JavaScript Stock Charting + Technical Analysis Library

Version 02.2014

CONTENTS

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! Version 2.14 is a major architectural release. For users of previous versions, an especially those who have made custom integrations, 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 two 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, and user customization. Use this to quickly integrate professional quality stock charts into your website with minimal coding.

stx-quickstart.html – For developers looking 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.

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 access to End of Day or Real-Time data via an Internet API. 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. A complete API reference with each method and member of the library can be found in the second section of this manual. Separately from the 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. stx-demo.html can be used as an off-the-shelf charting solution that can be integrated into your website very quickly. This demo makes extensive use of UI components that are built into the stx library but you are in no way required to use this UI for your own implementation.

stx-quickstart.html

A minimum programmatic implementation with just a few lines of JavaScript and HTML. stx-quickstart.html displays a functional stock chart immediately.

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 dark and light "themes" that users can select in the stx-demo.html widget. You may add, remove or customize these.

STX_SAMPLE_DAILY.js, STX_SAMPLE_5min.js, STX_SAMPLE_30min.js

Sample files containing JSON examples of historical data in the format required by the charting engine. These data files are used by the examples.

stx.js

The base library that should be included first in your application. It contains several types of functions:

STXChart definition
The core object for creating and manipulating stock charts. (Programmers familiar with C can think of STXChart as the header definition and stxKernelOs.js as the object file).

Utility functions
$$() is provided as shorthand for getElementById() and $$$() for querySelector(). $$("mydiv") is the same as jquery $("#mydiv") which is the same as $$$("#mydiv"). These are provided as a convenience but you can also use jquery selectors in your code if you are working with jquery.

Market functions
Functionality related specifically to financial markets. 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.

Studies
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.

Drawings
A basic API for creating drawings with a few samples.

Internationalization
Internationalization is optionally supported using the ECMA-402 Intl standard. This standard is not widely adopted by browsers yet but is enabled with the charting library. Language translation capabilities are also included.

UI

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.

stxKernelOs.js

This file contains the core charting code. 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.

stx-sprite.png

An image "sprite" file that contain all of the images and icons necessary to implement charts.

stxTimeZoneData.js

Contains timezone definition data.

stxThirdParty.js

This file consolidates code from third party open source libraries used by the charting application. Please note that some of the code from these libraries has been modified so the actual third party libraries are no longer drop in replacements.

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.

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.

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.

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.

GETTING STARTED

Unzip the files and place your copy of stxKernelOs.js (sometimes 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 (if hosting in an iframe then the iframe domain must match). 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 fake host within your 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_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-demo.html or stx-quickstart.html for safekeeping and reference. Then use one of these files as a starting point for your application.

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) or the mousewheel may be used to zoom. 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.

Drawing

Crosshairs are automatically enabled whenever a drawing tool is active (drawing tools are activated programmatically such as enabling through a menu selection). To make a drawing in a browser, click once to begin the drawing, move, 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 "X" 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 7 minute, 3 day or 2 week

periods). For instance to display a 15 minute period you could specify 3 x 5minute 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 timeframe of the data that is sent into the charting engine (from your data server) while "period" indicates the compounding (multiplication) factor. So for instance a 12 minute chart can be created with an interval of "3 minutes" (3 minute bars from your database), which is multiplied by a period of "4".

Usage

Simply create a new STXChart object (stxx=new STXChart(divContainer)) and then call stxx.newStart(data) to load a chart. The library comes with sample data for both daily and intraday data. The same process is used to load either type of data. Simply set the interval accordingly using setPeriodicityV2() before loading calling newChart().

