ionic
INFORMATION

ShareScope

ShareScript Language Reference

9th Edition

ShareScript Language Reference

This document is aimed at reasonably experienced programmers or those who really want to understand, at a deeper level, how ShareScript works. Such an understanding will enable you to write better organised, faster code and generally get the most out of ShareScript.

You will find in this Language Reference a description of the classes, objects and functions that are defined by ShareScript. It contains four sections:

- **Section 1** describes ShareScript's **Global Functions and Objects** which are available for use throughout ShareScope and provide access to the price and fundamentals database, analytics, and access to a console (which can be useful when debugging).
- **Section 2** describes **Column Objects**, which provide an interface to ShareScript columns in ShareScope's list screens. You can add your own behaviour to a column by overriding certain functions. These objects also provide a number of methods and properties specifically for use when you are creating a custom column. These methods are not accessible elsewhere.
- **Section 3** describes **Indicator Objects**, which provide an interface to ShareScript indicators. Like Columns, these objects also provide methods and properties specifically for use with custom indicators, and these are not accessible elsewhere.
- **Section 4** describes **Chart Study Objects**, which provide an interface to ShareScript chart studies.
- **Section 5** describes the **Storage Object**, a property of both Column, Indicator and Chart Study objects. This allows you to store persistent data (e.g. user settings) in a custom column, indicator or study.

Classes, objects and functions are arranged alphabetically in each section. Some examples of use are given when they clarify an explanation, but this reference is not intended as an introduction to the ShareScript language.

Note that many of the functions can throw (catchable) exceptions and these are listed where appropriate. `TypeError` exceptions are not explicitly listed, but are thrown by any ShareScript function that receives a parameter that is not of the expected type.

Core JavaScript

ShareScript extends the JavaScript language, and this reference only documents ShareScript's additions to the language. "JavaScript, The Definitive Guide" by David Flanagan from O'Reilly Press is now in its 5th Edition and provides a complete introduction and reference to the core JavaScript language.

Part III of that book is a reference guide to JavaScript's built in functions and objects (such as the Date class and Maths functions). This document is intended to sit alongside that section (or any other similar JavaScript reference guide) to provide a complete reference for the ShareScript language[1].

[1] Note that ShareScript is based on version 1.6 of the JavaScript language. A small number of functions (which are often useful when working with arrays) were added in 1.6, and are not described in "JavaScript, The Definitive Guide" (which covers version 1.5). Information about these additional functions can be found at:
`http://developer.mozilla.org/en/New_in_JavaScript_1.6`

What's new in ShareScript v1.35?

This edition of the reference covers the new objects and functions available in the latest update to ShareScript (v1.35) released in Winter 2013.

Like previous minor releases, there are no major new features in this version of ShareScript. However there are a number of enhancements which are outlined here, and in more detail in the relevant sections of this reference.

To support the new set of advanced results added to the ShareScope database, there are new constants defined by the `Result` object. There are also some minor changes to the existing constants to normalise the treatment of EPS which should always be post-tax.

Some new list constants have been added to the `List` object to reflect recent changes and additions to ShareScope.

Some new types and parameters are now available for Risk Analysis to reflect additions to ShareScope.

The `getBarLength()` method has been added to indicators (it previously only existed in studies).

The ShareScript Library

ShareScope comes with a useful library of ShareScript functions available for use in your own scripts. The library file (SSLib.ssl) will be updated in future releases, and currently contains:

(i) A set of functions to simplify basic user input

(ii) A function to look for crosses in datasets

To use these functions, first load the library, then call the desired function. All the functions are declared in the SSLib namespace to avoid naming conflicts. e.g.

```
load("Libraries/SSlib.ssl");
var num = SSLib.intInput("Please enter a number", 0, 1000);
```

Note that the functions in the SSLib for building weekly and monthly bar data are still present, but now simply call the built-in methods `getWeeklyBarArray()` and `getMonthlyBarArray()` to obtain the data. Older scripts using these SSLib functions should now be much faster. New scripts should simply call these `Share` object methods directly, rather than using SSLib.

Global Functions and Objects Reference

Analytic Classes (single value) ShareScript v1.1

access ShareScope's built-in analytics Object→*Analytic*

Introduction

The analytics discussed here are "single value" analytics – i.e. the output of each analytic calculation is always a single value. These analytic classes are documented together since they share a common usage pattern. Note that there is a second set of analytic classes, where the calculation produces multiple output values (e.g. MACD). These have a different usage pattern and are documented later.

Constructors

`new ATR(period)`	Average True Range[*]	
`new Correlation(period)`	Correlation[**]	**ShareScript v1.31**
`new CMO(period)`	Chande Momentum Oscillator	
`new CCI(period)`	Commodity Channel Index[*]	
`new CTI(period)`	Chande Trend Index	
`new Momentum(period)`	Momentum	
`new OnBalVol()`	On Balance Volume[*]	
`new Oscillator(period)`	Oscillator	

`new PriceOsc(short, long)`
`new PriceOsc(short, long, MAtype)`
	Price Oscillator	

`new RSI(period)`
`new RSI(period, RSItype)`
	Relative Strength Index	

`new UltimateOsc(period, period, period)`	Ultimate Oscillator[*]	
`new Variance(period)`	Sample variance	
`new Volatility(period)`	Volatility	
`new VHF(period)`	Vertical Horizontal Filter	
`new Williams(period)`	Williams %R[*]	
`new WilliamsAD(period)`	Williams Acc/Dist[*]	

Call one of the above constructors to create an analytic object of the desired type. See the table below for more information about the constructor arguments.

Arguments

`period`	An integer specifying a period (or buffer) length for the analytic.
`short`	An integer specifying the short period length.
`long`	An integer specifying the long period length.
`MAtype`	Optional moving average type. You should supply a static constant defined on

[*] indicates an OHLCV analytic – see the description for details.
[**] Correlation is slightly different from the other analytics since it requires a pair of data series – see the description for usage details.

	the `MA` object (e.g. `MA.Exponential`). The default is `MA.Simple`.
`RSItype`	Optional RSI type. You should supply a static constant defined on the `RSI` object. Values are `RSI.Wilder` (default), `RSI.Exponential` or `RSI.Simple`.

Throws

`RangeError`	If an invalid period or type is passed to the constructor.

Methods

The following methods are used to get the current value of an analytic, and to add new values to the analytic. These functions are documented fully in the pages that follow.

`getNext()`	Return the new analytic value after adding new values to the buffer.
`getValue()`	Return the current analytic value.

Description

The analytic classes provide access to ShareScope's built-in analytic/indicator calculations. Together with the moving average (`MA`) class, these can be used as building blocks for your own custom analyses or indicators.

The analytics discussed here are "single value" analytics – i.e. the output of each analytic calculation is always a single value. As such, they share a common usage pattern, which is very similar to the interface provided by `MA` objects. Like `MA` objects, analytic objects maintain their own buffers of values, and allow you to feed new values in and get a new analytic value out.

The input for the `getNext()` function differs depending on the analytic. Most analytics expect a number (usually the close price). However, some analytics use OHLCV data as input for their calculations. These analytics are indicated with a * symbol to the right of their names above.

One exception to the pattern above is the `Correlation` analytic, which calculates the correlation between the changes of *two* different data-series over the required period. Unlike the other analytics, each call to `getNext()` expects two inputs – the current value of each of the two data series that you are correlating. Also note that since the calculation looks at the changes between consecutive data points, you need to input (by calling `getNext`) one more pair of data points than the period. i.e. for a 10 period correlation, you will need to pass 11 pairs of values. See the last example below for an illustration of usage.

When an analytic requires OHLCV data, you should either pass a `PriceData` or `Bar` object to `getNext()`, or alternatively you can create your own object which *must* have open, high, low, close and volume properties defined. Note, however, that only the `OnBalVol` analytic uses the volume field. See the description of the `getNext()` method for more information on its usage with the different analytic types.

Examples

The following example gets the closing price history for Lloyds TSB Plc. It then calculates and outputs the value of the RSI analytic across the whole price history:

```
var closes = getShare("LSE:LLOY").getCloseArray();
var rsi = new RSI(20, RSI.Simple);
for (var i=0; i<closes.length; i++)
	print(rsi.getNext(closes[i]));
```

Note that the input to `getNext()` can also be an array (like `MA.getNext`), so you can pass the complete price history to an indicator with a single call to `getNext()` and get back the most recent value of the indicator:

```
var prices = share.getCloseArray();
var rsi = new RSI(20);
var latest = rsi.getNext(prices);
```

This final example illustrates the use of the `Correlation` analytic, which is used slightly differently to the other analytics in that it acts on two data-series rather than a single one. We will calculate the correlation of an instrument's price changes to that of the FTSE 100 index, over the last 20 days. Note that we take care to make sure that we are passing matching (in date) data points for both series.

```
var period = 20;
var corr = new Correlation(period);
var prices = getShare("LSE:LLOY").getPriceArray(period + 1);
var ftse = getShare("UKI:UKX");
for (var i=0; i<prices.length; i++)
        corr.getNext(prices[i].close, ftse.getCloseOnDate(prices[i].dateNum));
print("Correlation of LLOY to FTSE100 is " + corr.getValue());
```

See Also

`MA`, `PriceData`, Analytic Classes (multi-value)

Analytic.getNext()

add value(s) into an analytic, getting the new analytic value out

Synopsis

`analytic``.getNext(numbers...)`	non-OHLCV indicators
`analytic``.getNext(objects...)`	OHLCV indicators

Arguments

`numbers...`	One or more numbers, or arrays of numbers. These are input to the analytic from left to right. When a parameter is an array, elements are processed from index 0 to index.length-1.
`objects...`	One or more objects that have open, high, low, close and volume properties, or arrays of these objects (normally, `PriceData` or `Bar` objects are used). These are input to the analytic from left to right. When a parameter is an array, elements are processed from index 0 to index.length-1.

Returns

A number giving the current analytic value.

Description

`getNext()` returns a number which is the current value of an analytic object, after adding one or more input values to the analytic.

When an analytic requires OHLCV data as input, you should normally pass `PriceData` object(s) to `getNext()`. A `PriceData` object represents a single OHLCV bar in an indicator. In the context of a chart study, `Bar` objects are also suitable input.

Note, however, that you can also construct your own objects, and pass these to an OHLCV analytic. These objects **must** define open, high, low, close and volume properties. However, note that in most cases, the volume can be zero, since only the `OnBalVol` indicator uses this data. Please refer to ShareScope's Help to find out how each analytic is calculated.

Example

The following two examples show the use of each form of `getNext()`. First, the example we saw earlier, calculating the RSI analytic using closing price data:

```
var closes = getShare("LSE:LLOY").getCloseArray();
var rsi = new RSI(20, RSI.Simple);
for (var i=0; i<closes.length; i++)
      print(rsi.getNext(closes[i]));
```

Now, we calculate the ATR, which is an OHLCV analytic. Note how we obtain an array of `PriceData` objects (OHLCV bars) for the share, rather than just the closing prices. Each `PriceData` record is then passed to `getNext()` to calculate the indicator:

```
var prices = getShare("LSE:LLOY").getPriceArray();
var atr = new ATR(14);
for (var i=0; i<prices.length; i++)
      print(atr.getNext(prices[i]));
```

See Also

`PriceData`, `Share.getCloseArray()`, `Share.getPriceArray()`, `Indicator.getGraph()`

Analytic.getValue()

get the current analytic value

Synopsis

analytic.`getValue()`

Returns

A number giving the current analytic value.

Description

`getValue()` returns a number which is the current value of the analytic.

Analytic Classes (multi-value) — ShareScript v1.1

access ShareScope's built-in analytics — Object→*Analytic*

Introduction

The analytics discussed here are "multi-value" analytics – i.e. each analytic calculation produces more than one output (e.g. the main value and a signal line value). Note that there is another set of analytic classes, where each calculation produces a single output value. These have a different usage pattern and are documented above.

Constructors

`new AdaptiveStochOsc(min, max, slow, sig, MAtype)`
`new AdaptiveStochOsc(min, max, slow, sig)`
	Adaptive Stochastic Oscillator*

`new ADX(period)`	ADX*
`new Aroon(period)`	Aroon*
`new MACD(short, long, sig)`	MACD
`new MinMax(period)`	Rolling Min/Max

* indicates an OHLCV analytic – see the description for details.

`new StochOsc(period, slow, sig, MAtype)`
`new StochOsc(period, slow, sig)`
	Stochastic Oscillator*

`new Trend(period)`	Rolling Trend calculation

Call one of the above constructors to create an analytic object of the desired type. See the table below for more information about the constructor arguments.

Arguments

`period`	An integer specifying a period (or buffer) length for the analytic.
`sig`	An integer specifying a signal line period.
`slow`	An integer specifying a slowing period.
`min, max`	Integers specifying the min and max periods for an adaptive stochastic oscillator.
`short, long`	Integers specifying the short and long period lengths for an MACD.
`MAtype`	Optional moving average type. You should supply a static constant defined on the `MA` object (e.g. `MA.Exponential`). The default is `MA.Simple`.

Throws

`RangeError`	If an invalid period or type is passed to the constructor.

Methods

The following method is common to all the multi-value analytics, and is used to put new data into the analytic calculation:

`next()`	Put new data into the analytic.

