minimal coding.

stx-quickstart.html – For developers lookup to use the library as an API, the quickstart shows you how to programmatically create a chart with just a few lines of JavaScript and supporting HTML code.

stx-sample.html – A rough integration for developers with an abundance of sample code that demonstrates how to integrate and extend the API. This is a good starting place for complex integrations and customizations.

All of the componentry within the sample files may be utilized or rewritten to suit your needs but in its essence, the library simply requires that you create a DIV tag somewhere on your site, a place for the chart to reside, and everything else is under your control.

# Data

The library does not include securities data. Developers must provide their own data in the required format. Sample code is included that demonstrates how to convert data into the required format. A sample data engine is provided that integrates Yahoo Finance API historical data.

For an additional fee we can provide API access to End of Day or Real-Time data. Please contact us for more information.

# Package

The zip file includes complete working examples in plain HTML. No external libraries are required to utilize the library although it is compatible with popular JavaScript libraries such as JQuery. Please refer to the end of this document for a complete API reference with each method and member of the library. Separate from the initial zip file, you will receive a copy of stxKernelOs.js which will be locked to the domain that you specified when you purchased the library.

## The Files

### stx-demo.html

A complete working “widget” for displaying stock charts on your website. This demo makes extensive use of a basic UI for menuing, color themes, dialog management and preference storage (stxUI.js) to provide an off-the-shelf charting solution that can be integrated very quickly.

### stx-quickstart.html

A minimum programmatic implementation with just a few lines of JavaScript and supporting HTML that displays a functional stock chart out of the box.

### stx-sample.html

A complete working example using the charting API with working examples that demonstrate how to utilize and extend the API. It contains a sample engine that loads a sample chart from a JavaScript array as well as a sample engine to display data from the Yahoo Finance API. Please see detailed documentation below that details this file. \*\* Please note that the Yahoo Finance API is generally not available on market open mornings and is also subject to bandwidth restrictions by IP address.

### stx-multi.html

A working example that demonstrates how to implement multiple charts within a single browser window. Note that in order to keep the example as straightforward as possible, that this is a simplified, stripped down implementation.

### stx-demo.css

The default css that accompanies stx-demo.html. These elements may be customized to meet your needs.

### stx-demo-theme-1.css, stx-demo-theme-2.css

Default “themes” that users can select in the stx-demo.html widget. You may add, remove or customize these.

### sample\_style.css

Chart componentry can be customized through css style sheets. This file contains the default styles for the library that may be overridden. This file also contains comments that describe usage of the styles.

### SAMPLEDATA.js

A sample file containing a JavaScript array of historical securities data for the ticker symbol SPY. Please refer to the code in sample.html to learn how to convert such data for use by the charting engine.

### SAMPLEINTRADAY.js

A sample file containing a JavaScript array of 5 minute intervals.

### stxUtilities.js

A basic utilities library that contains many useful functions for building JavaScript applications.

### stxMarket.js

Contains functionality related specifically to financial markets. This library contains market calendars and market open and close times that may be overridden or supplemented. Note that scheduled holidays should be updated on an annual basis. Unscheduled market closings should also be entered as soon as possible after the fact because these will affect the accuracy of drawings such as trendlines that pass through those dates.

### stxStudies.js

A technical indicators interface. Developers control which indicators are available through basic HTML (i.e. by including or excluding them from the studies menu). Developers may also implement and integrate custom indicators using the studies API.

### stxChart.js

This file contains the basic interface for drawing and manipulating stock charts. This file goes hand in hand with stxKernelOs.js. (Programmers familiar with C should think of stxChart.js as the header file and stxKernelOs.js as the object file).

### stxKernelOs.js

This file contains the core charting code and technical indicator library. The code is obfuscated and locked to the domain that you’ve registered. A comprehensive “Injection” API allows you to extend and leverage the code.

### stxI18N.js

Internationalization is optionally supported using the ECMA-402 Intl standard. This standard is not widely adopted by browsers yet but is enabled with the inclusion of Intl.js. stxI18N.js provides a template for language translation. See the section on Internationalization.

### stxUI.js

A basic UI implementation including menuing, dialog management, preferences, and storage. The stx-demo.html widget is built using this library.

### stxModulus.js

This library contains the technical indicator engine. It is obfuscated and cannot be used without a functioning stxKernelOs.js.

Various image files for creating the chart controls.

# Third Party Libraries

### excanvas.js

<http://excanvas.sourceforge.net/>

Internet Explorer 8 is supported using the excanvas.js library. Please see commentary in sample.html for specific support of IE8.

### Intl.js

<https://github.com/andyearnshaw/Intl.js>

Provides ECMA-402 Intl support for browsers that do not yet support the standard. A database of locale files is also included in the directory “locale-data” which can be provided on request for those customers who will be internationalizing. The locale data file is 5mb large and only needs to be delivered once.

### iscroll.js

<http://cubiq.org/iscroll-4>

Supports scrolling html elements on tablets, mouse and mousewheel. This slightly modified version supports Windows tablets. It is required to use stxUI.js.

### Timezone-js-master

<https://github.com/mde/timezone-js>

Supports timezone conversion in js. We have pre-compiled the timezone data into json format as file data.js in this directory. These files must be included in order to support multiple timezones.

# 

# Getting Started

Unzip the files and place your copy of stxKernelOs.js (usually sent separately) in the same directory. Note that unless you are using a trial copy, stxKernelOs.js will be domain locked to the domain that you specified when you purchased the library. This means that the library will not function unless it is hosted on that domain. For the purposes of development and evaluation however there are two choices:

a) Develop on a machine within the registered domain. You can do this quickly and easily by manually modifying your development machine’s host file to contain a domain, for instance:

blahblah.mydomain.com 127.0.0.1

b) The library will also function when running directly from the filesystem on your local development box. That is, you can double click on the sample.html file and develop this way:

[file:///my](file:///C:\my)\_director/sample.html

Using either of these methods, once unzipped, open stx-demo.html in a compatible browser. A sample chart should immediately appear. If a chart does not appear then a problem exists with one of the library components. Open a browser debugger window (F12 on Internet Explorer, Firebug, Chrome or Safari development tools) and check the console for any errors.

Once the chart appears you can begin integration or development. We recommend making a copy of stx-sample.html, stx-demo.html or stx-quickstart.html for safekeeping and reference. Then use one of these files as a starting point for your application. Note that there are “chartContainer” html elements that the library depends on so when transferring code out of sample.html be sure to understand those elements (described below under “Implementation”) and bring them along.

# Interface Usage

This charting engine displays interactive charts on an HTML5 canvas. Developers must provide a DOM element (div) which will contain the canvas (in the samples see id=chartContainer). The library does the rest. The basic interface supports:

### Panning

On a web browser, grab with the mouse and pan the chart in the same manner as Google Maps. On touch devices use one finger to pan the chart. If crosshairs are enabled (such as when drawing) then a single finger moves the crosshairs while two fingers pan the chart (on Windows touch devices make sure your two fingers are stacked vertically, as if you were opening or closing a sliding door).