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");
```

STXChart will automatically expand the canvas size to the boundaries of the DOM element. The charting engine should automatically detect container size changes. (Note that the container object can be a relative style, such as width=100%. The same is not true for height since HTML5 does not include the concept of 100% height for page size. To create a full screen chart you must manually set the height for your container using JavaScript).

HTML5 canvases do not natively support CSS but for convenience STXChart has been developed so that the style of 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 in 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).

DOM componentry (HTML) are built dynamically and layered on top of the canvas and repositioned by the STXChart object. These include the icons for managing panel position, the panel labels, 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 and date labels (floatHR, floatDate). These DOM elements are loaded with mouseover events that automatically hide the crosshairs when they are in focus and also respond to touch events.

Tip:

You may eliminate the "zoom", "home" and "more" controls if you wish to provide your own navigational components by designing and including your own divs within the chart container. You may also modify the css to change the look of the existing components. Finally, if you set these components to null in the stx object before creating a chart then it will not display these components at all.

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

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.

The charting engine internally uses the STX.Plotter object which organizes canvas drawing calls so that they are batched to the GPU. If you are creating new functionality that draws on the canvas you can achieve accelerated performance by using the STX.Plotter object rather than making direct canvas calls.

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

DATA ENGINE

STXChart is initialized with the function newChart() or by calling setMasterData() directly. 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*
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.

Note: The values or Open, High, Low, Close & Volume must be integers, not strings! Use parseFloat() if your data server provides strings.

Please note: The charting engine works best with an uninterrupted (contiguous) array of datapoints. Please scrub your data so that there are no gaps before passing it into the charting engine. We can provide sample code for scrubbing data or consolidating ticks if you require. If there are gaps in your data then the x-axis will shift to cover up those gaps.

Within STXChart we use the following terminology:

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

STXChart.chart.**dataSet** – Data that has been consolidated for periodicity and includes 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.

When a new periodicity is selected (setPeriodicityV2()) a new dataSet is created. This allows indicators to automatically work with different periodicities.

Note: when loading data for intraday charts, set stx.chart.beginHour, stx.chart.beginMinute, stx.chart.endHour, stx.chart.endMinute according to the opening and closing times of your exchange. This is not necessary if the security trades 24 hours.

STREAMING

Streaming new OHLC bars

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).

stx-demo.html contains a dummy engine that simulates streaming charts. Uncomment the line "//streamSimulation()" to enable the streaming simulation.

Last Sale Streaming

If your data feed does not supply OHLC updates but only delivers "last sale" (trade) updates then use the following interface:

```
stxx.streamTrade(price, volume);
```

This mechanism will automatically calculate OHLC bars as new trades are passed in. It will automatically calculate the new bar based on the current periodicity (daily, hourly, minute, etc).

Time intervals without any trades will gap accordingly once a new trade arrives. On a candle chart these gaps will appear as dashes. Note that this mechanism precisely follows the opening and closing times of the market (stx.chart.beginHour, stx.chart.beginMinute, stx.chart.endHour, stx.chart.endMinute). Passing in trades after the close of the market will not create new ticks. If handling extended hours trading be sure to set the ending times accordingly.

EXTENSIBILITY

Customization Basics

The charting engine is essentially an animation engine. When a user scrolls or pans a chart, it is redrawn at animation speeds, providing the appearance of a smoothly moving chart in much the same way that a video game operates. STXChart.draw() is the rendering function that is called to animate the chart and so the most common method for extending chart functionality is to extend the

draw() function itself. This can be done easily with the Injection API (see below for more details on the Injection API).

There are two ways to visually extend the draw() function:

1) By drawing on the canvas
2) By displaying HTML components

For instance, as a developer you may wish to display an arrow somewhere on the screen. This could be accomplished either by physically drawing an arrow using the canvas, or by creating an HTML arrow object and superimposing it on the canvas. The choice of which technique to use depends on your use case. Drawing on the canvas takes more effort, but if you are sharing charts then this is a must because HTML elements will not render on a shared chart. However, complex graphics may be easier to create as HTML images and they can support mouse events and therefore might provide a better solution for interactive elements that you wish to place on the chart.

Your first step to customization is to append your own code to the end of the draw() function. You can do this with the following statement:

```
STXChart.prototype.append("draw", function(){
    // my custom code here
});
```

Logically, placing customized drawings or objects on the chart requires finding the appropriate positioning along the x and y axis. The charting engine provides functions that allow you to determine the physical X and Y coordinates in pixels given dates and prices.

The charting display is subdivided into panels. Each study is a panel. The chart itself is also a panel. Each panel has its own y-axis. The stxx.panels object contains all of the panels on the chart. Drawing on the chart panel is the simplest and most common task. You can get the chart panel easily:

```
STXChart.prototype.append("draw", function(){
    var panel=this.chart.panel;
});
```

Once you have access to a panel you can determine the X and Y coordinates on the canvas given a price and date (or dateTime:

```
STXChart.prototype.append("draw", function(){
    var panel=this.chart.panel;
    var dt=yyyymmddhhss(myDate);
    var x=this.pixelFromDate(dt, panel.chart);
    var y=this.pixelFromPrice(myPrice, panel);
```

```
});
```

Note that pixelFromPrice() will take a value rather than a price if you are dealing with a study panel (for instance a stochastics panel would require a value between 1 and 100).

At this point you could draw using HTML5 canvas commands:

```
STXChart.prototype.append("draw", function(){
    var panel=this.chart.panel;
    var dt=yyyymmddhhss(myDate);
    var x=this.pixelFromDate(dt, panel.chart);
    var y=this.pixelFromPrice(myPrice, panel);
    this.chart.context.fillRect(x,y,x+10,y+10);
});
```

Placement of HTML objects depends on how they are created. If objects are added to the chart container then positioning is quite simple:

```
// in your html page
var myHTMLObject=document.createElement("DIV");
myHTMLObject.style.position="absolute";
stxx.chart.container.appendChild(myHTMLObject);

STXChart.prototype.append("draw", function(){
    var panel=this.chart.panel;
    var dt=yyyymmddhhss(myDate);
    var x=this.pixelFromDate(dt, panel.chart);
    var y=this.pixelFromPrice(myPrice, panel);

    myHTMLObject.style.left=x+"px";
    myHTMLObject.style.top=y+"px";
});
```

If objects are absolutely positioned at the document level you can use resolveX() and resolveY() to convert to absolute screen coordinates

```
// in your html page
var myHTMLObject=document.createElement("DIV");
myHTMLObject.style.position="absolute";
document.body.appendChild(myHTMLObject);

STXChart.prototype.append("draw", function(){
    var panel=this.chart.panel;
    var dt=yyyymmddhhss(myDate);
    var x=this.pixelFromDate(dt, panel.chart);
```

```
    var y=this.pixelFromPrice(myPrice, panel);

    myHTMLObject.style.left=this.resolveX(x)+"px";
    myHTMLObject.style.top=this.resolveY(y)+"px";
});
```

Management of HTML objects is up to you as the developer so you'll need to build logic into your draw() extension to determine when to display or not display the objects.

You can use panel.top, panel.bottom, chart.left and chart.right to determine the boundaries of a panel.

It is also possible to work with "bars". A bar is a candle location on the screen. The number of ticks displayed on the screen is chart.maxTicks.

For instance if you wanted to draw a line between bars 5 and 10 on the screen:

```
STXChart.prototype.append("draw", function(){
    var panel=this.chart.panel;
    var x0=this.pixelFromBar(5);
    var x1=this.pixelFromBar(10);
    // draw line between x0 and x1
});
```

Note: If using transformations (such comparison charts that transform a price to a percentage) then pixelFromPriceTransform() will return a Y value based on the transformation. For instance, to get the Y location of a closing price in a comparison chart you would use pixelFromPriceTransform(someBar.Close). Whereas if you wanted to get the pixel for the Y axis location of 10% you would use pixelFromPrice(10). If no transformation is in effect then the two functions are equivalent.

Injection API: Prepend and Append

STXChart supports developer extension through an "injection API" that provides prepend and append functionality to any built-in method. Essentially what this means is that a developer can write code that will be run either before (prepend) or after (append) any internal STXChart function (such as draw() or mouseMove()). This gives developers the ability to supplement, override or ignore any of the built in functionality.

User defined prepend or append functions are called with the same arguments as the built in function. "this" will also be set to the actual STXChart object (i.e. stxx). 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.

The syntax for the injection API is as follows. In this example a developer has chosen to do something every time the chart is drawn. The code in the attached function will be called after stxx.draw(). "this" is set to the stxx object itself. Here we simply move a global element so that it is aligned with the cursor x position (this.cx).