AdaptiveStochOsc, MACD & StochOsc Methods

`getMain()`	Get the analytic main value.
`getSignal()`	Get the signal line value.

ADX Methods

`getPDI()`	Get +ve directional indicator value.
`getNDI()`	Get -ve directional indicator value.
`getADX()`	Get average directional index.
`getADXR()`	Get average directional index rating.

Aroon Methods

`getUp()`	Get Aroon up value.
`getDown()`	Get Aroon down value.

MinMax Methods

`getMin()`	Get the minimum value in the buffer.
`getMax()`	Get the maximum value in the buffer.

Trend Methods

`getSlope()`	Get the trend line gradient.
`getValue()`	Get the trend's latest value.
`getStdDev()`	Get the standard deviation of the trend.

Description

This second set of analytic classes provides access to ShareScope's remaining built-in analytic/indicator calculations. Together with the moving average (`MA`) class, these can be used as building blocks for your own custom analyses or indicators.

The analytics listed above are "multi-value" analytics – i.e. each analytic calculation results in more than one value. Unlike `MA`s and single-value indicators, there is no `getNext()` method. Instead, you should feed a new value into the calculation using the `next()` method, then call one or more of the appropriate `get()` functions to return the desired calculation result.

As with `getNext()` for single-value analytics, the input for `next()` differs depending on the analytic. Most analytics expect a number (usually the close price). However, some analytics use OHLCV data as input for their calculations. These analytics are indicated with a * symbol to the right of their constructor descriptions above.

When an analytic requires OHLCV data, you should either pass a `PriceData` or `Bar` object to `next()`, or alternatively you can create your own object which *must* have open, high, low, close and volume properties defined. Note, however, that none of these analytics use the volume field, which can be assigned a `null` value. See the description of the `next()` method for more information on its usage with the different analytic types.

Information about all the analytics (with the exception of MinMax) and their calculations can be found in the ShareScope Help.

`MinMax` objects provide a simple building block useful in many custom indicators. A buffer size is specified to the constructor. New values can be added to the buffer, and the oldest values are automatically removed when the buffer is full. At any point, you can query the object for the current maximum and minimum values in the buffer.

Example

The following example gets the closing price history for Lloyds TSB Plc. It then calculates and outputs the values of the MACD analytic across the whole price history:

```
var closes = getShare("LSE:LLOY").getCloseArray();
var macd = new MACD(13, 26, 9);
for (var i=0; i<closes.length; i++)
{
      macd.next(closes[i]);
      print("macd = " + macd.getMain() + " signal = " + macd.getSignal() );
}
```

This next example illustrates the `MinMax` object in use. Note how `next()` can accept either a single value, or an array of values:

```
var x = new MinMax(5);
x.next([10,5,4,7,9]);
x.getMax();   // returns 10
x.next(2);    // oldest value discarded (buffer is now 5,4,7,9,2)
x.getMax();   // returns 9
```

See Also

`MA`, `PriceData`, Analytic Classes (single value)

Analytic.next()

add value(s) into an analytic

Synopsis

analytic.next(numbers...)	non-OHLCV indicators
analytic.next(objects...)	OHLCV indicators

Arguments

numbers...	One or more numbers, or arrays of numbers. These are input to the analytic from left to right. When a parameter is an array, elements are processed from index 0 to index.length-1.
objects...	One or more objects that have open, high, low, close and volume properties, or arrays of these objects (normally, PriceData or Bar objects are used). These are input to the analytic from left to right. When a parameter is an array, elements are processed from index 0 to index.length-1.

Returns

Nothing. Use the appropriate analytic get() function to get a result of the calculation.

Description

next() is the multi-value analytic equivalent of the getNext() function. See the description for Analytic.getNext() above for more information about using this function.

Analytic.get() methods

get a result from an analytic calculation

Returns

A number giving the appropriate result of the analytic calculation, or undefined if the buffer is empty.

Bar **ShareScript v1.2**

represents a chart bar in a chart study Object→Bar

Synopsis

Bar.*property*

Construction

Bar objects are made available through the bars array property of ChartStudy objects. They cannot be created using the normal JavaScript new() operator.

Properties

open	Read only	The bar’s opening price in the minor currency unit (e.g. Pence).
high	Read only	The bar’s high price in the minor currency unit (e.g. Pence).
low	Read only	The bar’s low price in the minor currency unit (e.g. Pence).
close	Read only	The bar’s closing price in the minor currency unit (e.g. Pence).

volume	Read only	The volume of shares traded in the bar.
date	Read only	The date/time (a JavaScript Date object) of the bar end point.
dateNum	Read only	An integer representation of the date (see below for details).
timeNum	Read only	An integer representation of the time (seconds since midnight).
isOHLCV	Read only	A boolean value (true or false) indicating whether the record is for a day where ShareScope has full OHLCV data. See below for further details.
isComplete	Read only	A boolean value (true or false) indicating whether the bar is complete. A non-complete (i.e. partial) bar is still accumulating data and the OHLCV values may change.
colour	Read/Write	An integer encoding the bar colour.
pen	Read/Write	A constant from the Pen object specifying the type of pen used to draw the bar (e.g. Pen.Solid).
penWidth	Read/Write	An integer giving the width of the pen. Valid values are 0 to 7. Widths >0 are only allowed for a pen type of Pen.Solid.

Description

Bar objects represent OHLCV bars on intraday and historical charts. They are made available through the built-in study object properties: bar and bars. Bar objects share much in common with PriceData objects, but have some additional write-able properties relating to the bar's visual appearance on a chart.

Because Bar objects share key properties with PriceData objects (i.e. OHLCV values), they can often be used where a PriceData object is expected (e.g. with analytics).

As with PriceData objects, the dateNum & timeNum fields provides an integer representation of the date and time. This is made available since JavaScript Date objects are relatively costly (in terms of execution speed) to use and compare.

The isOHLCV field indicates whether ShareScope has full OHLCV data for the instrument on the date of the bar. The ShareScope database has closing price only data for some instruments before a certain date – in this case the isOHLCV field will be false, and the open, high and low values will be the same as the close value. Note that when the period of a PriceData record is *longer* than one day, the isOHLCV field will always be true.

The isComplete field indicates whether the bar is a complete or partial bar. Complete bars have fixed OHLCV values, whereas partial bars are still accumulating data and the OHLCV values may change as new data arrives in the intraday feed.

See Also

ChartStudy.bars, PriceData, dateNum(), timeNum(), Colour, Pen

beep() ShareScript v1.2

make a beep sound

Synopsis

beep()

Description

beep() is a global function that causes the computer to play a short alerting sound. Your program is paused while the sound plays (about half a second).

See Also

print()

BidOfferData

ShareScript v1.1

details an intraday price change

Object→BidOfferData

Synopsis

BidOfferData.*property*

Construction

BidOfferData objects are returned in an array by the getIBidOfferArray() series of Share object methods. They cannot be created using the normal JavaScript new() operator.

Properties

bid	The bid price in the minor currency unit (e.g. Pence), or undefined if no value is available.
offer	The offer price in the minor currency unit (e.g. Pence), or undefined if no value is available.
mid	The mid price in the minor currency unit (e.g. Pence), or undefined if no value is available.
date	The date/time (a JavaScript Date object), or undefined if no value is available.
dateNum	An integer representation of the date (see below for details).
timeNum	An integer representation of the time (seconds since midnight).
index	An integer giving the index of this event within the second (see below for more information). **ShareScript v1.33**
isInAuction	A Boolean value (true or false) indicating whether this record relates to an intraday auction. If true, then the mid, bid and offer properties are all set to the indicative auction uncrossing price. **ShareScript v1.34**

Description

BidOfferData objects represent an intraday price.

The dateNum & timeNum fields provides an integer representation of the date and time. This is made available since JavaScript Date objects are relatively costly (in terms of execution speed) to use and compare.

Even though the time resolution provided by the date and timeNum fields is limited to a second, the records are always returned in the correct sequence by getIBidOfferArray(). However, if you need to determine to ordering of trades (which are returned separated by the getITradeArray() method) with respect to bid/offers, then you must use the index fields present in both arrays to determine the proper ordering of trades and prices that occur in the same second. This field starts at zero each second and increments with each trade or bid/offer occurring within that second.

See Also

`dateNum()`, `timeNum()`, `Share.getIBidOfferArray()`, `Share.getIBidOfferArrayOnDate()`, `PriceData`, `TradeData`

Colour

colour functions and constants

Synopsis

```
Colour.constant
Colour.function()
```

Constants

`Black`

`White`

`Red`	`DarkRed`	`LightRed`
`Green`	`DarkGreen`	`LightGreen`
`Yellow`	`DarkYellow`	`LightYellow`
`Blue`	`DarkBlue`	`LightBlue`
`Magenta`	`DarkMagenta`	`LightMagenta`
`Cyan`	`DarkCyan`	`LightCyan`
`Grey`	`DarkGrey`	`LightGrey`
`Gray`	`DarkGray`	`LightGray`

Static functions

`RGB()`	Create a colour from red, green and blue values.
`getRValue()`	Get the red component of a colour.
`getGValue()`	Get the green component of a colour.
`getBValue()`	Get the blue component of a colour.

Description

`Colour` is a global object that defines properties that provide functions and constants to create and examine colours (which are represented as integers). Currently, its sole use is with `Indicator` objects.

`Colour` is not a class of objects like `Date` or `MA`, and there is no `Colour()` constructor. It can be considered to be the same type of thing as the JavaScript `Math` object.

Examples

`Colour.Blue, Colour.RGB(64,0,128)`

See Also

`Indicator.setSeriesColour()`, `Indicator.getBackColour()`, `Dialog.addColPicker()`

Colour.getBValue()

get the blue component of a colour

Synopsis

`Colour.getBValue(colour)`

Arguments

`colour`	An integer encoding a colour.

Returns

An integer that is the blue component of a colour (0-255).

Colour.getGValue()

get the green component of a colour

Synopsis

`Colour.getGValue(colour)`

Arguments

`colour`	An integer encoding a colour.

Returns

An integer that is the green component of a colour (0-255).

Colour.getRValue()

get the red component of a colour

Synopsis

`Colour.getRValue(colour)`

Arguments

`colour`	An integer encoding a colour.

Returns

An integer that is the red component of a colour (0-255).

Colour.RGB()

create a custom colour from red, green and blue values

Synopsis

`Colour.RGB(red, green, blue)`

Arguments

`red`	An integer from 0 to 255 representing the amount of red.
`green`	An integer from 0 to 255 representing the amount of green.
`blue`	An integer from 0 to 255 representing the amount of blue.

Returns

An integer that encodes the specified colour.

clear()

clears the console

Synopsis

`clear()`

Description

`clear()` is a global function that clears the ShareScript console. The console window will also be shown if it is not currently visible.

See Also

`print()`

dateNum()

create a ShareScope dateNum

Synopsis

```
dateNum(dateObj)
dateNum(year, month, day)
```

Arguments

`dateObj`	A JavaScript `Date` object to be used to create the dateNum.
`year`	The year as an integer, in 4 digit format e.g. 2007.
`month`	The month as an integer, from 0 (January) to 11 (December).
`Day`	The day of the month as an integer, from 1-31.

Returns

An integer dateNum representing the date.

Throws

`RangeError`	If any of the arguments are out of range.

Description

`dateNum()` is a global function that you can use to create a dateNum value (which is just an integer that compactly represents a date). Scripts using dateNums will be faster than those using JavaScript `Date` objects. Note that the year, month and day arguments are the same as those used with JavaScript `Date` objects (i.e. the months start at 0, days start at 1).

Normally, you will not need to create dateNums yourself, but will obtain them from a `PriceData` , `BidOfferData` and `TradeData` records. You can then use one of the other dateNum functions below to inspect the date.

There is also a set of `timeNum()` functions that can be used to compactly represent the time part of a JavaScript `Date` object.

See Also

PriceData, TradeData, BidOfferData, timeNum, dateNumGetYear(), dateNumGetMonth(), dateNumGetDay(), Share.getIDateNum()

dateNumGetYear()

return the year for a ShareScope dateNum

Synopsis

dateNumGetYear(n)

Arguments

n	An integer dateNum obtained e.g. from a PriceData record or dateNum().

Returns

An integer providing the 4-digit year.

Description

dateNumGetYear() is a global function that returns the 4-digit year of a dateNum value. See dateNum() for more information about dateNums.

Note that the corresponding Date method is Date.getFullYear(). The Date.getYear() function is deprecated.

Example

```
var pd = getShare("LSE:LLOY").getPrice(); // get a PriceData record
var yr = dateNumGetYear(pd.dateNum);      // return year of the price record
```

See Also

dateNum(), dateNumGetMonth(), dateNumGetDay()

dateNumGetMonth()

return the month for a ShareScope dateNum

Synopsis

dateNumGetMonth(n)

Arguments

n	An integer dateNum obtained e.g. from a PriceData record or dateNum().

Returns

An integer providing the month (0-11).

Description

dateNumGetMonth() is a global function that returns the month of a dateNum value. See dateNum() for more information about dateNums.

To make month values easily interchangeable with those used by JavaScript Date objects, months (unlike days) are numbered from 0 (January) to 11 (December).

See Also

`dateNum()`, `dateNumGetYear()`, `dateNumGetDay()`

dateNumGetDay()

return the day in the month for a ShareScope dateNum

Synopsis

`dateNumGetDay(n)`

Arguments

`n`	A integer dateNum obtained e.g. from a `PriceData` record or `dateNum()`.