### Zooming

Screen controls are provided to zoom in and out (plus and minus buttons). On touch devices users may also pinch to zoom or expand. Users may zoom out to any level. Zoom-In is limited to the pixel dimensions of the canvas. Zoom functions are available programmatically so you may decide to eliminate the screen controls in favor of other mechanisms.

### Drawing

Crosshairs are automatically enabled whenever a drawing tool is active (drawing tools are activated programmatically such as enabling through a menu selection). In browsers, click once to begin a drawing and then click once again to finish. On touch devices, first position the crosshairs and then single tap to begin a drawing. Position the crosshairs a second time and then single tap again to finish the drawing. Tap twice to abandon the drawing. To delete a drawing with a mouse, highlight the drawing and then right click. On touch devices, highlight the drawing and then tap twice or press the trashcan icon that appears when the drawing is highlighted.

### Panels

Studies appear in panels that are stacked on the canvas. Panels may be repositioned by pressing the up or down icons. They may be removed by pressing the trash can icon. The “focus” icon temporarily hides the other panels so as to quickly provide more screen space for evaluating a study. Panels can be resized by grabbing the panel’s “handle” and dragging to the desired size.

### Periodicity

STXChart natively supports minute, daily, weekly and monthly periods. It also supports any compound aggregate of each (such as 3 day periods or 2 week periods). Intraday periods can also be compounded. For instance to display a 16 minute period you may specify 4x4minute intervals. Periodicity can be set programmatically or through a menuing system. The sample files demonstrate how to switch between periods and how to compound periods.

Terminology

We use the term “interval” to describe the base data timeframes while “period” indicates the compounding (multiplication) factor. So for instance a 12 minute chart can be created with an interval of “3 minutes” and a period of “4”.

Usage

The library comes with sample data for both daily (SAMPLEDATA.js) and intraday data (SAMPLEINTRADAY.js). Within stx-sample.html see loadChart() for an example of loading daily data and loadIntradayChart() for an example of loading intraday data.

If setPeriodicityV2() detects that the interval has changed (daily to weekly, or 3 to 5 for instance) then it will attempt to call the callback stxx.dataCallback(). Register this callback if you wish to make changes to masterData with an event driven model.

# Implementation

Developers should provide a DOM element to contain the chart (such as id=chartContainer in sample.html). Initialize STXChart by constructing with a reference to the DOM object:

var stxx=new STXChart($$(“myContainer”);

or by passing the DOM element into the function initializeChart():

initializeChart($$(“myContainer”);

STXChart will automatically expand the canvas size to the boundaries of the DOM element. If the DOM element is ever resized, call resizeChart() to readjust the canvas size. (Note that the container object can be a relative style, such as width=100%. The canvas object will be automatically sized to the container object when the page is rendered. However, if the size of the container object *itself* changes dynamically such as when a user resizes the browser screen then your application must call resizeChart() otherwise the canvas will not be sized appropriately).

HTML5 canvas does not natively support CSS but for convenience STXChart has been developed so that the style of many rendered components can be controlled from a style sheet. For instance the class stx\_xaxis controls the font size, style and color of the text that is drawn along the xaxis. All “stx…” classes must be accessible otherwise those components will not render. Note that not every css capability is supported. Please see the CSS source code comments for specifics.

# Internals

A chart is represented and rendered by a STXChart object. Charts are drawn on an HTML5 canvas (STXChart.chart.canvas). Create a separate STXChart object for each chart you wish to display on the screen (and devote a dedicated container div to each). A second canvas (STXChart.chart.tempCanvas) is layered on top of the primary canvas and is utilized to draw temporary drawings (using a second canvas speeds up rendering while users are resizing their drawings).

The sample implementations include the required DOM componentry that is layered on top of the canvas and repositioned by the STXChart object. These include the icons that control panel position, the panel label, the zoom buttons and the “home” button that appears if you scroll leftward. The “handles” that control panel height are DOM elements as are the crosshairs and the floating price label (floatHR). These DOM elements are loaded with mouseover events that automatically hide the crosshairs when they are in focus and also respond to touch events. *Note that these objects must be contained in the canvas container object (id=chartContainer in sample.html) otherwise they will not appear in the correct positions.* If you are displaying more than one chart on the screen be sure to provide a separate container div, each with a copy of this componentry, for each chart. In the samples, this means is everything within the “chartContainer” div.

You may eliminate the zoom, home and more controls if you wish to provide your own navigational components. The iconsTemplate however is required for the charting engine to function.

Panning and zooming (animation) are effected by redrawing the HTML5 canvas. Modern browsers provide very high performance especially when graphic acceleration is available (nearly all devices). Animation of the chart will appear smooth to end user.

Overall performance depends on the amount of data displayed (width of chart & number of bars) and processing (number of studies) and of course the user’s hardware and browser choice. Performance also can be negatively impacted when developers “plug in” additional DOM components that auto update with panning and zooming events (such as a head’s up display).

The charting engine internally uses the STX.Plotter object whuch caches canvas drawing calls so that they are batched to the GPU. Use this device for accelerated performance when creating new functionality rather than drawing directly to the canvas.

The charting engine supports both IOS style (Apple & Android) and Windows style touch events.

# Data Engine

STXChart is initialized with the function setMasterData(). The required format is a JavaScript array of OHLC objects. The objects must be in the following format:

**Date** – A date or date/time string. Various formats are supported such as YYYY-MM-DD

**Open**

**High**

**Low**

**Close**

**Volume**

**Adj\_Close** – If available the closing price adjusted for splits or dividends. If not available then developers should **not** allow adjusted price mode to be available to end users.

Note that Open, High and Low are not required when only supporting line charts.

*Please note: The charting engine requires an uninterrupted (contiguous) array of datapoints. Please scrub your data so that there are no gaps before calling setMasterData(). We can provide sample code for scrubbing data or consolidating ticks if you require.*

Within STXChart we use the following terminology:

**STXChart.masterData** – The raw data loaded into the charting engine

**STXChart.chart.dataSet** – Data consolidated for periodicity and including calculated values such as indicators. dataSet represents all of the data available when the user pans from beginning to end of a chart.

**STXChart.chart.dataSegment** – The window of data that is displayed on the screen. Essentially a slice of the dataSet. As the user scrolls the dataSegment is continually updated. It can be thought of as a viewport into the dataSet.

Studies are generally computed on a dataSet. They are computed once, so that scrolling can remain quick for the end user. Computed values are stored in the dataSet and thus available programmatically. *Performance note*: studies are recalculated whenever appendMasterData() is called.