```
STXChart.prototype.append("draw", function(){
    myDiv.style.left=this.cx+"px";
});
```

In this example the developers has chose to develop a completely custom yaxis. By returning "true" the built-in stxx.createYAxis() functionality is completely bypasses.

```
STXChart.prototype.prepend("createYAxis",function(){
    // draw my own y axis
    return true;
});
```

Note that you may prepend or append more than one function. Each injected function is stacked "outward" from the core function.

prepend >> prepend >> prepend >> function << append << append << append

Why an Injection API?

A significant problem for API developers is addressing the need for customization by each customer. Every customization request forces an API to grow in size and complexity, potentially breaking other customizations and causing a proliferation of unit tests.

Traditionally, API vendors have take one a few approaches to supporting customizations. Many vendors simple say"no"to customization that would take the API too far off course. Those who say"yes"end up either creating

complex configuration files or the API ends up with hundreds of toggles and entitlements to turn custom functionality on or off.

Another approach is to urge customers to license and modify source code. The problem with source code modification is that it creates a huge burden on the customer development team when a new release arrives. Too often source code licensees end up years behind and do not benefit from new releases.

The Injection API solves these problems by providing the same flexibility as source code modification without the risks. Each customer can create custom source code and still maintain forward compatibility with new releases.

Change Callbacks

Developers may register to receive events whenever the user adjusts the layout of a chart or whenever a drawing (vector) 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.

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.

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=true whenever one of these elements is tapped. This will bypass the tap event that would normally begin a drawing. Also call modalBegin() and modalEnd() for

mouseover and mouseout events. *Note, be sure to set the z-index appropriately otherwise objects may not reliably receive their mouse events.*

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, chart);
var charty=sts.pixelFromPrice(price, panel);
var screenx=stx.resolveX(chartx);
var screeny=stx.resolveY(charty);
domObject.style.top=screenx;
domObject.style.left=screeny;
```

Drawing

To programmatically create a drawing:

```
stxx.createDrawing("segment",{
    "pnl":"chart",
    "d0":"201209230000",
    "d1","201202230000",
    "v0": 144.90,
    "v1" 142
    });
```

Built in drawing types and their parameters are:
"segment", "line", "ray"
- "col" – color
- "pnl" – panel name
- "pattern" – solid, dashed or dotted
- "lw" – line width
- "d0" – starting date
- "d1" – ending date
- "v0" – starting value (price)
- "v1" – ending value (price)

"fibonacci" – see STXChart.currentVectorParameters in stx.js
"annotation"
- Same as above
- "text" – text to display
- "stem" – {
  - "d": - date of stem start
  - "v": - value of stem start
- }

"ellipse"
    Same as above
    "fillColor" – color to fill

"rectangle"
    Same as above
    "fillColor" – color to fill

If no color is specified when building a drawing then color will be set to "auto" and the chart will automatically display white or black depending on the background.

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 futures/forex weekend hours). Override or modify the isHoliday() and isHalfDay() 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 chart.beginHour, chart.beginMinute, chart.endHour and chart.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 (currently only Chrome has internal support for Intl). Localization is used to display prices and dates in locale specific format. *(A full locale file database is included with the Intl.js project and can be provided as a separate zip file. Please request this file if you will require localizations.)*

Translation is supported via the STXChart.translationCallback event handler. Set this to a function that accepts an English word and returns the translated word. The default implementation is to use STXI18N.translate which provides built in support for managing wordlists.

To implement localization, first download, unzip and install the locale files. Intl.js conveniently provides all locale libraries in JSONP format so supporting multiple locales can be done dynamically, and is as easy as this code:

```
<script src="Intl.js></script>
```

```
var stxx=new STXChart();
stxx.setLocale("en-US"); // You may set the locale for the chart at any time
```

stxI18N.js contains useful tools for managing multi-lingual implementations. STXI18N.wordLists should be populated with translations. The word list object can be generated using the STXI18N.missingWordList(lang) tool. This tool will parse through your UI and find all text elements. It will print out the text elements that are untranslated for the the given language. Once the word lists are created, STXI18N.translateUI(lang) can be called to automatically translate your UI to the given language.