Returns

An integer providing the day (1-31).

Description

`dateNumGetDay()` is a global function that returns the day in the month of a dateNum value. See `dateNum()` for more information about dateNums.

See Also

`dateNum()`, `dateNumGetYear()`, `dateNumGetMonth()`

Dialog ShareScript v1.1

create a dialog box for user-input Object→Dialog

Constructor

```
new Dialog()
new Dialog(title, width, height)
```

The `Dialog()` constructor creates a Dialog object which represents a Windows dialog box. Dialog boxes can contain user interface elements (or controls) such as buttons and input fields to obtain input from the user. If no parameters are passed to the constructor, a basic dialog with a default width, height and caption will be created.

Arguments

`title`	Optional string. The caption to be displayed at the top of the dialog box. By default “ShareScript Dialog” is used.
`width`	Optional. The width of the dialog box (in dialog units). The default is 200.
`height`	Optional. The height of the dialog box (in dialog units). The default is 200.

Throws

`RangeError`	If the width and height specify a dialog box that is too small.

Constants

The following static constants are defined to test the return value from `Dialog.show()`:

`Ok`	The user completed the dialog by clicking on the “OK” button.

`Cancel`	The user completed the dialog by clicking on the “Cancel” button.
`User`	The lowest value reserved for user-defined buttons. See `Dialog.addButton()` for further details.

Methods

The following methods are defined on the `Dialog` object. These functions are documented fully in the pages that follow.

`addButton()`	Add a user-defined button	
`addCancelButton()`	Add a “Cancel” button.	
`addColPicker()`	Add a colour-picker control.	
`addColLinePicker()`	Add a colour and line picker control.	
`addDatePicker()`	Add a date picker control.	**ShareScript v1.32**
`addDropList()`	Add a control which presents a drop down list of values.	
`addGroupBox()`	Add a group box. This draws a box round a group of controls.	
`addHelpButton()`	Add a “Help” button to show an HTML help file.	
`addIntEdit()`	Add a text-edit control which accepts only integers (whole numbers).	
`addNumEdit()`	Add a text-edit control which accepts any number.	
`addOkButton()`	Add an “Ok” button.	
`addSharePicker()`	Add a “Find a share” control.	**ShareScript v1.34**
`addText()`	Add explanatory text to the dialog.	
`addTextEdit()`	Add a text-edit control which accepts any text.	
`addTickBox()`	Add a tick-box control (also known as a checkbox).	
`getValue()`	After showing the dialog, get the value of a control.	
`show()`	Show the dialog, and return how it completes (with Ok or Cancel).	

Description

The `Dialog` object allows you to create and present Dialog boxes to the user. The general procedure for creating a dialog box in ShareScript is as follows:

(i) create a `Dialog` object using the constructor, specifying the size and caption required.

(ii) add Ok and Cancel buttons, plus any other controls required using the set of `Dialog.add()` functions. Each control (with the exception of buttons, text, and group boxes) must be given a unique name, so the value can be retrieved later. Note that the first control added will have focus when the dialog is activated, and the TAB key order will be the same as the order that controls were added.

(iii) Call `Dialog.show()` to display the dialog to the user. The function will return when the user either closes the dialog, or clicks an Ok or Cancel button. The return value of the function will be `Dialog.Ok` if the user clicked an Ok button, `Dialog.Cancel` otherwise.

(iv) Use `Dialog.getValue()` to retrieve the value of each named control.

Dialogs may only be used at certain times by ShareScript. e.g. you *are* allowed to display a dialog when an indicator is being added to a graph, but *not* when the indicator data needs to be calculated. This prevents dialogs from being displayed inappropriately and disrupting user interaction with ShareScope. This point is discussed further below in the description of the `Dialog.show()` method.

Dialog layout can be specifed by passing x,y position, width and height for each control. You can also use –1 for these parameters to make use of automatic layout and sizing. This automatic layout is described below.

Buttons will be positioned automatically at the right hand side of the dialog box, with the first button added at the top, and subsequent buttons positioned below. Automatic width and height will give the default windows button size.

Other controls will be positioned 50 units from the left hand side of the dialog box, with the first added at the top, and subsequent controls below. Automatic width and height provide defaults that are usually appropriate for the control being added. If a specific x or y position is given for a control, any subsequent control will be automatically positioned below that control.

Group boxes do not provide automatic sizing or positioning (i.e. –1 is not a valid input). However, they modify the positioning of any subsequent control, such that the next control added with automatic positioning will be placed inside the group box, one third of the width from the left.

Like group boxes, **Text** does not provide automatic sizing or positioning. Also the addition of text to a dialog does not modify the positioning of any subsequent control.

Example

The following example creates a dialog to ask the user for the value of a “signal period” parameter. If the user clicks Ok, the value is printed to the console. Note how all the controls are automatically positioned, with labels being presented before and after the edit box.

```
var dlg = new Dialog("Enter settings", 180, 50);
dlg.addOkButton();
dlg.addCancelButton();
dlg.addIntEdit("period", -1, -1, -1, -1, "Signal Period", "days", 10);
if (dlg.show() == Dialog.Ok)
	print("User clicked okay with period = " + dlg.getValue("period"));
else
	print("User cancelled");
```

Dialog.addButton()

add a user-defined button to a dialog

Synopsis

dialog.addButton(x, y, w, h, caption, id)

Arguments

`x, y`	The x and y position of the button (in dialog units). –1 for default position.
`w, h`	The width and height of the button (in dialog units). –1 for default size.
`caption`	The label for the button (string).
`id`	An integer value which will be returned by `Dialog.show()`. Normally, this should be a value greater than or equal to the `Dialog.User` constant. See description below for more details.

Description

`addButton()` adds a user-defined button to the dialog. This is not normally necessary, and it is easier to use the `addOkButton()` and `addCancelButton()` functions instead.

When the user clicks a user-defined button, it will terminate the dialog (after checking the data is valid and within the ranges supplied). `Dialog.show()` will return the `id` value associated with the button. Normally you will wish to avoid using the `Dialog.Ok`, and `Dialog.Cancel` constants. For this reason, the `Dialog.User` constant is defined as the first value that is not used by a pre-defined button (see example below).

Example

```
dlg = new Dialog()
dlg.addButton(-1,-1,-1,-1,"Next >>", Dialog.User+0);
dlg.addButton(-1,-1,-1,-1,"<< Prev", Dialog.User+1);
if (dlg.show() == Dialog.User+0)
        ...
else
        ...
```

Dialog.addCancelButton()

add a Cancel button to a dialog

Synopsis

dialog.`addCancelButton()`
dialog.`addCancelButton(x, y, w, h, caption)`

Arguments

`x, y`	The x and y position of the button (in dialog units). –1 for default position.
`w, h`	The width and height of the button (in dialog units). –1 for default size.
`caption`	The label for the button (string). Default is "Cancel".

Description

`addCancelButton()` adds a Cancel button to the dialog. If the user clicks this button to terminate the dialog, `Dialog.show()` will return `Dialog.Cancel`. Normally you can call this method without any parameters to create a correctly labelled and positioned button. You can only add a single Cancel button to a dialog.

Dialog.addColPicker()

add a colour picker control to a dialog

Synopsis

dialog.`addColPicker(name, x, y, w, h)`
dialog.`addColPicker(name, x, y, w, h, leftLabel, rightLabel, val)`

Arguments

`name`	The control's name (string).
`x, y`	The x and y position of the control (in dialog units). –1 for default position.
`w, h`	The width and height of the control (in dialog units). –1 for default size.
`leftLabel`	Optional. A label to be placed to the left of the control (string).

`rightLabel`	Optional. A label to be placed to the right of the control (string).
`val`	Optional. The initial colour shown by the button (integer). If not specified, the default is black.

Description

`addColPicker()` adds a colour picker control to the dialog. The name of the control should uniquely identify the control, so the value can be retrieved by `Dialog.getValue()`.

Example

This shows a colour picker being added to a dialog. The colour picker is initialised to the value of the variable `lineColour` (red):

```
var lineColour = Colour.Red;
var dlg = new Dialog("example",150,30);
dlg.addOkButton();
dlg.addColPicker("col1", -1, -1, -1, -1, "Line Colour", "", lineColour);
dlg.show();
lineColour = dlg.getValue("col1");
```

See Also

`Colour`, `Dialog.addColLinePicker()`, `Indicator.setSeriesColour()`

Dialog.addColLinePicker()

add a colour and line picker control to a dialog

Synopsis

```
dialog.addColLinePicker(name, x, y, w, h)
dialog.addColLinePicker(name, x, y, w, h, left, right, valCol, valStyle, valWidth)
```

Arguments

`name`	The control's name (string).
`x, y`	The x and y position of the control (in dialog units). –1 for default position.
`w, h`	The width and height of the control (in dialog units). –1 for default size.
`left`	Optional. A label to be placed to the left of the control (string).
`right`	Optional. A label to be placed to the right of the control (string).
`valCol`	Optional. The initial colour shown by the button (integer). If not specified, the default is black.
`valStyle`	Optional. The initial pen style shown by the button (integer). If not specified, the style will be `Pen.Solid`.
`valWidth`	Optional. The initial pen width shown by the button (integer). Valid values are 0 to 7. If not specified this defaults to 0 (the thinnest line). Greater widths are only allowed for a pen style of `Pen.Solid`.

Description

`addColPicker()` adds a colour and line style picker control to the dialog. The name of the control should uniquely identify the control, so the value can be retrieved by `Dialog.getValue()`.

This control is unique in that `Dialog.getValue()` returns an object, rather than a simple value. The object has several named fields to allow the independent retrieval of the line colour, pen style and pen width. This is shown in the example below.

Example

This shows a colour and line picker being added to a dialog. Note how the different attributes of the control are referenced following calls to `Dialog.getValue()`.

```
var dlg = new Dialog("example",150,30);
dlg.addOkButton();
dlg.addColLinePicker("line1", -1, -1, -1, -1, "Signal Line");
dlg.show();
colour = dlg.getValue("line1").colour;
pen = dlg.getValue("line1").pen;
width = dlg.getValue("line1").width;
```

See Also

`Colour`, `Pen`, `Dialog.addColPicker()`, `Indicator.setSeriesLineStyle()`

Dialog.addDatePicker() ShareScript v1.32

add a date picker control to a dialog

Synopsis

```
dialog.addDatePicker(name, x, y, w, h)
dialog.addDatePicker(name, x, y, w, h, leftLabel, rightLabel, val)
```

Arguments

`name`	The control's name (string).
`x, y`	The x and y position of the control (in dialog units). –1 for default position.
`w, h`	The width and height of the control (in dialog units). –1 for default size.
`leftLabel`	Optional. A label to be placed to the left of the control (string).
`rightLabel`	Optional. A label to be placed to the right of the control (string).
`val`	Optional. The initial date to be shown (JavaScript `Date` object). If not specified, the default is today's date.

Description

`addDatePicker()` adds a date picker control to the dialog. The name of the control should uniquely identify the control, so the value can be retrieved by `Dialog.getValue()`. Because the date picker has several on screen components, you should generally use the default width and height of the control.

Example

This shows a date picker being added to a dialog. The date picker is initialised to the the first of January 2010. The returned date is converted to a dateNum (e.g. for use with the storage area).

```
var dlg = new Dialog("example",200,30);
dlg.addOkButton();
dlg.addDatePicker("date1", -1, -1, -1, -1, "Start Date", "", new Date(2010,0,1));
dlg.show();
var startDate = dateNum(dlg.getValue("date1"));
```

See Also

`dateNum`

Dialog.addDropList()

add a drop-down list control to a dialog

Synopsis

```
dialog.addDropList(name, x, y, w, h, items)
dialog.addDropList(name, x, y, w, h, items, leftLabel, rightLabel, val)
```

Arguments

`name`	The control’s name (string).
`x, y`	The x and y position of the control (in dialog units). –1 for default position.
`w, h`	The width and height of the control (in dialog units). –1 for default size.
`items`	An array of strings to be presented as options.
`leftLabel`	Optional. A label to be placed to the left of the control (string).
`rightLabel`	Optional. A label to be placed to the right of the control (string).
`val`	Optional. An integer specifying the initial item selected. These are counted from 0. (If not specified, this defaults to 0 i.e. the first item).

Description

`addDropList()` adds a drop-down list control to the dialog. The name of the control should uniquely identify the control, so the value can be retrieved by `Dialog.getValue()`.

Example

This shows the drop-down list control being added to a dialog.

```
var dlg = new Dialog("example",200,30);
dlg.addOkButton();
dlg.addDropList("list1", -1,-1,-1,-1, ["Apple", "Orange"], "Pick a fruit");
dlg.show();
```

Dialog.addGroupBox()

add a group box to a dialog

Synopsis

```
dialog.addGroupBox(x, y, w, h)
dialog.addGroupBox(x, y, w, h, caption)
```

Arguments

`x, y`	The x and y position of the control (in dialog units).
`w, h`	The width and height of the control (in dialog units).
`caption`	The text label (string) displayed at the top of the box.

Description

`addGroupBox()` adds a group box to the dialog. This special dialog element can be used to visually organise or group controls with related functions.