When a new periodicity is selected (setPeriodicity()) a new dataSet is created. This allows indicators to automatically work with different periodicities.

# Streaming

A dataSet may be updated at any time by modifying the masterData and then calling createDataSet() [followed by draw() ], however to provide a streaming or continually updating interface, we recommend using appendMasterData().

*Please note that appendMasterData() takes an array of OHLC objects. When streaming a single bar at a time simply pass an array containing a single object.*

var newData=[{Date:”01-05-2015 09:32”, Open:105, High:107, Low:104, Close:106, Volume:1000}];

appendMasterData(newData);

If the first OHLC object in the array shares the same time signature as the latest OHLC object in the current chart then it will replace that bar rather than appending (streaming updates).

Please note that you must pass in a complete OHLC bar. **appendMasterData() does not support last sale ticks** (that is, it will not automatically keep track of OHLC for you).

Sample.html contains a dummy engine that simulates streaming charts. Uncomment the line “//streamSimulation()” to enable the streaming simulation.

Performance Note: appendMasterData() will create a new dataSet and thus recalculate all studies each time it is called. Limit the overall size of the dataSet in order to maximize performance.

# Extensibility

## Injection API: Prepend and Append

STXChart supports developer extension through the use of prepend and append functions. A developer function may be registered with the object to be called before (prepend) or after (append) each built in STXChart method. This API provides developers with comprehensive control over the charting engine: the ability to supplement, override or eliminate functional components.

The system will call a user defined prepend or append function with the same arguments as the built in function. “this” will also be set to the actual STXChart object. In this manner, the user defined function behaves exactly as if it were a member function of the object itself. A prepended function can choose to bypass the built in method entirely by returning “true”, thus providing developers with the ability to completely override native behavior.

As an example, a developer could choose to develop a completely custom yaxis:

function myYaxis(){

//draw my yaxis using this.chart.canvas

return true; // override the built in y axis

}

STXChart.prototype.prepend(“createYAxis”,myYAxis); // inject my functionality

Another developer wishing to customize the study panels could register a function to be called before the internal function drawPanels() was called. That function might for instance create custom dom elements and position then appropriately:

function myPanelSupplement(){

for(var panel in this.panels){ // iterate through panels

// Add or move a dom object

}

}

STXChart.prototype.append(“drawPanels”, myPanelSupplement);

The stx-sample.html file contains many examples of how to utilize the injection API to accomplish commonly requested functionality.

## Change Callbacks

Developers may register to receive events whenever the user adjusts the layout of a chart or whenever a vector (drawing) is added or removed. This is accomplished by assigning a callback function to STXChart.changeCallback. The function must be of the form:

fc(stxChart, change)

“stxChart” will be the STXChart object that is registering an event. “change” will be one of:

“layout” - A change to the layout of the screen has occurred. Check stxx.layout

“drawing” - A drawing operation has begun

“vector” - A drawing operation has completed. Check stxx.chart.memory for the currently displayed vectors.

The library does not automatically retain the previous state before a change event. Developers should maintain the prior state externally (by cloning stxx.layout and stxx.chart.memory) if such functionality is required (for instance to save configurations on a server as a JSON object).

## DOM Extension

Extending the chart through the positioning of tappable DOM objects is a good way to add functionality without dealing with the intracasies of the canvas. A few things must be noted when using this technique however.

Use STXChart.openDialog to “turn off” chart interactivity. For instance, if popping up a dialog box that is within the boundaries of the chart, set STXChart.openDialog to a non empty string. All mouse or touch events within the chart area will be ignored as long as this string is not blank. Set it back to “” in order to turn the chart event management back on.

Tappable DOM objects may be added within chart boundaries so long as some protocols are observed. Set STXChart.cancelTouchSingleClick to true whenever one of these elements is tapped. This will bypass the tap event that would normally begin a drawing. Also call undisplayCrosshairs() and displayCrosshairs() for mouseover and mouseout event. Finally, be sure to set the z-index appropriately otherwise objects may not reliably receive their mouse events on some browsers.

Given x and y positions on a chart you can utilize STXChart.resolveX(x) and STXChart.resolveY(y) to find the absolute positions on the screen. For instance to display a DOM object at a precise point given a price and date you could use the following code (assuming absolute positioning at the <body> level):

var chartx=stx.pixelFromDate(dt);

var charty=sts.pixelFromPrice(price);

var screenx=stx.resolveX(chartx);

var screeny=stx.resolveY(charty);

domObject.style.top=screenx;

domObject.style.left=screeny;

Note that when implementing clickable html elements, in general, onclick events provide more predictable behavior on touch devices than ontouchend. The downside to onclick is a ~200ms response delay which provides for a slightly less responsive interface. Experiment with both but be cognizant of the fact that ontouchend may behave erratically in certain scenarios.

## Customizing Market Defaults

By default the system supports US Equity market holidays and hours as well as Forex and Futures markets that trade 24 hours. The STXMarket.js file contains the code to handle these configurations (and is automatically aware of closing hours on Sundays). Override or modify the isHoliday() and isHalfDay() static methods if necessary to support international exchanges or other markets.

For intraday charts initialize the STXChart object with the opening and closing hours of an exchange if different than US Equities markets using the beginHour, beginMinute, endHour and endMinute members.

## Internationalization

The charting API supports localization and translation. Localization is supported through the ECMA-402 specification. The shim library Intl.js is included to support the specification on browsers that do not yet have the standard built in. Localization is used to display prices and dates in locale specific format. A full locale file database is included with the Intl.js project.

Translation is supported via the STXChart.translationCallback event handler. Set this to a function that accepts an English word and returns the translated word. For convenience, stxI18N.js provides a basic infrastructure

To implement localization first include the Intl.js library. Then include the locales that you wish to support. Intl.js conveniently provides all locale libraries in JSONP format so supporting multiple locales is as easy as this code:

<script src=”Intl.js></script>

<script src=”locale-data/jsonp/en-US.js”>

<script src=”locale-data/jsonp/de.js”>

// etc, other additional locales.

// You may also load the locale-data files dynamically using JSONP or Ajax

var stxx=new STXChart();

stxx.setLocale(“en-US”); // You may set the locale for the chart at any time

Translation is supported via the STXChart.translationCallback event handler. Set this to a function that accepts an English word and returns the translated word. For convenience, stxI18N.js provides a basic infrastructure for translating via a JSON wordlist. A sample German translation wordlist is provided. The charting engine will look for translations for study names and parameters as defined in stxI18N.js.

## Creating Custom Indicators

The library provides an interface that developers can use to build custom indicators. Indicators may be displayed in a panel or as overlays on the chart. Convenience functions are provided for drawing lines based on a series (STXStudies.displaySeriesAsLine) or histograms (STXStudies.createHistogram). Other visualizations can be coded with some knowledge of HTML5 canvas.