Series

The series capability allows developers to add overlays of unrelated data sets. A series is a data set that doesn't necessarily fall on the price axis. An example of a series might be PE ratio. The Y-axis for the stock chart would have nothing to do with the PE ratio, and so PE ratio would display as an overlayed line (an "overlay" is similar to a series in that it appears on the chart, but overlays share the same price axis and are created using the Study library).

There are two steps to overlaying a series:

1) Add a data field to each quote in your masterData. For instance, your new field might be called "pe". (Be sure to call createDataSet() after making any changes to masterData)

```
for(var i=0;i<stxx.chart.masterData.length;i++){
    stxx.chart.masterData["pe"]=somePEValue
}
stxx.createDataSet();
```

2) Add a series using the addSeries() method. (call removeSeries() to remove the series).

```
var myseries=stxx.addSeries("pe", parameters);
```

addSeries will return a reference to the series. Field should be the name of your field ("pe") and parameters is an object that describes how to display the series.

```
parameters={
    color: , // color for the series line
    width: , // width in pixels for the line
    panel: , // The name of the panel in which to display the line (defaults to "chart")
    type: , // "step" or "line", defaults to "line"
    marginTop: , // Pixel margin from top of chart, or if less than 1, the percentage from
top of chart
    marginBottom: , // pixels of percentage margin from bottom of chart
}
```

The series library will accommodate continuous or sporadic data points. For instance, "pe" can be measured daily and so would be a continuous line but "dividend" would only occur on a quarterly basis (sporadic). Sporadic data points are automatically connected with continuous lines, however if type="step" is selected in parameters, then the line will be drawn horizontally between sporadic points and then step vertically to the next point, like stairsteps.

Users can delete a series the same way that they can delete an overlay, by hovering over it and right clicking (or tapping the trashcan icon on a touch

device). To prevent users from deleting series set "permanent=true" on the returning series:

```
myseries.permanent=true;
```

Sometimes it is desirable to display a series either on the top half, bottom half or centered on the screen. Positioning can be controlled using the marginTop and marginBottom parameters. Setting these to a whole number will control the margin or padding between the top or bottom of the series and the top or bottom of the chart. If you set these values to a number between 0 and 1 then the parameter will be interpreted as a percentage and automatically adjust to the size of the chart.

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 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, // true if it appears over the chart like a MA
"initializeFN": fn1, // null usually
"calculateFN": fn2,  // Your function for calculating the indicator
"seriesFN": fn3, // null if you just want to display one or more lines
"inputs": {"Period":14}, // list of inputs to display in dialog
"outputs":{"Slow":"#000000","Fast":"#0000FF"}, // outputs to display
"range":"bypass", // usually null
"nohorizontal":true // otherwise a line will display at the zero value
};
```

Each of the functions above (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 is set in the HTML menu and may be a more descriptive name.*

overlay – If true then the indicator will be overlayed on the chart (like a moving average), 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.

Note: The dataSet member should be of the form:

```
<output name> + " " + sd.name.
```

This allows the charting engine to display multiple versions of your indicator with different parameters. By following this convention, displaySeriesAsLine will work automatically. For instance when enabling RSI, this is what chart.dataSet will look like

```
console.log(stxx.chart.dataSet[0]);
"Open": 100,
"Close": 100,
"High"": 100,
"Low": 100,
"Volume": 5000,
"Result RSI (14)":45.60232
```

If you had two RSI panels open it might look like this:

```
console.log(stxx.chart.dataSet[0]);
"Open": 100,
"Close": 100,
"High"": 100,
"Low": 100,
"Volume": 5000,
"Result RSI (14)":45.60232,
"Result RSI (26)":56.12121
```

If an indicator only has one value then by default it will be named "Result" but if there are more than one output then each will be named. For instance, in the sample example you would see:

```
console.log(stxx.chart.dataSet[0]);
"Open": 100,
"Close": 100,
"High"": 100,
"Low": 100,
```