Unlike a normal control, it does not take a name as a parameter since the user cannot interact with this control, and hence it has no value to return through `Dialog.getValue()`.

Adding a group box to your dialog will modify the current "cursor" location for automatic placement of controls. After a call to `addGroupBox()`, the cursor will be placed just inside the top margin of the group box, and about a third of the way in from the left. Any subsequent controls added with an x, y position of (-1,-1) will appear inside the group box.

Dialog.addHelpButton()

add a Help button to a dialog

Synopsis

dialog.addHelpButton(file)
dialog.addCancelButton(file, x, y, w, h, caption)

Arguments

file	The path to an HTML help file. The path must be relative to the ShareScript directory and specify a file with an ".html" extension.
x, y	The x and y position of the button (in dialog units). –1 for default position.
w, h	The width and height of the button (in dialog units). –1 for default size.
caption	The label for the button (string). Default is "Help".

Description

`addHelpButton()` adds a Help button to the dialog. Unlike other buttons, a Help button does not terminate the dialog, but will instead show a specified HTML file in the user's web-browser. Normally you can call this method without any parameters to create a correctly labelled and positioned button. You can only add a single Help button to a dialog.

Where possible, you should use the same file name for both your column/indicator script and the associated help file. You should also use the `getScriptPath()` function to build the full filename, so your script can continue to work if a user moves the script and associated help file to a subdirectory.

Examples

The first example below uses an explicit path to the help file. However, this will stop working if the user organises their scripts by moving the script and help file to a subfolder. The second example will continue to work.

```
dlg.addHelpButton("Columns/MyHelp.html");
```

```
dlg.addHelpButton(getScriptPath() + "MyHelp.html");
```

See Also

getScriptPath()

Dialog.addIntEdit()/Dialog.addNumEdit()

add an edit box control for numerical input to a dialog

Synopsis

dialog.addIntEdit(name, x, y, w, h)
dialog.addIntEdit(name, x, y, w, h, leftLabel, rightLabel, val, min, max)

```
dialog.addNumEdit(name, x, y, w, h)
dialog.addNumEdit(name, x, y, w, h, leftLabel, rightLabel, val, min, max)
```

Arguments

`name`	The control's name (string).
`x, y`	The x and y position of the control (in dialog units). –1 for default position.
`w, h`	The width and height of the control (in dialog units). –1 for default size.
`leftLabel`	Optional. A label to be placed to the left of the control (string).
`rightLabel`	Optional. A label to be placed to the right of the control (string).
`val`	Optional. A number specifying the initial value of the edit box. (If not specified, this defaults to 0).
`min`	Optional. A number specifying the minimum acceptable value of the edit box. (If not specified, this defaults to 0).
`max`	Optional. A number specifying the maximum acceptable value of the edit box. (If not specified, this defaults to 1000).

Description

`addIntEdit()` and `addNumEdit()` add edit box controls to the dialog allowing the user to specify a numerical parameter. Use `addIntEdit()` to add a control that will only accept integers (whole numbers). To allow any number to be accepted use `addNumEdit()`. The name of the control should uniquely identify the control, so the value can be retrieved by `Dialog.getValue()`.

Note that the minimum and maximum values are only enforced when the user clicks the Ok button (or a user-defined button). If the user clicks Cancel or closes the dialog, the values obtained by `Dialog.getValue()` will be the last value of the control, which could be outside the range specified.

Example

This shows an integer only edit box control being added to a dialog. The edit box defaults to a value of 7, with minimum and maximum values of 2 and 14.

```
var dlg = new Dialog("example",180,30);
dlg.addIntEdit("period", -1,-1,-1,-1, "Signal Period", "days", period, 2, 14);
```

See Also

`Dialog.addTextEdit()`

Dialog.addOkButton()

add an Ok button to a dialog

Synopsis

```
dialog.addOkButton()
dialog.addOkButton(x, y, w, h, caption)
```

Arguments

`x, y`	The x and y position of the button (in dialog units). –1 for default position.
`w, h`	The width and height of the button (in dialog units). –1 for default size.
`caption`	The label for the button (string). Default is "Ok".

Description

addOkButton() adds an Ok button to the dialog. If the user clicks this button to terminate the dialog, Dialog.show() will return Dialog.Ok. Normally you can call this method without any parameters to create a correctly labelled and positioned button. You can only add a single Ok button to a dialog.

When the Ok button is clicked, the dialog will alert the user if any control values are outside the range allowed. Input focus is transferred to the offending value, and the user has the chance to correct the mistake.

Dialog.addSharePicker()

add a “Find a share” control to a dialog

Synopsis

```
dialog.addSharePicker(name, x, y, w, h)
dialog.addSharePicker(name, x, y, w, h, leftLabel, rightLabel, val)
```

Arguments

name	The control’s name (string).
x, y	The x and y position of the control (in dialog units). –1 for default position.
w, h	The width and height of the control (in dialog units). –1 for default size.
leftLabel	Optional. A label to be placed to the left of the control (string).
rightLabel	Optional. A label to be placed to the right of the control (string).
val	Optional. The initial instrument shown by the button (Share object). If not specified, the default is the FTSE100 index (UKI:UKX).

Description

addSharePicker() adds a share picker control to the dialog. This is a button that can be clicked to invoke the “Find a share” dialog. The button displays the exchange and epic of the currently chosen share. The name of the control should uniquely identify the control, so the value can be retrieved by Dialog.getValue().

The “Find a share” dialog defaults to showing the same instrument list (e.g. the ALL list) that was selected the previous time the find dialog was invoked. However the user can select any list from the dropdown menu. If you want to restrict the type of instrument that can be selected, then you can check the instrument after the dialog is dismissed, and redisplay the dialog if necessary (see the example below).

Example

This shows a share picker being added to a dialog. The picker defaults to the FTSE All-share index. The dialog is shown again if an index isn’t selected (unless the user cancels the dialog).

```
var benchmark = getShare(“UKI:ASX”);
var dlg, ret;
do {
      dlg = new Dialog("Select an index",180,50);
      dlg.addOkButton();
      dlg.addCancelButton();
      dlg.addSharePicker("bench", -1, -1, -1, -1, "Benchmark:", "", benchmark);
      ret = dlg.show();
      benchmark = dlg.getValue("bench");
}
while (ret == Dialog.Ok && benchmark.getType() != “Index”)
```

Dialog.addText()

add text to the dialog

Synopsis

dialog.addText(x, y, w, h, text)

Arguments

x, y	The x and y position of the control (in dialog units).
w, h	The width and height of the control (in dialog units).
text	The text (string) to be displayed inside the defined rectangle.

Description

addText() allows you to add text anywhere on a dialog. You can use this to give help or instructions to the user.

Unlike a normal control, it does not take a name as a parameter since the user cannot interact with this control, and hence it has no value to return through Dialog.getValue().

Text is displayed left-justified, and will automatically be laid out across multiple lines (with word breaks). Any text that would extend beyond the bottom of the rectangle is not displayed.

There is normally no need to use this method to label individual controls, since each control can have its own left and right hand side labels.

Dialog.addTextEdit()

add an edit box control to a dialog

Synopsis

dialog.addTextEdit(name, x, y, w, h)
dialog.addTextEdit(name, x, y, w, h, leftLabel, rightLabel, val)

Arguments

name	The control's name (string).
x, y	The x and y position of the control (in dialog units). –1 for default position.
w, h	The width and height of the control (in dialog units). –1 for default size.
leftLabel	Optional. A label to be placed to the left of the control (string).
rightLabel	Optional. A label to be placed to the right of the control (string).
val	Optional. A string specifying the initial contents of the box. (If not specified, the edit box will be empty).

Description

addTextEdit() adds an edit box to the dialog. This edit box control can accept any input, with a maximum length of 512 characters (this limit was 80 in earlier versions of ShareScript). The name of the control should uniquely identify the control, so the value can be retrieved by Dialog.getValue().

See Also

Dialog.addIntEdit(), Dialog.addNumEdit()

Dialog.addTickBox()

add a tick-box control to a dialog

Synopsis

dialog.addTickBox(name, x, y, w, h)
dialog.addTickBox(name, x, y, w, h, text, val)

Arguments

name	The control's name (string).
x, y	The x and y position of the control (in dialog units). –1 for default position.
w, h	The width and height of the control including the text (in dialog units). –1 for default size.
text	The text displayed to the right of the control (string).
val	Optional. A boolean value specifying the initial state of the box. (If not specified, this defaults to false i.e. not ticked).

Description

addTickBox() adds a tick-box control to the dialog. This allows you to ask the user to make an on/off decision. The name of the control should uniquely identify the control, so the value can be retrieved by Dialog.getValue().

Note that unlike many of the other controls, the tick-box does not have separate left and right labels. Instead, the text (displayed to the right) is considered an integral part of this control, and is included within the control's width.

Dialog.getValue()

get a control's value from the dialog

Synopsis

dialog.getValue(name)

Arguments

name	The control's name (string).

Returns

The value of the control. The returned type varies by control (see below for details).

Throws

RangeError	If an invalid field is requested.

Description

getValue() can be called after Dialog.show() to retrieve the user-specified value of a control. Note that values are guaranteed to be within their specified ranges when Dialog.show() returns any value except Dialog.Cancel.

The return type of getValue() depends on the type of control for which a value is requested. The table below shows the return type for each control:

Control	Return type for getValue()

Tick-box	Boolean. `true` = ticked, `false` = unticked
Drop-list	Number. 0 is the first item, 1 the second, etc.
Numeric edit boxes	Number.
Text edit box	String.
Colour picker	Number.
Colour/Line picker	Object. Has colour, pen & width fields (all Numbers).
Date picker	A Javascript `Date` object.
Share picker	A `Share` object.

Dialog.show()

presents the dialog to the user and blocks until the dialog is dismissed

Synopsis

dialog.`show()`

Returns

A number indicating whether the user clicked Ok or Cancel (or a user-defined button).

Throws

`Error`	If ShareScope blocks presentation of the dialog.

Description

`show()` will display the dialog to the user, and will not return until the user has finished interacting with the dialog. The return value can be compared to the static constants `Dialog.Ok`, `Dialog.Cancel` and `Dialog.User`.

The call to `show()` may throw an exception if ShareScope blocks presentation of the dialog. ShareScope will do this to prevent dialogs being shown at inappropriate times.

Dialogs may be shown from scripts run from the console, and usually from the `init()` method of a ShareScript column or indicator (when the `init()` status is `Adding` or `Editing`). You cannot show a dialog when the `getVal()` method is called on a column, nor when the `getGraph()` method is called on an indicator.

File — ShareScript v1.1

allows reading and writing of text files to disk — Object→File

Constructor

```
new File()
new File(filename)
new File(filename, mode)
```

With no arguments, the `File()` constructor creates a `File` object that you can use to open a file on disk for reading or writing. Once you have created the `File` object in this way, you can then use its `open()` method to associate it with a particular file.

Alternatively, you can pass a filename (and, optionally, a mode) to the constructor to construct the `File` object and open a file in one operation. If only the filename is specified, the file will be opened for reading only.

Arguments

`filename`	Optional. The name of the file to open.
`mode`	Optional. The mode used to open the file. This should be one of the static constants defined on the File object (e.g. `File.ReadMode`). See below for the possible values. If no `mode` is given, the file will be opened for reading only.

Throws

`RangeError`	If an invalid filename or mode was specified.

Constants

The following static constants can be used to specify the mode to open the file.

`ReadMode`	Open the file for reading only. It must exist.
`WriteMode`	Open the file for writing. If the file does not exist, it will be created. An existing file (if present) will be overwritten.
`AppendMode`	Any output will be appended to an existing file (if present). If the file does not exist, it will be created.

Methods

The following methods are defined on the `File` object. These functions are documented fully in the pages that follow.

`open()`	Opens a file.
`close()`	Closes a file.
`readLine()`	Read a line of text from the file.
`writeLine()`	Write a line of text to the file.

Description

The `File` object provides access to reading and writing text files. A `File` object can be associated with a particular file on the disk by calling the `open()` method, or by passing a filename to the constructor. You should call `close()` when you have finished reading from or writing to the file. You can then reuse the `File` object to open another file if you wish. Note that if you forget to close a file, this will be done for you when the `File` object is garbage collected.

You can read files from anywhere in the `ShareScript` directory. However, **you can only write to the `ShareScript/Output` directory**.

If the filename does not begin with a slash (e.g. "test.csv") it will be read/written relative to the `ShareScript/Output` directory. If the filename does begin with a slash (e.g. "/Columns/test.csv") it will be read/written relative to the `ShareScript` directory. You can use either forward (/) or back (\) slash characters to separate elements of the path. However, backslashes need to be escaped in strings ("\\").

If you wish to read a file (e.g. a CSV file) that sits in the same directory as your script, you should use `getScriptPath()` to make sure the script can continue to find the file, even if the user moves the script and associatied files to subfolder (see last example below).