Custom indicators should be registered by adding a StudyDescriptor JSON object to the StudyLibrary. On load, the system will read these descriptors and make them available. Manually create a menu item for any new indicator. Please refer to the STXStudies.js code which is unobfuscated and provides working examples.

STXStudies.studyLibrary[myName]={

"overlay": false,

"initializeFN": fn1,

"calculateFN": fn2,

"seriesFN": fn3,

“tickFN”: fn4,

"inputs": {"Period":14,"Divergence":false},

“outputs”:{“Slow”:”#000000”,”Fast”:”#0000FF”},

“Range”:”bypass”,

“nohorizonta;”:true

};

Each of the functions (fn1,fn2,fn3,fn4) are developer provided functions that initialize, calculate and render the indicator. Be sure to create the StudyDescriptor *below* where those functions are defined in the code!

myName – This is the internal name for the indicator. By default this is what appears in the label for the panel so a short name is recommended (although the panel display label can also be overridden). The actual menu item may be a more descriptive name.

overlay – If true then the indicator will be overlayed on the chart, otherwise a panel will be created for the indicator.

initializeFN – Optional, provide a function that should be called to initialize the indicator when it is first loaded by the app. Usually this can be left undefined as the default functionality will work for most indicators.

calculateFN – Required, this function calculates the data set for the indicator. See STXStudies.calculateKlinger for a working example. You are only guaranteed OHLCV data to be available. This function should set new members in the dataSet which will then be referred to by the rendering methods.

seriesFN, tickFN – These are the possible rendering methods. Please utilize only one as they are mutually exclusive. SeriesFN renders all of the data points in one fell swoop, such as when drawing a moving average line across a chart. tickFN gets called for each bar on the chart, allowing individual drawings such as a volume chart or parabolic sar. If both are left undefined then by default STXStudies.displaySeriesAsLine will be called (if your indicator is a simple line then this will likely work for you). If your indicator displays multiple lines then displaySeriesAsLine will be called for each calculated point. The calculated points are defined by “outputs”.

Inputs – This defines the input types for the indicator. These will be displayed in the dialog box to the user and the results will be available to the calculation. If this is left blank then “period” will be the default indicator. The value defines the default value. Boolean values will create a checkbox while numeric values will create an input box. A string of “ma” will present a select box of moving average types. A string of “field” will present a select box of all of the available fields (open, high, low, etc). You may also include an array of fields in order to create a custom select box. (See the studyLibrary in stxStudies.js for usage examples).

Outputs – This defines the outputs from the calculations and which default color to be used to draw them. These are displayed in the user dialog and then available to the rendering functions.

Range – Optional. By default the min and max of the indicator panel are determined automatically by the function STXStudies.determineMinMax. Range can be optionally set to “0 to 100”, “-1 to 1” or “bypass”. If bypass is chosen then the calculation function must set the range (panel.min and panel.max).

Nohorizontal – Optional. By default, a horizontal line is drawn at the zero point for indicator panels. Set this to true to override this behavior (such as with MACD where the horizontal is to be drawn at the 50% mark rather than at zero).

Parameters – Optional object containing additional parameters required by a study. See the implementation of overbought and oversold zones for RSI as an example.

### Examples

STXStudies.displayMACD=function(stx, sd, quotes) {

STXStudies.createHistogram(stx, sd, quotes, true);

STXStudies.displaySeriesAsLine(stx, sd, quotes);

}

displayMACD is a series display. The stx object is the chart object. sd is the StudyDescriptor. quotes is the dataSegment to display.

STXStudies.displayPSAR=function(stx, sd, x, quotes){

var x0=stx.computePosition(x, 0);

var y0=stx.pixelFromPrice(quotes[x]["Result " + sd.name]);

stx.plotLine(x0, x0+3, y0, y0, sd.outputs["Result"], "segment");

}

displayPSAR is a tick display. The stx object is the chart. sd is the StudyDescriptor. X is the current pointer in the quotes array. sd.outputs[“Result”] will contain the user selected color for the “Result” output. (Result is the default output if none are defined.)

Convenience functions are available for drawing to the chart:

stx.computePosition(x, offet). Returns the horizontal position of the current tick. Offset is the number of pixels left or right from center, usually left zero. (Note that stx.layout.candleWidth contains the current bar width for the chart).

stx.pixelFromPrice(price). Returns the vertical position for a given price.

stx.createBlock(x, w, y, h, cssClass). Draws a block on the screen and obtains the color from the cssClass.

stx.plotLine(x0, x1, y0, y1, color, type). Draws a line on the screen and automatically crops to the edges of the chart display. (To draw on indicator panels requires using native canvas tools). This can take an actual color or a css class containing a color.

stx.canvasColor(cssName). Returns the color given a css class name.

stx.canvasFont(cssName) . Returns a canvas compatible font string give a css class name.

## What's New

### Version 9-2013

Added stx-quickstart.html file.

Added stx-demo.html sample file – supports changing chart colors, enabling themes, smart menus, smart dialogs, symbol lookup widget, Scrolling manager, Timezone selector, Storage manager, drawing toolbar, head’s up display

Added stx-demo.css, stx-demo-theme-1.css, stx-demo-theme2.css files.

Added stxUI.js library file.

Renamed sample.html to stx-sample.html and included more coding examples.

Consolidated images into stx-chart-icons.png and stx-ui-icons.png sprite files.

$$$() convenience function for using css selectors

clearCanvas() renamed STX.clearCanvas(). Now supports clearing of canvas on old (defective) Android devices. Enable this functionality by setting global STXChart.useOldAndroidClear to true.

Default color palette for drawing modified to more useful, web friendly colors

STX.newChild() convenience function for creating new HTML elements

stxStudies – library now supports “auto” colors which will automatically be white or black depending on the brightness of the chart background

STXStudies.quickAddStudy – convenience function for programmatically adding new studies.

RSI now defaults to display of overbought and oversold lines. Override the stxStudies.studyLibrary “parameters” to eliminate or change the defaults.

Studies may now be programmatically created with the “permanent” flag which will prevent users from deleting them.

Optional “yaxis” library parameter can override drawing the y-axis for a study.

Default study y-axis spacing reduced to provide more y-axis points.

Line drawings that go off screen no longer display horizontally at top or bottom of the chart.

Dashed and dotted line drawings now supported.

Timezone support.

beginHour, beginMinute, etc now automatically default to 24 hour trading unless overridden.

minutesInSession now automatically calculated.

STXChart.registeredCharts eliminated in favor of STXChart.registeredContainers which contains references to the HTML containing elements.

showMeasure html element eliminated (consolidated with mSticky)

x-axis drawing and layout improved to eliminate gaps and overlaps

STXChart.formatPrice() now available to format price in locale format

STXChart.constructVector() now takes a “parameters” parameter which describes the width and pattern of the line, segment, etc.