```
"Volume": 5000,
"Fast MyIndicator (14)":56.4454,
"Slow MyIndicator (14)":45.5655
```

seriesFN – SeriesFN renders all of the data points in one fell swoop, such as when drawing a moving average line across a chart. If 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 and colors 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
- 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).
- An array of strings will display a custom select box.

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 calculate function and at all times in chart.dataSet and chart.dataSegment.

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 set 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.

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

Creating Custom Indicators: Display

Example

```
STXStudies.displayMACD=function(stx, sd, quotes) {
  STXStudies.createHistogram(stx, sd, quotes, true);
  STXStudies.displaySeriesAsLine(stx, sd, quotes);
    // do any other canvas manipulation in here
}
```

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

Convenience functions are available for drawing to the chart:

stx.pixelFromBar(x)- returns the horizontal position of the tick.(Note that stx.layout.candleWidth contains the current bar width for the chart).

stx.pixelFromPrice(price, panel)- returns the vertical position for a given price (or value for a panel).

stx.plotLine(x0, x1, y0, y1, color, type)- draws a line on the screen and automatically crops to the edges of the panel display. 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.

Example custom indicator (creates dots):

```
function myCustomDisplay=function(stx, sd, quotes) {
    for(var i=0;i<quotes.length;i++){
        var quote=quotes[i];
        If(!quote) continue;
        var myValue=quote[sd.name];
        var x=stx.pixelFromBar(i);
        var y=stx.pixelFromPrice(myValue, sd.panel);
        stx.plotLine(x,x,y,y,sd.outputs[sd.name], "segment");
    }
}
```

WHAT'S NEW

Version 02-2014

To move to this release, you will need to change your include files. stxUtilities.js, stxMarkets.js, stxStudies.js, stxDrawings.js and stxStudies.js have been consolidated into a single function stx.js. Intl.js, Timezone.js and iScroll.js have been consolidated into a single function stxThirdParty.js. Please see the <script> tags within stx-demo.html.

- Chart "componentry" is now dynamically created. No need to include these html elements in your chart container unless you wish to override them.
- Added support for "series" overlays
- Added last sale streaming capability
- Drawings can now be made in panels
- Mousewheel zoom
- "Mountain" chart type
- Y-Axis labels are now native canvas and not DOM. Optionally display labels for all indicators and overlays.
- Native importLayout() and exportLayout()
- Consolidation of file structure.
- Create a new chart using newChart() function which consolidates setMasterData(), initializeChart(), createDataSet() and draw().
- Injection API now support multiple append and prepend functions.
- Increased developer control over the y-axis on charts and studies
- Built in modalBegin() and modalEnd() functions
- floatDate now a built in capability
- this.currentPanel now always equals the current panel that the cusor is within
- priceFromPixel() now takes an optional second "panel" parameter and can be used on indicator panels
- Further performance enhancements
- Less sensitive vertical scrolling (horizontal pans are now more even)
- Better pinching (pinches now occur where the user places her fingers and more consistent with the motion of the pinch)
- Automatic resizing of chart canvas even when container is resized dynamically
- Developer control over Fibonacci tool layout
- Added createDrawing() convenience function
- Undo functionality now built in using undoLast() function
- deleteAllPanels() function now available

Version 11-2013

Performance tuning. A significant effort has been put into performance tuning the application so that it can provide quicker rendering and user interface.

Support for HTML5 Canvas animation by setting STXChart.useAnimation=true. Animation is automatically enabled for Android devices.

Drawing tool overhaul. A new drawing tool architecture has been released. Building custom drawing tools is much easier. Rectangles and ellipses are now supported natively. Line width and pattern can now be selected. Annotations no longer include a "stem".

Right hand side whitespace is now automatically maintained when a user selects a chart. This provides an improved experience for end users.
A new translation API and toolkit now included in stxI18N.js. See the documentation on Internationalization.

Logarithmic charting display has been improved. Users can now pan and resize logarithmic charts without affecting the aspect ratio.

Improved pinching and zooming. The chart is now intelligent about it's placement when zooming, especially when pinching on touch devices.

Scatterplot chart now supported for datasets with more than one value per x-axis location.

getStartDate(), setStartDate(), leftTick() methods provide convenient programmatic access to chart navigation.

Vertical line tool.

tickFN functionality now deprecated. seriesFN should be used for all custom indicators in order to boost performance.

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.
Members:
currentPanel – The current panel that the cursor is within
crosshairTick – The current tick (dataSet) that the cursor is over
crosshairValue – The current yaxis location (value or price) that the cursor is over
cx – The current X position in the canvas for the cursor
cy – The current Y position in the canvas for the cursor
STXChart.crosshairX – Current X position on the page for the cursor
STXChart.crosshairY – Current Y position on the page for the cursor

Chart – Main Functions

createXAxis()

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

createYAxis()

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.

Managing Decimal Places

The Y-Axis automatically manages decimal place precision. The default behavior is to set the number of decimal places to the maximum number that is contained in the masterData. If a single entry in masterData contains 5 decimal places then the Y-Axis will always show 5 decimal places.

You may override this by setting stxx.chart.panel.yAxis.decimalPlaces equal to a hard set number of decimal places.

createVolumeChart(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.

initializeDisplay (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.

headsUpHR ()

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.

setPeriodicityV2 (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.

mousemove(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.