Examples

Creates a file called "test.txt" (in the ShareScript Output folder) and writes some text to it:

```
var f = new File();
```

```
f.open("test.txt", File.WriteMode);
f.writeLine("Hello world.");
f.close();
```

This second example reads the first line of the file, this time showing the alternative method where we open the file direct from the constructor:

```
var f = new File("test.txt", File.ReadMode);
var str = f.readLine();
f.close();
```

This final example shows how to open a CSV data file that is in the same directory as the script itself:

```
var f = new File(getScriptPath() + "data.csv", File.ReadMode);
```

See Also

`getScriptPath()`

File.open()

open a file for reading, writing or appending

Synopsis

file`.open(filename)`
file`.open(filename, mode)`

Arguments

`filename`	The name of the file to open.
`mode`	Optional. The mode used to open the file. This should be one of the static constants defined on the File object (e.g. `File.ReadMode`). See above for the possible values. If no `mode` is given, the file will be opened for reading only.

Throws

`Error`	If the `File` object already has an open file.
`RangeError`	If an invalid filename or mode was specified.

Description

`open()` associates the `File` object with a particular file on disk, and opens the file for reading, writing or appending (depending on the mode specified). The file must be closed before `open()` can be called again.

See Also

`getScriptPath()`

File.close()

closes the file

Synopsis

file`.close()`

Throws

`Error`	If the `File` object has no open file.

Description

Use `close()` when you have finished reading from or writing to a file.

File.readLine()

read a complete line of text from the file

Synopsis

`file``.readLine()`

Returns

A string with the next complete line read from the file (with CR/LF characters removed). The function will return `undefined` if the end of the file has been reached.

Throws

`Error`	If the `File` object has no open file, or if there was an error reading from the file.

Description

`readLine()` reads a line of text from a file, starting at the beginning (when the file has just been opened). Each call to `readLine()` will return the next complete line of the file until the end of the file is reached, at which point `undefined` will be returned.

A line is defined as being zero or more characters terminated by a line-feed. Note that the maximum length of a line is 1024 characters.

Example

This example prints out each line of a file called “test.txt”:

```
var f = new File("test.txt");       // default mode is File.ModeRead
while (str = f.readLine())
{
      print(str);
}
f.close()
```

File.writeLine()

write a line of text to the file

Synopsis

`file``.writeLine()`
`file``.writeLine(s)`

Arguments

`s`	An (optional) string to write to the file.

Throws

`Error`	If the `File` object has no open file, or if there was an error writing to the file.

Description

`writeLine()` writes the specified text to the file. The file must have been opened in write or append mode. A line-feed character is automatically appended to the text. If no argument is passed to `writeLine()` it will add an empty line to the file.

getList()

get an array of Share objects corresponding to one of ShareScope's built-in lists

Synopsis

```
getList(listID)
```

Arguments

`listID`	A value specifying the list. Possible values are defined as static constants by the `List` object.

Returns

An array of `Share` objects that belong to the requested list.

Throws

`RangeError`	If an invalid `listID` was specified.

Description

`getList()` is a global function that returns an array of `Share` objects corresponding to one of ShareScope's built-in lists (e.g. the FTSE 100 list). The possible lists that can be requested are defined as static constants by the `List` object.

Example

```
var ftse100 = getList(List.FTSE100);
```

See Also

`List`, `getShare()`, `Share.getAssocShares()`, `Share.getSectorIndex()`, `getPortfolio()`

getPortfolio()

get an array of Share objects corresponding to a portfolio

Synopsis

```
getPortfolio(s)
```

Arguments

`s`	A string specifying the name of a user-portfolio.

Returns

An array of `Share` objects that belong to the requested portfolio.

Throws

`RangeError`	If the portfolio does not exist.

Description

`getPortfolio()` is a global function that returns an array of `Share` objects corresponding to a ShareScope user-portfolio.

See Also

`getShare()`, `Share.getAssocShares()`, `Share.getSectorIndex()`, `getList()`

getPortfolioNames() ShareScript v1.1

returns the names of ShareScope's user portfolios

Synopsis

```
getPortfolioNames()
getPortfolioNames(groups)
```

Arguments

`groups`	Optional boolean value. Specify `true` to include group portfolios (default) or `false` to exclude them.

Returns

An array of strings. Each element is the name of a ShareScope user portfolio.

Description

`getPortfolioNames()` is a global function that returns an array of strings providing the names of ShareScope's user portfolios. A portfolio name returned by this function can be passed to `getPortfolio()` to get the list of `Share` objects in the portfolio.

See Also

`getPortfolio()`

getScriptPath() ShareScript v1.3

returns the path of the calling script

Synopsis

```
getScriptPath()
getScriptPath(includeFilename)
```

Arguments

`includeFilename`	Optional boolean value. Specify `true` to include the filename of the script, `false` to exclude it (the default).

Returns

A string with the path of the calling script, relative to the ShareScript directory.

Description

`getScriptPath()` is a global function that returns the path of the script it was called from. The filename of the script is not included by default, and the path ends in a trailing slash. The filename is the script is appended if you pass `true` to the function.

`getScriptPath()` will return `undefined` if not called from a script (e.g. from the console).

Example

If called from a column script called `MyCol.ss`, you would get the following results from `getScriptPath()`:

`getScriptPath()` would return "/Columns/"

`getScriptPath(true)` would return "/Columns/MyCol.ss"

getShare()

get the Share object corresponding to a specified instrument

Synopsis

```
getShare(s)
getShare(shareScopeID)                                    ShareScript v1.1
getShare(shareScopeID, shareNum)                          ShareScript v1.1
```

Arguments

`s`	A string specifying the exchange and epic of the instrument. This should be of the form EXCHANGE:EPIC. If EXCHANGE is not supplied, LSE is assumed.
`shareScopeID`	The ShareScope ID for the company.
`ShareNum`	Optional share number. If not specified, the primary share is returned.

Returns

A `Share` object corresponding to the specified instrument.

Throws

`RangeError`	If the specified instrument cannot be identified.

Description

`getShare()` is a global function that returns a reference to a `Share` object. This share object can then be queried by calling one of its methods e.g. `Share.getClose()`.

Examples

```
var lloy = getShare("LSE:LLOY");
lloy.getClose();     // returns the latest close for Lloyds TSB Plc

var hbos = getShare(3428);
hbos.getName();      // returns "HBOS PLC"
```

See Also

`Share.getSectorIndex()`, `getList()`, `getPortfolio()`, `Share.getAssocShares()`, `Share.getShareScopeID()`, `Share.getShareNum()`

getSSAccountNumber() ShareScript v1.2

returns the ShareScope Account Number

Synopsis

```
getSSAccountNumber()
```

Returns

The account number as an integer

Description

`getSSAccountNumber()` is a global function that returns the user's ShareScope account number.

List

constants for access to ShareScope's built-in lists

Synopsis

`List.`*`constant`*

Constants

`All`
`Shares`
`InvestmentTrusts`
`Indices`
`FTSE100`
`FTSE250`
`FTSESmallCap`
`FTSEFledgling`
`LSENonIndex`
`AIM`
`Warrant`
`Preference`
`Convertibles`
`Income`
`Capital`
`Other`
`Imports`
`FTSE350`
`UnitTrusts`
`TechMARK` (`TechMark` – deprecated)
`FT30`
`TechMARKFocus` (`TechMark100` – deprecated)
`Gilts`
`Commodities`
`FX`

`FTActuaries`	**ShareScript v1.3**
`Bonds`	
`ETFS`	
`LSE`	
`NASDAQ`	
`NYSE`	
`NYSEMKT` (`AMEX` – deprecated)	
`Europe`	
`US`	
`UK`	
`LSEShares`	
`FTSE350SectorIndices`	**ShareScript v1.1**
`FTSEAllShare`	**ShareScript v1.1**
`CoveredWarrants`	**ShareScript v1.1**
`LSEFullyListed`	**ShareScript v1.2**
`IndexFutures`	**ShareScript v1.2**
`PlusMarkets`	**ShareScript v1.3**
`NASDAQ100`	**ShareScript v1.33**
`DJ30`	**ShareScript v1.33**
`USShares`	**ShareScript v1.33**
`AIM100`	**ShareScript v1.35**
`AIMUK50`	**ShareScript v1.35**
`AIMAllShare`	**ShareScript v1.35**
`IMASectorIndices`	**ShareScript v1.35**

Description

List is a global object that defines constants that refer to ShareScope's built-in lists. These can be used with the global `getList()` function as follows –

```
var ftse100 = getList(List.FTSE100);
```

List is not a class of objects like `Date` or `MA`, and there is no `List()` constructor. It can be considered to be the same type of thing as the JavaScript `Math` object.

See Also

`getList()`

load()

load and execute a ShareScript file

Synopsis

`load(s)`

Arguments

`s`	A string specifying the filename to load and execute, relative to the ShareScript directory.

Description

`load()` is a global function that loads and executes a ShareScript file. Any variables and functions contained in the file will become defined in the global object. `load()` is also a property of the Column and Indicator objects, allowing you to load and define library functions in specific objects, rather than globally.

Any ShareScript library files should be named with an ".ssl", rather than a ".ss" file extension. Future versions of ShareScope may require this naming convention.

Example

```
load("libraries/myLib.ssl");
```

See Also

`Column`, `Indicator`, `getScriptPath()`

MA

provides access to a range of moving average calculations Object→MA

Constructor

```
new MA(period)
new MA(period, type)
new MA(period, type, values...)
```

Creates a moving average object with the `period` and `type` specified. If no `type` is specified, a simple moving average is created.

You can fill the moving average buffer by passing one or more numbers (or arrays of numbers) as the third parameter. If less values are specified than the MA period, then the first value will be repeated to fill the start of the buffer. If no values are specified then the buffer is empty and will be filled by the first call to `getNext()` with the same rules.

`MA()` may also be called as a function, without the `new` operator. When invoked in this way it will return an average of the values passed as the third parameter.

Arguments

`period`	An integer specifying the period.

`type`	Optional. The type of moving average to create. This should be one of the static constants defined on the MA object (e.g. `MA.Exponential`). See below for the possible values. If no `type` is given, a simple moving average will be created.
`values...`	Optional. One or more numbers, or arrays of numbers. These are added to the moving average from left to right. When a parameter is an array, elements are processed from index 0 to index.length-1. When `MA` is used as a function (see below), if no values are specified the function will return `undefined`.

Throws

`RangeError`	If an invalid MA period or type is specified.

Constants

The following static constants can be used to specify the type of moving average in the constructor. For example, the following statement: `var ma1 = new MA(10, MA.Simple)` would create a 10 period simple moving average.

`Simple`	Simple moving average.
`Exponential`	Exponential moving average.
`Weighted`	Weighted moving average.
`Triangular`	Triangular moving average.
`VariableVHF`	VHF form of exponential moving average.
`VariableCMO`	A form of variable moving average based on the Chande Momentum Oscillator.
`Vidya`	Variable-Index Dynamic Average (VIDYA).

Methods

The following methods are used to get the current value of the MA, and to add new values to the MA. These functions are documented fully in the pages that follow.

`getValue()`	Return the current MA value.
`getNext()`	Return the new MA value after adding new values to the buffer.

Description

The `MA` datatype allows you to create individual moving average objects, which maintain their own buffers of values, and allow you to feed new values in and get a new average value out.

`MA()` can also be used as a function to return a one-off average of a set of values. In this case, the values to average must be provided as the third parameter. Note that if you specify less values than the period, the first value passed will be repeated to fill the buffer. If you specify more values than the period, the earliest values will “fall out” the moving average (though may still have an effect on the value e.g. if an exponential average is being calculated).

Examples

Using **MA as a function**:

```
MA(10, MA.Simple, [0,1,2,3,4,5,6,7,8,9]); → 4.5

MA(10, MA.Simple, 1,2); → 1.1

MA(10, MA.Simple, [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]); → 10.5
```

Note in the first example, we pass an array to initialise the buffer, and get back a simple average of the ten values.

The second example shows an alternative way of passing values to initialise the buffer. In this case, only 2 values are passed, so the buffer is filled by repeating the first value.

The third example shows how the earliest (leftmost) values are discarded from the buffer. You can therefore pass the whole price history for an instrument to the `MA` function, and get the current moving average value easily:

```
var prices = instrument.getCloseArray();
print(MA(60, MA.Exponential, prices));
```

When using **MA as an object**, you create a new moving average object, then add the values one at a time, getting a new average out each time.

```
var sma = new MA(5, MA.Simple);     // create a 5 period moving average
sma.getNext(1);                     // returns the new MA value of 1
```

The code above creates a 5 period simple moving average. The MA buffer is not initialised. We then call getNext, passing a value of 1. Since the buffer is empty, and we have specified fewer values than the period of 5, the buffer is filled by repeating the value we passed. The MA buffer will now look like this:

1	1	1	1	1

```
sma.getNext(2);                     // returns the new MA value of 1.2
```

The next call to getNext passes a value of 2. This causes the left-most value to be discarded from the buffer, and the new value of 2 is added at the right. The average is then calculated and returned across the 5 new values in the buffer, which looks like this:

1	1	1	1	2

This fuller example below simply prints out a 10-period simple moving average value to the console for the whole price history of an instrument.

```
var prices = instrument.getCloseArray();
var sma = new MA(10, MA.Simple);
for (var i=0; i<prices.length; i++)
{
	var val = sma.getNext(prices[i]);
	print(val);
}
```

MA.getNext()

add value(s) into a moving average, getting the new average value out

Synopsis

ma.getNext(values...)

Arguments

`values...`	One or more numbers, or arrays of numbers. These are added to the moving average buffer from left to right. When a parameter is an array, elements are processed from index 0 to index.length-1.

Returns

A number giving the current MA value.