Candle borders now supported.

“hollow” candle chart type now supported.

Panel close icon moved to top of panel, and changed from trashcan to “X”

### Version 7-2013

This version adds support for the following:

Internationalization – See the new section on Internationalization and new files stxI18N.js as well as the Intl.js library.

Multiple Charts – Multiple charts may now be displayed on the screen without the use of iframes. See the new stx-multi.html sample file.

Internet Explorer 8 Support – Using the excanvas.js shim library. Note that in order to support IE8 charts must not be created until the document body has been fully loaded. sample.html therefore no longer calls loadChart() inline but instead calls it via body onload.

Improved layout of x-axis eliminates gapped spacing and overlapping labels.

STXChart objects may now be set to manage their own mouse and touch events by setting STXChart.manageTouchAndMouse to true. When this is done the document level mouse and touch events can be eliminated from the main html page.

Some minor changes were made to sample\_style.css to better support the menu layouts in sample.html. measureLit and measureUnlit classes were deprecated.

STXStudies.go() now accepts an STXChart object as a second parameter. This is necessary to support translation.

stx.displayInitialized variable is now set when a chart is rendered. The old method of checking stx.chart.crossX!=null is no longer valid.

A streaming chart simulator was added to sample.html in order to better demonstrate the usage of appendMasterData().

STXChart may now be constructed with a container DIV reference. That container may be accessed at STXChart.chart.container.

References to HTML navigational componentry are not included in STXChart.chart. These include mSticky, showMeasure, annotationSave, annotationCancel and chartControls.

STXChart.registeredCharts now contains an array of all STXChart objects on the page.

Right clicking through context menu can now be overridden by replacing document.oncontextmenu. See stxChart.js

### Version 5-2013

**Important!** stxModulus.js has been added as a new file. Please be sure to include this in your application before stxKernelOs.js with the charset=”ISO-8859-1” attribute.

**Important!** The stx.computeLength() object has been modified. It no longer takes a height but instead takes two prices, high and low. This was necessary to support logarithmic charts which cannot by their nature support linear length calculations. If you've used this function in your code then please modify those cases to the new signature.

**Important!** STXMarket.nextPeriod() and STXMarket.prevPeriod() both now require an interval. This was necessary for supporting intraday charts with n-period intervals. If you've used these functions independently then please modify your code to use the new signature.

**Important!** Div objects that previously were placed outside of the chart containing object must now be positioned within the chart containing object. In sample.html this is the section under the “CHART TEMPLATES” comment. This decision was driven by the increasing complexity of positioning the objects on charts that are being used in many different ways. By placing these objects within the chart container the library has fewer calculations to make and the likelihood of bugs is decreased.

The layout of mSticky and showMeasure have changed slightly.

Also in this version div tags have been added for the crosshairs. In previous versions the crosshairs were created dynamically by the library but this now gives web developers more control over these tags. (note that the z-index for crosshairs has also been reduced from 3 to 2).

A currentHR tag has been added that displays the current price on the Y-axis in green or red depending on whether the security is currently up or down.

A button has been added to toggle crosshairs on or off.

Semi-Logarithmic scale is now supported. Set stxx.layout.semiLog=true to turn on semi-log scaling.

A new periodicity function stxx.setPeriodicityV2(period, interval) allows more flexible periodicity for intraday charts. See the new section in this documentation on periodicity. (Note that setPeriodicity() remains functionally deprecated).

The functions for floatDate and floatHR in sample.html have been modified to reflect the changes in the contained objects. If you've copied these functions into your own code be sure to modify it to reflect these changes.

The mouse and touch events section in sample.html main() function have been modified to provide better support for Windows8 touch events. The library is now compatible with “all in one” computers that use both touch and a mouse. For complete touch support please migrate to this new functionality.

A new library stxSocial.js is available. This library supports chart image creation and sharing. Please contact us if you'd like to license this package.

stx.appendMasterData() can now be used to implement auto-updating or streaming charts. See the new section in the above documentation.

STXStudies.removeStudy() can now be used to programmatically remove a study.

Several new convenience functions have been added to stxUtilities.js.

### Version 3-2013

This version remains unchanged but sample.html now includes an example of an intraday chart.

### Version 2-2013

Charting performance is greatly improved in this version thanks to smarter rendering algorithms to the HTML5 Canvas. Zooming and panning have also been improved for a better overall user experience.

This version includes support for Windows 8 touch events which makes it compatible with Windows Surface. We've also improved our support for Android devices (please note that there is a bug in the current Android browser that causes it to crash under heavy load such as can be experienced on tablets. We recommend that users use Android Chrome which does not crash and has much better performance).

This version uses a different image nomenclature for the icons. In this package you should find several icons that begin with “stx”. The icons are the same as the previous versions but this new naming system is less likely to conflict with other naming conventions.

**New features include:**

A label “currentHR” which shows the current price on the y-axis in red or green.

Colored bar charts (red if the stock is down, green if it is up)

Crosshair offset can be set programmatically and the default has changed on touch devices to be a little easier for users to see

Magnet mode allows drawing tools to snap to points on the chart

STXChart.chart.symbolDisplay allows programatic change of the display label on the charting panel

Sharp display on iPad retina displays

**New tools**

STX.Plotter – High performance drawing on the canvas

dashedLineTo - Dashed line drawing support

STX.textLabel – Convenience tool for drawing a text label on the canvas

# Native Methods

Signatures of native methods that may be accessed using prepend and append extensions. Your function will receive the same arguments and "this" will be set to the current STXChart object.

## Chart

### Return to “Home”

STXChart.prototype.home=function();

Call this method to navigate to the latest tick on the chart. The chart will instantly be repositioned with the latest tick on the far right of the screen

### Create X Axis

STXChart.prototype.createXAxis=function(){

Call this method to create the X axis (date axis). The default implementation will calculate future dates based on STXMarket.nextPeriod(), STXMarket.nextDay(), and STXMarket.nextWeek()

Those functions subsequently utilize the STXMarket.isHoliday() function. You can override the STXChart.hideDates() method to hide the dates but keep the grid lines. Use css styles stx\_xaxis and stx\_xaxis\_dark to control colors and fonts for the dates. Use css styles stx\_grid and stx\_grid\_dark to control the grid line colors. The dark styles are used when the grid changes to a major point such as the start of a new day on an intraday chart, or a new month on a daily chart.