```
consolidatedQuote(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.

```
displayChart(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()
```

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.

```
createDataSet()
```

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).

```
drawCurrentHR(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).

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

adjustPanelPositions ()

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.

createPanel (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.

stackPanel (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).

drawPanels()

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

Crosshairs

createCrosshairs ()

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.

setCrosshairColors ()

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.

doDisplayCrosshairs ()

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.

```
undisplayCrosshairs ()
```

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

```
setMeasure (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

```
undo()
```

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.

```
drawVectors()
```

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.hideDrawings is set to true then this method will be bypassed (so as to implement a "hide drawings" user interface function if desired).

```
serializeDrawings ()
```

Returns an array of all drawings in serialized form. This array can be stored in order to rebuild the drawings at a later date using reconstructDrawings().

```
reconstructDrawings (arr)
```

Reconstucts drawings from the given array. The array should be in the format returned by serializeDrawings().

```
abortDrawings()
```

Causes all drawings to abort. Call this when you wish to cause all drawings to cease operation. Used internally when initializing a chart, such as for a new symbol.

```
clearDrawings()
```

Removes all drawings from the chart.

User Interaction Events

```
mouseup(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.

```
mousedown(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.

```
touchSingleClick(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.

```
touchDoubleClick(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.

```
touchmove(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.

```
touchstart(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()

```
touchend(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.

```
home()
```

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

---

NON INJECTABLE METHODS

Chart

Chart - Initialize

```
initializeChart()
```

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

Chart – load master data

```
setMasterData(masterData)
```

Initializes the chart with master data (see main documentation)

Chart – Resize Chart

```
resizeChart()
```

Resizes the chart and panels.

Navigation

```
zoomIn()
```

```
zoomOut()
```

```
resizeCanvas()
```

Resizes the canvas itself to the size of the container.

```
leftTick()
```

Returns the tick (location in the dataSet) of the first bar on the current screen

```
getStartDate()
```

Returns the date of the first bar on the screen

```
setStartDate(date)
```

Sets the first bar on the screen (left most bar) to the date specified (if it exists in the dataSet)

```
setRange(leftDate, rightDate, padding)
```

Automatically snaps the chart to the left and right dates. Optional "padding" (in ticks) can be used to create white space on the right side of the screen.