Description

`getNext()` returns a number which is the current value of a moving average object, after adding one or more values to the moving average buffer.

MA.getValue()

get the current moving average value

Synopsis

ma.`getValue()`

Returns

A number giving the current MA value.

Description

`getValue()` returns a number which is the current value of the moving average object.

NativeLibrary ShareScript v1.2

allows a script to call functions in a DLL (native code library) Object→NativeLibrary

Constructor

`new NativeLibrary(filename)`

The `NativeLibrary()` constructor creates a `NativeLibrary` object which acts as an interface between ShareScope and a DLL file, allowing you to call native code functions in the DLL.

Arguments

`filename`	The name of (or path to) the DLL file to open.

Throws

`RangeError`	If the DLL file cannot be found.
`Error`	If the script is not allowed to access DLL files.

Methods

The `NativeLibrary` object defines a single method to allow you to calls functions in the DLL.

`call()`	Calls a function in the DLL.

Description

The `NativeLibrary` class allows you to call functions exported by a DLL. Access is disabled by default, and must be specifically enabled by the user first.

A script can request the user to enable DLL access by including the following directive:

```
//@UsesDLL:Yes
```

If the user has not enabled DLL access, the `NativeLibrary` constructor will throw an exception.

The calling convention of the called functions can either be the standard calling convention, or the C calling convention (`__stdcall` or `__cdecl`). ShareScript can pass a (limited) range of data types to the function, and can obtain the return value of the function. This is documented below where the `call()` method is described in detail.

The library is opened when the constructor is called, and closed when the `NativeLibrary` object goes out of scope and is garbage collected by the engine.

ShareScope will look for the DLL specified by `filename` in both the main ShareScope directory, and system directories. Normally, you should place DLLs in your ShareScript libraries folder, or alongside the script itself. The examples below show how to specify the DLL filename in both these cases. You can use either forward (/) or back (\) slash characters to separate elements of the path. However, backslashes need to be escaped in strings ("\\").

Examples

The first example shows how to load a library that is located in your ShareScript libraries folder:

```
var myLib = new NativeLibrary("ShareScript/Libraries/MyLib.DLL");
```

The next example shows how to load a library that is located in the same directory as your script:

```
var myLib = new NativeLibrary("ShareScript" + getScriptPath() + "MyLib.DLL");
```

The last example uses a system DLL, calling the Win32 MessageBox function to display an alert to the user, and returns the button clicked back to ShareScript:

```
var user32 = new NativeLibrary("user32.dll");
var ret = user32.call("MessageBoxA", "iiSSi", 0, "Message", "Title", 1);
```

See Also

`getScriptPath()`

NativeLibrary.call()

call a function in a DLL

Synopsis

nativeLibrary.`call(funcName, argTypes, args....)`

Arguments

`funcName`	The name (a string) of the function to call in the DLL.
`argTypes`	A string listing the argument and return types (see below).
`args`	One or more values to pass to the DLL (optional).

Returns

The return value of the called function. This will be forced to the type specified by `argTypes[0]`.

Description

This method calls a function in the DLL wrapped by the `NativeLibrary` object. You must specify the name of the function you want to call, and a string (`argTypes`) specifying the return type and the type of each argument (if any) that you will pass to the function.

The first character of `argTypes` should be the return value of the function which should match that of the C prototype of the function. The allowed values are:

Character for argTypes	C Type
v	void
i	int
f	float

Any subsequent characters in argTypes should be the type of each parameter you will pass to the function (again matching the C prototype):

Character for argTypes	C Type	JavaScript Type you should pass
i	int	Number
f	float	Number
S	char *	String
I	int *	An array of Numbers
F	float *	An array of Numbers

Note that when you pass an array of numbers from ShareScript to your DLL, any modifications made to the block of memory will be reflected back in the JavaScript array when the function returns. This allows your native functions to modify data and pass it back to ShareScript. This is illustrated in the example below.

Example

Suppose we have a C function that adds one to each element of an array of floats:

```
int _stdcall addone(float *p, int len) {
	for (int i = 0; i < len; i++)
		p[i]++;
	return 1;
}
```

You can call this function from ShareScript like this:

```
var myLib = new NativeLibrary("ShareScript\\Libraries\\MyLib.DLL");
var data = new Array(1.5, 2, 3);
myLib.call("addone", "iFi", data, data.length);
print(data);  // will print 2.5, 3, 4
```

Pen

pen style constants

Synopsis

`Pen.`*`constant`*

Constants

Solid	A solid pen.
Dash	A dashed pen.
Dot	A dotted pen.
DashDot	A dash-dot pen.
DashDotDot	A dash-dot-dot pen.

Description

`Pen` is an object that defines constants to create pens. Currently, its sole use is with `Indicator` objects. `Pen` is not a class of objects like `Date` or `MA`, and there is no `Pen()` constructor. It can be considered to be the same type of thing as the JavaScript `Math` object.

See Also

`Indicator.setSeriesLineStyle()`, `Dialog.AddColLinePicker()`

PriceData

represents an OHLCV bar Object→PriceData

Synopsis

`PriceData.`*`property`*

Construction

`PriceData` objects are returned by the `getPrice()` series of `Share` object methods, the `getWeekly/MonthlyBarArray()` methods, the intraday `getIBarArray()` method, or are provided to an `Indicator`'s `getGraph()` function. They cannot be created using the normal JavaScript `new()` operator.

Properties

`open`	The open price in the minor currency unit (e.g. Pence), or `undefined` if no value is available.
`high`	The high price in the minor currency unit (e.g. Pence), or `undefined` if no value is available.
`low`	The low price in the minor currency unit (e.g. Pence), or `undefined` if no value is available.
`close`	The close price in the minor currency unit (e.g. Pence), or `undefined` if no value is available.
`volume`	The volume, or `undefined` if no value is available.
`adjustment`	The adjustment factor, or `undefined` if no value is available. This is always 1 for the most recent price. You can divide a price by the adjustment figure to give the price as it was published on that day.
`date`	The date/time (a JavaScript `Date` object), or `undefined` if no value is available.
`dateNum`	An integer representation of the date (see below for details).
`timeNum`	An integer representation of the time (secs since midnight). **ShareScript v1.1**
`isOHLCV`	A boolean value (`true` or `false`) indicating whether the record is for a day where ShareScope has full OHLCV data. See below for further details.

Description

`PriceData` objects represent a single bar. They contain fields for Open, High, Low & Closing prices, the volume during the period, and the date/time of the *period-end*. When obtained from the `getPrice()` series of `Share` object methods, the period of the returned `PriceData` record(s) is always one day.

The `getWeeklyBarArray()`, `getMonthlyBarArray()` and `getIBarArray()` `Share` object methods return `PriceData` records with the requested periods (weekly, monthly, or a specified intraday period e.g. 5 mins).

An array of `PriceData` objects is also passed by ShareScope to the `Indicator.getGraph()` function. In this case, the period will depend on the graph time or indicator time period selected by the user when the indicator is added.

The `dateNum` & `timeNum` fields provides an integer representation of the date and time. This is made available since JavaScript `Date` objects are relatively costly (in terms of execution speed) to use and compare. You can use the ==, < and > operators to compare two `dateNum` fields (lower `dateNums` are earlier in the calendar, equal `dateNums` represent the same day). If you find yourself comparing dates between two streams of `PriceData` records, consider using `dateNum` instead.

The `isOHLCV` field indicates whether ShareScope has full OHLCV data for the instrument on the date of the `PriceData` record. The ShareScope database has closing price only data for some instruments before a certain date – in this case the `isOHLCV` field will be `false`, and the open, high and low values will be the same as the close value. Note that when the period of a `PriceData` record is *longer* than one day (which can be the case for the price bars passed to a ShareScript indicator, as discussed above), the `isOHLCV` field will always be `true`.

See Also

`Indicator`, `dateNum()`, `timeNum()`, `Share.getPrice()`, `Share.getPriceOnDate()`, `Share.getPriceArray()`, `Share.getPriceArrayDates()`,`Share.getIBarArray()`, `Share.getWeeklyBarArray()`, `Share.getMonthlyBarArray()`, `Share.getCurrency()`

print()

writes text to the console

Synopsis

`print(s)`

Arguments

`s`	A string containing the text to be written.

Description

`print()` is a global function that writes text to the ShareScript console. The `print()` function automatically generates a new line, but you can also use the "`\n`" sequence within the string to produce a line-break and output multiple lines of text from one string. The console window will also be shown if it is not currently visible.

See Also

`clear()`

RA

constants used to specify a Risk Analysis

Synopsis

`RA.`*`constant`*

Constants

Constant	Risk Analysis Type
`MeanReturn`	Mean Return on Investment %
`DevReturns`	Deviation of Returns %
`MeanActiveReturn`	Mean Active Return %
`DevActiveReturns`	Deviation of Active Returns %
`SharpeRatio`	Sharpe Ratio
`SortinoRatio`	Sortino Ratio
`RSquare`	R-Square
`MSquare`	M-Square
`Beta`	Beta
`Alpha`	Alpha %
`JensensAlpha`	Jensen's Alpha % **ShareScript v1.35**
`Treynor`	Treynor Performance Index %
`InfoRatio`	Information Ratio
`Volatility`	Volatility %
`Correlation`	Correlation

The following constants are defined to specify the **periodType** property:

`Daily, Weekly, Monthly, Quarterly, SemiAnnually, Annually`

The following constants are defined to specify the **periodPrice** property:

`LastPrice, AveragePrice, TypicalPrice, WeightedPrice`

Description

`RA` is a global object that defines constants that refer to the various types of risk analysis that can be performed. It also defines two additional sets of constants for specifying the analysis period length, and the means by which a price for the period is determined.

See Also

`Share.getRiskAnalysis()`

Result

constants to identify different company results

Synopsis

`Result.`*`constant`*

Constants

Constant	Value returned by **`Share.getResult()`** or **`Share.getResultArray()`**
`ResultType`	A number giving the result type (see the `ResultType` object).
`Type`	A string giving the result type (e.g. "Forecast").

Constant	Value returned by `Share.getResult()` or `Share.getResultArray()`
`Date`	A `Date` object giving the year end date.
`Profit`	A number giving the profits (in millions).
`EPS`	A number giving the EPS (in the minor currency unit). This value will be the normalised figure if available, announced otherwise (see `ResultType`).
`Dividend`	A number giving the total dividend for the year, but excluding any special dividends (in the minor currency unit).
`Turnover`	A number giving the turnover (in millions).
`ExDivDate`	A `Date` object giving the ex-dividend date.
`DivPayDate`	A `Date` object giving the dividend pay date.
`IsIFRS`	A boolean value indicating whether the result is IFRS.

Constant	Value returned by `Share.getResult()`
`NormPreTax`	Normalised pre-tax profits (in millions).
`NormPreTaxPS`	Normalised pre-tax profits per share. **ShareScript v1.35**
`NormPostTax`	Normalised post-tax profits. **ShareScript v1.35**
`NormEPS`	Normalised EPS (same as `Result.EPS`).
`ReportedPreTax`	Reported pre-tax profits (same as `Result.Profit`).
`ReportedPreTaxPS`	Reported pre-tax profits per share.
`ReportedPostTax`	Reported post-tax profits.
`ReportedEPS`	Reported EPS (same as deprecated `ReportedPostTaxPS`)
`TurnoverPS`	Turnover per share.
`Tax`	Tax paid.
`BookValue`	Book value (NAV).
`BookValuePS`	Book value per share.
`TangibleBookValue`	Tangible book value (NTAV).
`TangibleBookValuePS`	Tangible book value per share.
`Cash`	Net cash.
`CashPS`	Net cash per share.
`CashFlow`	Net cash flow.
`CashFlowPS`	Net cash flow per share.
`Capex`	Capital expenditure.
`CapexPS`	Capital expenditure per share.
`RD`	R&D expenditure.
`RDPS`	R&D expenditure per share.
`Depreciation`	Depreciation of tangible assets.
`ROCE`	Return on capital employed (ROCE).