This method sets the STXChart.chart.xaxis array which is a representation of the complete x-axis including future dates. Each array entry contains an object:

DT – The date/time displayed on the x-axis

date – yyyymmddhhmm string representation of the date

data – If the xaxis corrdinate is in the past, then a reference to the chart data element for that date

### Create Y Axis

STXChart.prototype.createYAxis=function(){

Call this method to create the Y axis (price axis). Significant logic is incorporated into this function to ensure a usable grid regardless of price granularity or magnitude. Use css style stx\_grid to manage colors of grid lines. Use stx\_yaxis to control font and color of the yaxis text (prices). Note that this.chart.roundit will control the number of decimal places displayed. This is computed automatically when loading masterData but can be overriden if desired. A roundit value of 100 will create two decimal places. 10000 will create four decimal places.

### Create Volume Chart (panel, not overlay)

STXChart.prototype.createVolumeChart=function(quotes){

This method creates a volume chart (not volume overlay) in a new panel. Volume charts are always panels called "vchart". If no volume exists then the chart will be watermarked with "Volume Not Available". Note that this.volbar() is the method that actually creates the volume bars. This method simply creates the panel and axis. quotes is the dataSegment.

### Initialize Display

STXChart.prototype.initializeDisplay=function(quotes){

This method initializes the display items for the chart. It is called with every draw() operation. The high and low values to display on the chart are calculated. Those values are subsequently used by createYAxis() which is called from within this method. This method also calls createCrosshairs(). stx.displayInitialized will be set to true after this method is called. quotes is the dataSegment.

### Update Price Label (HR) with Crosshair Values

STXChart.prototype.headsUpHR=function(){

This method computes and fills in the value of the "hr" div, which is the div that floats along the Y axis with the current price for the crosshair. This is an appropriate place to inject an append method for drawing a head's up display if desired.

### Set Chart Periodicity

STXChart.prototype.setPeriodicityV2=function(period, interval){

Call this method to change the periodicity of the chart. Period should be a numeric value. Interval should be one of "day", "week", "month" or for intraday charts an integer representing the length of a bar. Unlike most drawing operations, setPeriodicityV2() will force the creation of an entire new dataSet (this.createDataSet()). It is therefore a relatively expensive operation.

example setPeriodicityV2(1, "week"); // draw a weekly chart, assuming setMasterData has daily values

example setPeriodicityV2(7, 3); // draw a 21 minute chart, assuming setMasterData has 3 minute values

This method supercedes setPeriodicity() which has been deprecated.

### Mouse Move

STXChart.prototype.mousemove=function(e){

This method is at the heart of the interactive portion of the application (note that touchmove() uses the same internal methods and logic). this.grabbingScreen can be checked to determine whether the user is holding the mousebutton or finger down (or two fingers when in crosshair mode). this.resizingPanel can be checked to determine whether the used is holding the handle to one of the panels. this.ctrl can be used to determine whether the user is effecting a resizing gesture rather than a panning gesture. The values this.chart.left, this.chart.right, this.chart.top and this.chart.bottom can be referenced to see if the event occurs within the chart borders. this.openDialog can be referenced to determine whether a dialog is currently open (and should override most functionality in here).

Note that the default implementation of this functionality has some advanced logic involving timeouts in order to optimize display performance. The method this.headsUpHR() is called via timeouts rather than directly on mousemoves. On slow devices you may therefore see the hr div updating less frequently as events are dropped in order to maximize performance.

### Compute Consolidated Quote

STXChart.prototype.consolidatedQuote=function(quotes, position, periodicity, interval, dontRoll){

This method computes a consolidated quote for periodicities greater than a single day. It is called from within createDataSet(). Quotes is the dataSet. Position is the starting point for the consolidation. Normally, weekly and monthly charts are “rolled” (computed) from daily bar data. If you have precomputed weekly or monthly charts then pass dontRoll=true to this function.

### Run Ticks

STXChart.prototype.runTicks=function(i, quotes){

This method runs tick calculations for studies that use them (such as parabolic sar). “I” is the position within dataSegment.

### Display the Chart

STXChart.prototype.displayChart=function(quotes){

This method actually draws the chart. It calls this.candle(), this.bar(), this.drawLineChart(), this.volbar(), this.volUnderlay() and this.drawWaveChart() which are all internal functions. It also calls STXStudies.displayStudies(). Prepend or append functionality here if you need to supplement drawing operations.

### Draw

STXChart.prototype.draw=function(){

This method is the complete draw operation for a chart. It creates a new dataSegment (this.createDataSegment()), ensures that the chart hasn't been scrolled entirely off the screen by the user, draws the panels, axis, displays the chart and draws the vectos. Prepend or append ass necessary to supplement drawing operations. Draw is called continuously as a user pans or zooms a chart.

### Data Set - Create

STXChart.prototype.createDataSet=function(){

This method is used to create a new data set from the masterData. It calls the study calculateFN functions and consolidates quotes if periodicity is greater than 1 day. The results are stored in chart.dataSet (as opposed to chart.dataSegment which is called during each draw() operation and contains the data represented within the current viewport).

### Draw Current Price

STXChart.prototype.drawCurrentHR=function(dontRoll){

This method draws the floating current price. It is always the current price, not the current displayed price if the chart is scrolled backward. stx\_current\_hr\_up and stx\_current\_hr\_down are used to color the price depending on whether the most recent change is up or down from the previous tick. If a currentHR element is missing from the container then this method will do nothing. (see consolidatedQuote() for information on the dontRoll parameter).

### Data – Append Quotes (Streaming)

STXChart.prototype.appendMasterData(appendQuotes)

appendQuotes is an array of quotes in the same format as setMasterData. The kernel will append quotes to the end of the masterData. If the first quote has the same date/time as the existing final quote of masterData then it will be replaced rather than appended. Note that this function will call createDataSet() and draw() automatically.

## Panels

### Panels – Calculate Position

STXChart.prototype.adjustPanelPositions=function(){

This method calculates the appropriate positions of panels. Panel sizes are stored as a percentage of screen space. When the screen is resized, or a new panel is added or removed those dimensions must be recalculated along with the physical pixels that those dimensions represent. This method also adjusts the location of the "chartControls" div which contains the zoom buttons.

### Panel - Create

STXChart.prototype.createPanel=function(display, name, height){

This method creates a new panel. It reduces the percentage of any existing panels proportionally. If no panels exist then it allocates 20% of the existing canvas for the first panel. Optionally, height in pixels can be passed in. name is the internal name of the panel and must match the study associated with the panel. display contains the text to display in the div label associated with that panel. Internally this method will call stackPanel() and adjustPanelPositions(). Name is the name of the panel which can be used to retrieve it from the STXChart.panels object. Height is a default height for the panel. If height is not sent then the height is determined algorithmically.

At all times panel.top, panel.bottom and panel.height will represent the y coordinates for the panel. Use these members to support drawing operations.