```
setSpan(period, interval, padding)
```

Automatcally snaps the chart to the requested length of time. Interval can be "year", "month", "day", "week", "hour", or "minute". Period should be the number of intervals (for instance "30" minutes). Padding represents an optional number of ticks to create as white space on the right hand side of the chart.

(Note that the periodicity must be set correctly using setPeriodicityV2() prior to calling this function).

Create Canvas Watermark

```
rawWatermark(context, x, y, text)
```

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

Resolving X/Y Locations

```
resolveY(y)
```

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

```
resolveX(x)
```

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

```
backOutY(y)
```

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

```
backOutX(x)
```

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

Chart – Create Data Segment

```
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

```
removeOverlay(name)
```

Removes the named overlay

Chart – Display Sticky Message

```
displaySticky(message)
```

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

Chart – Translating Pixels, Ticks, Dates and Prices

```
pixelFromPrice(price)
```

Compute the Y pixel given the price. The pixel may be off of the visible canvas.

```
priceFromPixel(y)
```

Compute the price given a Y pixel location on the canvas.

```
pixelFromTick(tick)
```

Computes a pixel given a tick. Tick is the absolute tick from the dataSet (not the dataSegment). The pixel may be off screen.

```
tickFromPixel(x)
```

Computes the tick from a given pixel. The pixel may be off screen. The tick is the position in the dataSet.

```
tickFromDate(dt)
```

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

```
dateFromTick(tick)
```

Returns the date in yyyymmddhhmm text format

```
futureTick(date)
```

Returns the tick given a date that is in the future, beyond the last entry in the dataSet. This will iterate forward, taking into account market opening dates and times.

```
pastTick(date)
```

Returns a tick given a date that occurred before the first date in the dataSet.

```
pixelFromBar(bar)
```

Returns the X pixel given a current bar, or location in the dateSegment (use pixelFromTick to translate a dataSet location.

```
pixelFromDate(date)
```

Returns a pixel given a date. Pixel may be off the screen. If the date is before or after the last entries in the dataSet, then futureTick() or pastTick() will be called to calculate the location.

```
pixelFromValue(panel, value)
```

This function is not yet fully implemented but should be used instead of pixelFromPrice() when creating new drawing tools in order to support future drawings on study panels. Returns the pixel given a panel and a value on the panel Y axis.

```
valueFromPixel(y, panel)
```

This function is not yet fully implemented but should be used instead of priceFromPixel() when creating new drawing tools in order to support future drawings on study panels. Returns the Y value given the y pixel and a panel.

```
pixelFromValueAdjusted(panel, tick, value)
```

This function is not yet fully implemented but should be used instead of priceFromPixel() when creating new drawing tools in order to support future drawings on study panels. Returns the Y value given the y pixel and a panel. If a price, then this will return a historical adjusted value given a tick. This should be used when rendering drawings so that they will display correctly both when displaying adjusted and non-adjusted prices.

Time Zones & Locale

```
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.

```
timeShiftDate(dt)
```

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

```
setLocale(locale)
```

Sets the locale for the chart. Locale should be in a format support by Intl. Examples include "en" or "en-us"

Toggle Adjusted Prices

```
setAdjusted(boolean)
```

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

Set Chart Type

```
setChartType(value)
```

Sets the chart type to one of the following types: "line", "candle", "bar", "wave", "colored_bar", "hollow_candle","scatterplot"

Format price

```
formatPrice(price)
```

Formats a price for the chart's locale

Get default color, background color

```
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.

Clipping

```
startClip(panel)
```

Sets canvas clipping for the given panel name. Any drawings made on the canvas will be clipped to that panel.

```
endClip()
```

Ends the current clipping operation.

Panels

```
whichPanel(y)
```

Returns a panel object given a y canvas location.

```
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.

```
resizePanels()
```

Called internally when dragging a panel resize handle.

```
panelDown(panel)
```

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

```
panelUp(panel)
```

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

```
panelSolo(panel)
```

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

```
panelClose(panel)
```

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

Crosshairs

```
hideCrosshairs()
```

Hides crosshairs but does not delete them

```
showCrosshairs()
```

Shows crosshairs when hidden

Drawing

```
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.

```
deleteHighlighted()
```

Deletes any highlighted drawings or overlays

CSS / Styling

```
getCanvasColor(className)
```

Gets a hex color given a css class

```
getCanvasFontSize(className)
```

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

```
canvasColor(className, context)
```

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

```
canvasFont(className, context)
```

Sets canvas font context given a css class name

```
canvasStyle(className)
```

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