Constant	Value returned by `Share.getResult()`	
`ROE`	Return on equity (ROE).	
`QuickRatio`	Quick ratio.	
`CurrentRatio`	Current ratio.	
`OperatingMargin`	Operating margin (OM).	
`InterestCover`	Interest cover.	
`InterestPaid`	Interest paid.	
`NetBorrowing`	Net borrowing.	
`NetCurrentAssets`	Net current assets.	
`NetGearing`	Net gearing (including intangibles).	
`NetGearingEx`	Net gearing (excluding intangibles).	
`CashPercent`	Cash % (including intangibles).	
`CashPercentEx`	Cash % (including intangibles).	
`GrossGearing`	Gross gearing (including intangibles).	
`GrossGearingEx`	Gross gearing (including intangibles).	
`GrossGearing5`	Gross gearing under 5 years (including intangibles).	
`GrossGearing5Ex`	Gross gearing under 5 years (including intangibles).	
`GrossGearing1`	Gross gearing under 1 year (including intangibles).	
`GrossGearing1Ex`	Gross gearing under 1 year (including intangibles).	
`OperatingProfit`	Operating profit (reported).	ShareScript v1.35
`GrossProfit`	Gross profit.	ShareScript v1.35
`TotalProfit`	Total post-tax profit from all ops (inc discontd)	ShareScript v1.35
`TotalEPS`	Total post-tax EPS	ShareScript v1.35
`DividendYield`	Dividend yield (price at year end).	ShareScript v1.35
`GrossMargin`	Gross margin %.	ShareScript v1.35
`CostOfSales`	Cost of goods sold.	ShareScript v1.35
`FreeCashFlow`	Free cash flow.	ShareScript v1.35
`OperatingCashFlow`	Operating cash flow.	ShareScript v1.35
`EBIT`	Reported EBIT.	ShareScript v1.35
`EBITDA`	Reported EBITDA.	ShareScript v1.35
`NormEBIT`	Normalised EBIT.	ShareScript v1.35
`NormEBITDA`	Normalised EBITDA.	ShareScript v1.35
`RetainedProfit`	Retained proft.	ShareScript v1.35
`NumEmployees`	Number of employees.	ShareScript v1.35
`EnterpriseValue`	Enterprise value.	ShareScript v1.35

Constant	Value returned by `Share.getResult()`	
`CurrentAssets`	Current assets.	**ShareScript v1.35**
`TotalAssets`	Total assests.	**ShareScript v1.35**
`CurrentLiabilities`	Current liabilities.	**ShareScript v1.35**
`TotalLiabilities`	Total liabilities.	**ShareScript v1.35**
`TotalExpenses`	Total expenses.	**ShareScript v1.35**
`AdminExpenses`	Administrative expenses.	**ShareScript v1.35**
`NumShares`	Number of shares at year end.	**ShareScript v1.35**
`NumSharesAv`	Average number of shares across year.	**ShareScript v1.35**
`DebtToCapital`	Debt to capital.	**ShareScript v1.35**
`DebtToEquity`	Debt to equity.	**ShareScript v1.35**
`EarningsYield`	Earnings yield.	**ShareScript v1.35**
`DividendCover`	Dividend cover.	**ShareScript v1.35**
`ROA`	Return on assets/investment.	**ShareScript v1.35**
`ROCI`	Return on capital invested.	**ShareScript v1.35**
`CROCI`	Cash return on capital invested.	**ShareScript v1.35**
`AltmanZScore`	Altman Z-score.	**ShareScript v1.35**
`GreenblattRank`	Greenlatt magic formula rank.	**ShareScript v1.35**
`GrahamNumber`	Graham Number.	**ShareScript v1.35**
`NeffTRR`	John Neff's total return ratio.	**ShareScript v1.35**
`PiotroskiFScore`	Piotroski F-score.	**ShareScript v1.35**
`WorkingCapital`	Current assets minus current liabilities	**ShareScript v1.35**
`Minorities`	Share of profits owned by non-controlling ints.	**ShareScript v1.35**
`TotalEquity`	Total assets minus total liabilities (or NAV+min)	**ShareScript v1.35**

Description

`Result` is a global object that defines constants that refer to the various company results in ShareScope's database. These can be used with the `Share` object methods `getResult()` and `getResultArray()` functions as follows –

```
var profit = my_share.getResult(0, Result.Profit);
```

Note that `getResultArray()` only accepts the first set of constants, not the second.

Result is not a class of objects like `Date` or `MA`, and there is no `Result()` constructor. It can be considered to be the same type of thing as the JavaScript `Math` object.

See Also

`ResultType`, `Share.getResult()`, `Share.getResultArray()`

ResultType

constants to identify the different types of company results

Synopsis

`ResultType.`*`constant`*

Constants

`Announced`	Result has been announced but is not yet normalised.
`Final`	Result is has been normalised.
`Forecast`	Forecast result.
`Q1`	Quarter 1 result.
`Interim`	Interim (or Quarter 2) result.
`Q3`	Quarter 3 result.
`Special`	Special Dividend result.

Description

`ResultType` is a global object that defines constants that refer to the various types of company results in ShareScope's database. These are used to identify the return values from the `Share` object methods `getResult()` and `getResultArray()` functions when `Result.ResultType` is requested.

These constants allow you to check result types numerically, rather than having to compare the strings returned by `Result.Type`.

Note that only the first 3 values (`Announced`, `Final` & `Forecast`) can be returned by `getResult()` since the other types are not year-end results. These other types are returned when the complete set of results for the year is requested using `getResultArray()`.

See Also

`Result`, `Share.getResult()`, `Share.getResultArray()`

Share

represents a ShareScope instrument Object→Share

Construction

Because `Share` objects represent the individual underlying ShareScope instruments, they are finite in number and are not created using the normal JavaScript `new()` operator. Instead you can use the global `getShare()` function to retrieve a `Share` object for a specific instrument.

Methods

`Share` objects have no properties, but instead have a number of different methods that allow you to retrieve information about the underlying ShareScope instrument. These methods are listed below, and documented fully on the following pages.

Note that since some companies possess multiple instruments, some methods return information about the *instrument* itself (e.g. `getShareName()` might return "10p ord"). Other methods return information about the *company* that the instrument belongs to (e.g. `getName()` might return "Lloyds TSB Plc").

`getActivities()`	Get a brief description of the company's activities.	**ShareScript v1.32**
`getFullActivities()`	Get a full description of the company's activities.	**ShareScript v1.32**
`getAssocShares()`	Get an array of all `Share` objects belonging to the same company.	
`getCap()`	Get the market capitalisation the company.	
`getClose()`	Get a single closing price for the instrument.	
`getCloseArray()`	Get an array of closing prices for the instrument.	
`getCloseArrayDates()`	Get an array of closing prices for the instrument between two dates.	
`getCloseOnDate()`	Get a single closing price for the instrument on a specific date.	
`getCurrency()`	Get the price currency of the instrument.	
`getCurrencyR()`	Get the result reporting currency for the company.	
`getEMS()`	Get the exchange market size (EMS) for the instrument. Replaces NMS.	**ShareScript v1.32**
`getEPIC()`	Get the instrument's EPIC code.	
`getHigh()`	Get a single day's high price for the instrument.	
`getHighOnDate()`	Get a single day's high price for the instrument on a specific date.	
`getIndices()`	Get the index membership of the company.	
`getIndustry()`	Get the industry of the company.	
`getISIN()`	Get the instrument's ISIN.	**ShareScript v1.2**
`getIBarArray()`	Get intraday OHLCV bars.	**ShareScript v1.1**
`getIBarArrayOnDate()`	Get intraday OHLCV bars for a specific date.	**ShareScript v1.1**
`getIBid()`	Get the latest intraday bid price.	**ShareScript v1.1**
`getIBidOfferArray()`	Get an array of intraday Bid/Offer records.	**ShareScript v1.1**
`getIBidOfferArrayOnDate()`	Get an array of intraday Bid/Offer records for a specific date.	**ShareScript v1.1**
`getIClose()`	Get an intraday close price.	**ShareScript v1.1**
`getIDate()`	Get the date of intraday data.	**ShareScript v1.1**
`getIDateNum()`	Get the date (as a dateNum) of intraday data.	**ShareScript v1.1**
`getIMid()`	Get the latest intraday mid price.	**ShareScript v1.1**
`getIMidHigh()`	Get the intraday mid high price.	**ShareScript v1.3**
`getIMidLow()`	Get the intraday mid low price.	**ShareScript v1.3**
`GetIOffer()`	Get the latest intraday offer price.	**ShareScript v1.1**
`getIOpen()`	Get an intraday open price.	**ShareScript v1.1**
`getITradeArray()`	Get an array of intraday trade records.	**ShareScript v1.1**

`getITradeArrayOnDate()`	Get an array of intraday trade records for a specific date.	**ShareScript v1.1**
`getITradeHigh()`	Get the intraday trade high price.	**ShareScript v1.3**
`getITradeLow()`	Get the intraday trade low price.	**ShareScript v1.3**
`getListing()`	Get LSE instrument's listing (Full or AIM).	**ShareScript v1.3**
`getLow()`	Get a single day's low price for the instrument.	
`getLowOnDate()`	Get a single day's low price for the instrument on a specific date.	
`getMarket()`	Get ShareScope's exchange (or market) code for the instrument.	
`getMarketOpenTime()`	Get the market open time for the instrument.	**ShareScript v1.1**
`getMarketCloseTime()`	Get the market close time for the instrument.	**ShareScript v1.1**
`getMarketOffsetGMT()`	Get the market offset from GMT.	**ShareScript v1.1**
`getMonthlyBarArray()`	Get monthly OHLCV bars for the instrument.	**ShareScript v1.2**
`getName()`	Get the full name of a company or instrument.	
`getNotes()`	Get the share's notes column values.	**ShareScript v1.1**
`getNMS()`	Get the normal market size (NMS) for the instrument.	
`getNumShares()`	Get the number of shares.	
`getOpen()`	Get a single opening price for the instrument.	
`getOpenOnDate()`	Get a single opening price for the instrument on a specific date.	
`getPrice()`	Get a single OHLCV price record for the instrument.	
`getPriceArray()`	Get an array of OHLCV price records for the instrument.	
`getPriceArrayDates()`	Get an array of OHLCV price records for the instrument between two dates.	
`getPriceOnDate()`	Get a single OHLCV price record for the instrument on a specific date.	
`getResult()`	Provides basic access to the company's historic and forecast results.	
`getResultArray()`	Provides advanced access to the company's historic and forecast results.	
`getRiskAnalysis()`	Provides access to the Risk Analysis metrics for the instrument.	**ShareScript v1.31**
`getSector()`	Get the sector of the company.	
`getSectorIndex()`	Get a Share object corresponding to the sector index for the company.	
`getSEDOL()`	Get the instrument's SEDOL.	**ShareScript v1.3**
`getShareName()`	Get the name of the instrument.	
`getShareNum()`	Get the share number for the instrument.	**ShareScript v1.1**
`getShareScopeID()`	Get the ShareScope ID for the instrument.	**ShareScript v1.1**

`getSubSector()`	Get the sub-sector of the company.
`getSuperSector()`	Get the super-sector of the company. **ShareScript v1.3**
`getType()`	Get the type of the instrument.
`getTradingSystem()`	Get the instrument's LSE trading system. **ShareScript v1.3**
`getUncrossingPrice()`	Get the indicative auction uncrossing price. Only valid if the share is in an intraday auction, and there is a valid uncrossing price. **ShareScript v1.34**
`getVolume()`	Get a single day's volume for the instrument.
`getVolumeOnDate()`	Get a single day's volume for the instrument on a specific date.
`getWeeklyBarArray()`	Get weekly OHLCV bars for the instrument. **ShareScript v1.2**
`isInAuction()`	Returns true if share is in intraday auction. **ShareScript v1.34**
`isSuspended()`	Returns true if a company's shares are suspended.

Description

Every ShareScope instrument has a corresponding ShareScript `Share` object. You can obtain a reference to a specific `Share` object using the `getShare()` function. There are also a number of other functions that return either individual `Share` objects, or arrays of `Share` objects.

Often, you will receive a reference to a `Share` object when ShareScope passes one to specific functions you define on `Column` and `Indicator` objects.

Note that you can compare `Share` object references with the equality operator, which returns `true` if the underlying instrument is the same (since both references will be to the same `Share` object).

Once you have a `Share` object, you can get information about the instrument (e.g. company results, price history) by using one of the many `Share` object methods.

Example

This simple example requests an instrument from its EPIC, then gets the name (a string) of the company that the instrument belongs to.

```
var my_share = getShare("LSE:LLOY");
var name = my_share.getName();
```

Share.getActivities() ShareScript v1.32

get a short description of the company's activities

Synopsis

share.`getActivities()`

Returns

A string which describes the activities of the company. E.g. "Provider of communications services". An empty string is returned if ShareScope does not have information for the instrument.

Example

```
var vod = getShare("LSE:VOD");
print(vod.getActivities());
```

See Also

`Share.getFullActivities()`, `Share.getSector()`

Share.getFullActivities() ShareScript v1.32

get a detailed description of the company's activities

Synopsis

share.getFullActivities()

Returns

A string which describes the full activities of the company. The returned string may be very long and does not contain line breaks. An empty string is returned if ShareScope does not have information for the instrument.

See Also

`Share.getActivities()`, `Share.getSector()`

Share.getAssocShares()

get an array of all Share objects belonging to the same company

Synopsis

share.getAssocShares()

Returns

An array of `Share` objects, all of which belong to the same company as *share*, the instrument on which this method was called. Note that *share* will be included in the returned array.

Description

`getAssocShares()` is a `Share` object method. It returns an array containing the full set of instruments belonging to the same company as the `Share` object on which the method is called. If the company has e.g. only a single ordinary share, then the `Share` object itself will be the sole element of the returned array. The first element of the array is always the company's primary ordinary share.

Example

```
var my_share = getShare("LSE:HBOS");
var list = my_share.getAssocShares();      // returns an array of 5 Share objects
list[0].getShareName();                    // 25p Ords
list[1].getShareName();                    // 6.475% Non-Cumulative Preference £1
```

See Also

`Share.getSectorIndex()`, `getShare()`, `getList()`, `getPortfolio()`, `Share.getShareNum()`

Share.getCap()

get the market capitalisation the company

Synopsis

share.getCap()

Returns

A number giving the market capitalisation of the company (in Millions). The currency of this value can be obtained from `Share.getCurrency()`.