### Panel – Stack Panels

STXChart.prototype.stackPanel=function(display, name, percent){

When building a dispay from scratch with known dimensions (such as when using a predefined view) use the stackPanel function. This method instantiates the div tabs that make up the "icons" (up, down, solo, close buttons and the label). iconsTemplate, handleTemplate and closeXTemplate must exist for this method to work. See createPanel for field descriptions. (Note: closeXTemplate is now optional as of version 09-2013. If it does not exist then the close button (X) should be included in the iconsTemplate).

### Panels - Draw

STXChart.prototype.drawPanels=function(){

This method draws the panels and repositions the associated icons. It is called by draw()

## Crosshairs

### Create Crosshairs

STXChart.prototype.createCrosshairs=function(){

This method dynamically generates the div tags that are the crosshairs. It utilizes the function createDIVBlock which automatically destroys div tags if they are recreated. If you override this function please note that it may be called over again and that destruction of any custom div tags should be handled (it is currently called with every draw operation!). Also, be sure to set onmousedown and onmouseup functions that call event.preventDefault() in order that mouse events are passed through to the chart and not held up on the div tags themselves, since by definition the crosshairs are always located underneath the mouse! this.chart.crossX and this.chart.crossY hold references to the divs.

### Set Crosshair Colors

STXChart.prototype.setCrosshairColors=function(){

When a user initiates a drawing operation the crosshairs will change color (especially useful on tablet displays). This method is where that occurs. Use the css stx\_crosshair and stx\_crosshair\_drawing to set the colors for crosshairs. Note that this.chart.crossX and this.chart.crossY contain references to the actual divs.

### Display Crosshairs

STXChart.prototype.doDisplayCrosshairs=function(){

This method is called whenever the system determines that crosshairs should be displayed (enabling crosshairs or a drawing tool and within the bounds of the chart). It simply changes the display for this.chart.crossX.style.display, this.chart.crossY.style.display and this.chart.hr.style.display. Add an append here if you need to display other elements along with crosshairs. You can check if this.chart.crossX is null in order to determine whether a chart exists (it will be null when the app is started and until a chart is first drawn). This should be contrasted with showCrosshairs() which represents a user turning crosshairs on, rather than crosshairs simply showing them because the user navigated the mouse on to the chart.

### UnDisplay Crosshairs

STXChart.prototype.undisplayCrosshairs=function(){

This is the counter function to doDisplayCrosshairs() (not to be confused with hideCrosshairs()).

### Set Measurement Display

STXChart.prototype.setMeasure=function(price1, price2, tick1, tick2, hover){

This method computes and displays the measurement pop up when a user highlights a drawing. sMeasure must be the id tag of that div. showMeasure contains the actual message. vectorTrashCan is the id of the trashcan icon itself. Note that depending on the user's device (web or touch) and the type of drawing, one, the other, or both of these divs will be displayed.

## Drawing

### Abort Current Drawing Operations

STXChart.prototype.undo=function(){

Call this method to abort the current drawing operation. By default this occurs when a tablet user double taps on the screen. You could for instance capture the "esc" key on a web system and call this function. This is also the appropriate spot to append generalized undo functionality if you wish to build a stack of drawing operations (by also intercepting createVector() )

### Create a Vector

STXChart.prototype.createVector=function(x0, x1, y0, y1, color, type, parameters, data){

This method creates a vector (drawing). x0, x1, y0, y1 are the two point locations. Color can be either a hex color or a css reference. type should be one of "segment", "line", "ray", "fib retrace", "annotation". Parameters is an optional argument that may contain additional styling information. Currently supported parameters are:

lineWidth – Numeric, set to pixel width of desired line drawing

pattern – “solid”, “dotted” or “dashed”

data is ancillary data required for the drawing, currently only containing the text of an annotation.

Inject at this function in order to build a stack of drawings outside of the system, such as to implement a generalized undo system (or use the changeCallback to capture the state of this.chart.layout)

### Draw a Vector

STXChart.prototype.plotVector=function(vector, context, highlight, temporary){

This method actually draws the vector. It is called for every drawing for each drawing operation. It is also called when drawing temporary vectors. Vector should be a vector description (such as contained in this.chart.layout). context is optional, otherwise it will be the chart's html5 canvas context. highlight can optionally be set to inform the vector that it is to be drawn in highlighted state. temporary is set to indicate that the drawing should be drawn in temporary state (as when the user is in the process of drawing a tool).

### Draw Vectors

STXChart.prototype.drawVectors=function(){

This method is called with each drawing operation to draw the vectors (drawings) on the screen. It will automatically adjust the dimensions of drawings if the user switches from adjusted to nominal prices (on daily charts). If this.chart.hiddenVectors is set to true then this method will be bypassed (so as to implement a "hide drawings" user interface function if desired).

## User Interaction Events

### Mouse Up

STXChart.prototype.mouseup=function(e){

This method initiates most of the drawing operations (similar logic is incorpoarted in touchSingleClick and touchDoubleClick for touch devices). See mousemove for details that are useful for this method.

### Mouse Down

STXChart.prototype.mousedown=function(e){

This method is called when a mouse is held down. It's primary function is to set this.grabbingScreen and initiate the grab variables that are used to calculate pan and zoom.

### Touch – Single Click

STXChart.prototype.touchSingleClick=function(finger, x, y){

This method is called whenever a touch user taps the screen once. It essentially performs the same operations as mouseup(). Finger indicates the number of fingers now touching the device.

### Touch – Double Click

STXChart.prototype.touchDoubleClick=function(finger, x, y){

This method is called whenever a touch user taps the screen twice (double click). It essentially performs the same operations as a mouse right click (delete drawing) and also performs a vertical alignment and home operation depending on the user state.

### Touch – Move

STXChart.prototype.touchmove=function(e){

This method is called whenever a user moves their finger. This event must be registered by the main html page and associated with the window, document or body. Internal calculations for the charting engine are performed based on screen coordinates rather than coordinates of the canvas itself. Single finger moves will either pan or move the crosshair. Double finger moves will pan or zoom. Three finger moves will change periodicity.

### Touch – Start

STXChart.prototype.touchstart=function(e){

This method is called whenever a user touches the screen. It must also be registered in the main html page. Internal timeout logic is used to subsequently call touchSingleClick() or touchDoubleClick()

### Touch – End

STXChart.prototype.touchend=function(e){

This method is called whenver a user untouches the screen. It must also be registered in the main html page. Internal logic calculates inertia moves (swipe) based on move length and time.

# Non Injectable Methods

## Chart

### Chart - Initialize

STXChart.prototype.initializeChart()

Initializes a chart by creating a canvas, temp canvas for drawings, and setting initial event captures

### Chart – load master data

STXChart.prototype.setMasterData(masterData)

Initializes the chart with master data (see main documentation)

### Chart – Resize Chart

STXChart.prototype.resizeChart()

Resizes the chart and panels.

### Zoom In

STXChart.prototype.zoomIn()

### Zoom Out

STXChart.prototype.zoomOut()

### Chart – Resize Canvas

STXChart.prototype.resizeCanvas()

Resizes the canvas itself to the size of the container.

### Chart – Correct if off edge

STXChart.prototype.correctIfOffEdge()

Will pull the chart back onto the screen if the user has scrolled off. Called automatically by draw()

### Create Canvas Watermark

STXChart.prototype.rawWatermark(context, x, y, text)

Creates a watermark on the canvas. Pass this.chart.context in.

### Resolve Y location

STXChart.prototype.resolveY(y)

Provides the absolute screen Y location given the relative Y location within the canvas element

### Resolve X location

STXChart.prototype.resolveX(x)

Provides the absolute screen X location given the relative X location within the canvas element

### Back out Y location

STXChart.prototype.backOutY(y)

Provides the Y location within the canvas given an absolute screen position.

### Back out X location

STXChart.prototype.backOutX(x)

Provides the X location within the canvas given an absolute screen position

### Chart – Create Dataq Segment

STXChart.prototype.createDataSegment()

Creates a new data segment for the current viewport window. This is calculated based off of this.chart.scroll and this.chart.maxTicks

### Chart – Remove overlay

STXChart.prototype.removeOverlay(name)

Removes the named overlay

### Chart – Display Sticky Message

STXChart.prototype.displaySticky(message)

Displays the text in a "sticky" div near the mouse. Set message to null to delete.

### Chart – Compute Length

STXChart.prototype.computeLength(high, low)

Returns the length in pixels for a given price differential

### Chart – Locate Tick Position

STXChart.prototype.computePosition(x, offset)

Returns the pixel location given the tick. Optionally offset left or right pixels with position or negative value.

### Chart – Compute Pixel Given a Price

STXChart.prototype.pixelFromPrice(price)

### Chart – Compute Price Given a Pixel

STXChart.prototype.priceFromPixel(y)

### Chart – Compute Pixel Given a Tick

STXChart.prototype.pixelFromTick(tick)

Tick is the absolute tick from the dataSet (not the dataSegment)

### Chart – Compute Tick Given a Pixel

STXChart.prototype.tickFromPixel(x)

Pixel may be off screen

### Chart – Return tick from date

STXChart.prototype.tickFromDate(dt)

Returns the tick given a date (date should be in text format, not a javascript Date object)

### Chart – Return date from tick

STXChart.prototype.dateFromTick(tick)

Returns the date in yyyymmddhhmm text format

### Chart – Toggle Adjusted Prices

STXChart.prototype.setAdjusted(boolean)

Use to switch between adjusted and nominal prices (assuming they have been set in masterData)

### Chart – Set Chart Type

STXChart.prototype.setChartType(value)

Sets the chart type to one of the following types: "line", "candle", "bar", "wave", “colored\_bar”, “hollow\_candle”

### Chart – Hide Dates in X Axis

STXChart.prototype.hideDates()

Override with function that returns true to hide dates in x axis

### Chart – Set Time Zone

STXChart.prototype.setTimeZone(dataZone, displayZone)

Sets the timezone of the chart, or of the data. dataZone should be a valid timezone that indicates the timezone of the originating data (such as “America/Chicago”). displayZone should be a user selected timeZone. If displayZone is null then the timezone from the browser will be used to adjust the time. If both are null then no timezone adjustments will be made.

### Chart – Compute Time Zone Value

STXChart.prototype.timeShiftDate(dt)

Returns a date/time in the user’s time zone given a date/time in the originating zone for the chart data.

### Chart – Format price

STXChart.prototype.formatPrice(price)

Formats a price for the chart’s locale

### Chart – Get default color, background color

STXChart.prototype.getDefaultColor()

Computes the default color to draw indicator lines. Call this whenever the chart container’s background color is changed. The default color is computed based on the background color of the chart. This color is stored as this.defaultColor and the background color of the container is stored as this.containerColor.

## Panels

### Create Panel Watermark

STXChart.prototype.watermark(panel, h, v, text)

Creates a watermark on the desired panel. h should be "left" or "center". v should be "top" or "bottom". stx\_watermark controls the font.

### Panel – Resize

STXChart.prototype.resizePanels()

Called internally when dragging a panel resize handle.

### Panel – Down

STXChart.prototype.panelDown(panel)

Pass in a reference to the actual panel, not the name of the panel

### Panel - Up

STXChart.prototype.panelUp(panel)

Pass in a reference to the actual panel, not the name of the panel

### Panel - Solo

STXChart.prototype.panelSolo(panel)

Pass in a reference to the actual panel, not the name of the panel

### Panel - Close

STXChart.prototype.panelClose(panel)

Pass in a reference to the actual panel, not the name of the panel

## Crosshairs

### Crosshairs - Hide

STXChart.prototype.hideCrosshairs()

Hides crosshairs but does not delete them

### Crosshairs - Show

STXChart.prototype.showCrosshairs()

Shows crosshairs when hidden

## Drawing

### Drawing – Plot a Line, Segment, or Ray

STXChart.prototype.plotLine(x0, x1, y0, y1, color, type, context, confineToChart, parameters)

Draws a segment, ray or line. Color may be a css color value or a css canvasFont() object. confineToChart will cut the drawing off at the edge of the chart itself (above the x-axis), otherwise the drawing will only be confined to the canvas and may overlap study panels.

### Drawing – Construct vector

STXChart.prototype.constructVector(x0, x1, y0, y1, color, type, data)

Construct a vector object given the values. Automatically increases "gain" so that vectors appear accurate across daily and intraday charts

### Drawing – Delete Highlighted Drawing

STXChart.prototype.deleteHighlighted()

Deletes any highlighted drawings or overlays

### Drawing – Delete Vector

STXChart.prototype.deleteVector(vector)

Deletes a drawing

### Drawing – Highlight Vector

STXChart.prototype.highlightVector(vector)

Visually highlights a vector.

### Drawing – Un-highlight Vector

STXChart.prototype.unhighlightVector(vector)

Removes the highlight from a currently-highlighted vector

### Drawing – Clear All

STXChart.prototype.clearDrawings()

Clears all drawings

## CSS / Styling

### CSS – Get CSS Color

STXChart.prototype.getCanvasColor(className)

Gets a hex color given a css class

### CSS – Get Font Size

STXChart.prototype.getCanvasFontSize(className)

Gets font size in pixels given a css class name (without "px" attached)

### CSS – Set canvas color, width, alpha

STXChart.prototype.canvasColor(className, context)

Sets color, width, globalAlpha (opacity) for the canvas given a css class name

### CSS – Set Font

STXChart.prototype.canvasFont(className, context)

Sets canvas font context given a css class name

### CSS – Get Class Style

STXChart.prototype.canvasStyle(className)

Returns an object containing the class style given a css class name (used by plotLine() for instance)