See Also

`Share.getCurrency()`, `Share.getNumShares()`

Share.getClose()

get a single closing price for the instrument

Synopsis

share.getClose()
share.getClose(daysAgo)

Arguments

daysAgo	An optional integer specifying the number of trading days ago that you want to get a closing price for (0 is the most recent close, 1 is yesterday, etc).

Returns

A number giving the closing price of the instrument in the minor currency unit. `Undefined` can be returned if a price is not available (e.g. the requested day lies before the start of the instrument's price history).

Description

`getClose()` is a `Share` object method that returns the requested day's closing price. If the `daysAgo` parameter is not specified, `getClose()` returns the most recent close. If you require a full OHLCV record, consider using `Share.getPrice()` instead.

If you require closing prices across many days, consider using `Share.getCloseArray()`.

See Also

`Share.getCloseArray()`, `Share.getCloseOnDate()`, `Share.getCloseArrayDates()`, `Share.getOpen()`, `Share.getHigh()`, `Share.getLow()`, `Share.getVolume()`, `Share.getPrice()`, `Share.getCurrency()`

Share.getCloseArray()

get an array of closing prices for the instrument

Synopsis

share.getCloseArray()
share.getCloseArray(num)

Arguments

num	An optional integer specifying the number of prices required. e.g. 10 will return the most recent 10 closing prices.

Returns

An array of numbers giving the closing price on each of the days requested. The oldest record is first (array element 0). If more days are requested than are available in the price history, the length of the array may be shorter than `num`.

Description

`getCloseArray()` is a `Share` object method that returns multiple closing prices from the history. If the optional `num` parameter is not used, the entire closing price history is returned. The oldest price is at array[0]. The most recent price will be at array[array.length-1].

See Also

`Share.getPriceArray()`, `Share.getCloseArrayDates()`, `Share.getClose()`, `Share.getCloseOnDate()`, `Share.getCurrency()`

Share.getCloseArrayDates()

get an array of closing prices for the instrument between two dates

Synopsis

```
share.getCloseArrayDates()
share.getCloseArrayDates(start)
share.getCloseArrayDates(start, end)
```

Arguments

`start`	An optional JavaScript `Date` object specifying the date on which to start returning prices. If not specified, the start of the history is used.
`end`	An optional JavaScript `Date` object specifying the date on which to stop returning prices. If not specified, the end of the history is used.

Returns

An array of numbers giving the closing price on each of the days between the `start` and `end` dates (inclusive). The oldest record is first (array element 0). Records will not be returned for dates on which trading did not take place (e.g. weekends and holidays). The length of the array may be zero if no records match the criteria.

Description

`getCloseArrayDates()` is a `Share` object method that returns multiple closing prices from the history. If the `end` date is not specifed, the end of the history is assumed. If the `start` date is not specified, the start of the history is assumed. The oldest price is at array[0]. The most recent price will be at array[array.length-1].

See Also

`Share.getCloseArray()`, `Share.getCloseOnDate()`, `Share.getClose()`, `Share.getPriceArrayDates()`, `Share.getCurrency()`

Share.getCloseOnDate()

get a single closing price for the instrument on a specific date

Synopsis

```
share.getCloseOnDate(date)
```

Arguments

`date`	A JavaScript `Date` object (or alternatively a dateNum) specifying the date on which to return the closing price.

Returns

A number giving the closing price of the instrument in the minor currency unit. `Undefined` can be returned if the requested date lies outside of the instrument's history.

Description

`getCloseOnDate()` is a `Share` object method that returns the requested day's closing price. If `date` falls on a holiday or weekend, then the price on the preceding trading day will be returned.

If you require closing prices across many days, consider using `Share.getCloseArray()`.

Example

```
var close = my_share.getCloseOnDate(new Date(2007, 5, 13));
```

See Also

`getClose()`, `getCloseArray()`, `getCloseArrayDates()`, `getOpenOnDate()`, `getHighOnDate()`, `getLowOnDate()`, `getVolumeOnDate()`, `getPriceOnDate()`, `getCurrency()`

Share.getCurrency()

get the price currency of the instrument

Synopsis

share.`getCurrency()`

Returns

A string giving the ISO code for the currency of the instrument's price. Note that the code is for the major currency unit, while prices are always expressed in the minor unit.

See Also

`Share.getCurrencyR()`

Share.getCurrencyR()

get the result reporting currency for the company

Synopsis

share.`getCurrencyR()`

Returns

A string giving the ISO code for the currency of the company's results.

See Also

`Share.getCurrency()`

Share.getEMS() ShareScript v1.32

get the exchange market size (EMS) for the instrument

Synopsis

share.getEMS()

Returns

A number giving the exchange market size for the instrument (the unit is shares). This information is only available for a company's primary share. Note that EMS is equivalent to the old NMS and this method should be used in place of Share.getNMS().

See Also

Share.getAssocShares()

Share.getEPIC()

get the instrument's EPIC code

Synopsis

share.getEPIC()

Returns

A string giving the EPIC code of the instrument.

See Also

Share.getName(), Share.getMarket()

Share.getHigh()

get a single day's high price for the instrument

Synopsis

share.getHigh()
share.getHigh(daysAgo)

Arguments

daysAgo	An optional integer specifying the number of trading days ago that you want to get the day's high price for (0 is the most recent high, 1 is yesterday, etc).

Returns

A number giving the high price of the instrument in the minor currency unit. Undefined can be returned if a price is not available (e.g. the requested day lies before the start of the instrument's price history).

Description

getHigh() is a Share object method that returns the requested day's high price. If the daysAgo parameter is not specified, getHigh() returns the most recent day's high. If you require a full OHLCV record for an instrument, consider using Share.getPrice() instead.

See Also

`Share.getHighOnDate()`, `Share.getClose()`, `Share.getOpen()`, `Share.getLow()`, `Share.getVolume()`, `Share.getPrice()`, `Share.getCurrency()`

Share.getHighOnDate()

get a single day's high price for the instrument on a specific date

Synopsis

share.`getHighOnDate(date)`

Arguments

`date`	A JavaScript `Date` object (or alternatively a dateNum) specifying the date on which to return the day's high price.

Returns

A number giving the high price of the instrument in the minor currency unit. `Undefined` can be returned if the requested date lies outside of the instrument's history.

Description

`getHighOnDate()` is a `Share` object method that returns the requested day's high price for an instrument. If `date` falls on a holiday or weekend, then the price on the preceding trading day will be returned.

See Also

`Share.getHigh()`, `Share.getCloseOnDate()`, `Share.getOpenOnDate()`, `Share.getLowOnDate()`, `Share.getVolumeOnDate()`, `Share.getPriceOnDate()`, `Share.getCurrency()`

Share.getIndices()

get the index membership of the company

Synopsis

share.`getIndices()`

Returns

An array of strings with the names of the indices to which the company belongs. An empty array will be returned if the company belongs to no indices.

Description

`getIndices()` is a `Share` object method that returns the index membership of the company to which the `Share` object belongs.

The possible strings returned are – "`FT 30`", "`FTSE 100`", "`FTSE 250`", "`FTSE 350`", "`FTSE SmallCap`", "`FTSE All-Share`", "`FTSE Fledgling`", "`FTSE All-Small`", "`techMark`", "`techMark 100`", "`techMark mediscience`", "`DJ 30`", "`NASDAQ 100`".

See Also

`Share.getSectorIndex()`, `getList()`

Share.getIndustry()

get the industry of the company

Synopsis

share.getIndustry()

Returns

A string which gives the industry that the company is engaged in (e.g. "Financials").

See Also

Share.getSector(), Share.getSubSector(), Share.getSuperSector(), Share.getSectorIndex()

Share.getISIN() ShareScript v1.2

get the ISIN for an instrument

Synopsis

share.getISIN()

Returns

A string which gives the instrument's ISIN.

See Also

Share.getEPIC(), Share.getName(), Share.getShareScopeID(), Share.getSEDOL()

Share.getIBarArray() ShareScript v1.1

get intraday OHLCV bars for the instrument

Synopsis

share.getIBarArray()
share.getIBarArray(daysAgo, period)

Arguments

daysAgo	The index back into the set of intraday days (0 is the most recent day, 1 is previous, etc). Note that the previous day will be the previous day for which intraday data is available, not the previous trading day (see Share.getIDate()).
period	The bar length in seconds (must be at least 15). If not specified, 5 minute bars will be created.

Returns

An array of PriceData objects, or undefined if no data is available.

Description

getIBarArray() is a Share object method that returns intraday OHLCV bars (by default, 5 minute bars for the most recent day).

The bars returned will correspond to intraday chart bars with the following settings: (i) start with a full period, (ii) data derived from mid prices, (iii) normal market hours. Note that intraday bars, where possible, use the first price within a bar as the open, rather than the last price of the previous bar.

If you need bars calculated a different way, you can use the `Share.getITradeArray()` or `Share.getIBidOfferArray()` methods to obtain the raw data, then process the data as required.

Example

The following example obtains 1 hours bars for a share, for the most recent day, then prints the number of bars returned.

```
var share = getShare("LSE:TSCO");
var bars = share.getIBarArray(0, 60*60);
print (bars.length);
```

See Also

`PriceData`, `Share.getIBarArrayOnDate()`, `Share.getITradeArray()`, `Share.getIBidOfferArray()`

Share.getIBarArrayOnDate() ShareScript v1.1

get intraday OHLCV bars for the instrument for a specific date

Synopsis

share.`getIBarArrayOnDate(date, period)`

Arguments

`date`	A JavaScript `Date` object (or alternatively a dateNum) specifying the date for which to return the intraday price history.
`period`	The bar length in seconds (must be at least 15). If not specified, 5 minute bars will be created.

Returns

An array of `PriceData` objects, or `undefined` if no data is available.

Description

`getIBarArrayOnDate()` is a `Share` object method that returns intraday OHLCV bars for the specified date. See the entry for `Share.getIBarArray()` for full details.

See Also

`PriceData`, `Share.getIBarArray()`

Share.getIBid() ShareScript v1.1

get the latest intraday bid price for the instrument

Synopsis

share.`getIBid()`

Returns

A number giving the latest intraday bid price for the instrument in the minor currency unit. `Undefined` can be returned if a price is not available.

Description

This method is similar to `Share.getIMid()`. See the entry for that method for more details.

See Also

`Share.getIMid()`, `Share.getIOffer()`, `Share.getCurrency()`

Share.getIBidOfferArray() ShareScript v1.1

get the intraday price history for the instrument

Synopsis

share.getIBidOfferArray()
share.getIBidOfferArray(daysAgo)

Arguments

daysAgo	The index back into the set of intraday days (0 is the most recent day, 1 is previous, etc). Note that the previous day will be the previous day for which intraday data is available, not the previous trading day (see `Share.getIDate()`).

Returns

An array of `BidOfferData` objects, or `undefined` if no data is available.

Description

`getIBidOfferArray()` is a `Share` object method that returns an intraday price history (by default for the most recent day).

When a share is in intraday auction, ShareScope stops updating the bid/offer values and only updates the uncrossing price. In this case the history will contain some records with the uncrossing price instead of the bid/offer. If you wish to exclude these records from the history, you can inspect the `isInAuction` property of the `BidOfferData` records.

See Also

`BidOfferData`, `Share.getIBidOfferArrayOnDate()`

Share.getIBidOfferArrayOnDate() ShareScript v1.1

get the intraday price history for the instrument for a specific date

Synopsis

share.getIBidOfferArrayOnDate(date)

Arguments

date	A JavaScript `Date` object (or alternatively a dateNum) specifying the date for which to return the intraday price history.

Returns

An array of `BidOfferData` objects, or `undefined` if no data is available.

Description

`getIBidOfferArrayOnDate()` is a `Share` object method that returns an intraday price history for the selected date.

See Also

`BidOfferData`, `Share.getIBidOfferArray()`

Share.getIClose() ShareScript v1.1

get an intraday close price for the instrument

Synopsis

share.`getIClose()`
share.`getIClose(daysAgo)`

Arguments

`daysAgo`	The index back into the set of intraday days (0 is the most recent day, 1 is previous, etc). Note that the previous day will be the previous day for which intraday data is available, not the previous trading day (see `Share.getIDate()`).

Returns

A number giving the intraday close price for the instrument in the minor currency unit. `Undefined` can be returned if a price is not available (e.g. prior to the close).

Description

`getIClose()` is a `Share` object method that returns a closing price from the intraday data (by default from the most recent day). For LSE shares, this method will return a value shortly after the close at 4.30pm.

You can use the `Share.getIDate()` method to check the date of an intraday day. See the entry for that method for more details about indexing into the set of intraday days.

If you require a full intraday price history, you can use one of the `Share.getIBidOfferArray()` or `Share.getIBarArray()` methods.

See Also

`Share.getIOpen()`, `Share.getIMid()`, `Share.getIBidOfferArray()`, `Share.getIBarArray()`, `Share.getIDate()`, `Share.getCurrency()`

Share.getIDate() ShareScript v1.1

get the date of an intraday day

Synopsis

share.`getIDate()`
share.`getIDate(daysAgo)`

Arguments

`daysAgo`	The index back into the set of intraday days (0 is the most recent day, 1 is previous, etc). Note that the previous day will be the previous day for which intraday data is available, not the previous trading day.

Returns

A JavaScript `date` object, giving the date of an intraday day. `Undefined` will be returned if you index beyond the start of the set of intraday data.

Description

`getIDate()` is a `Share` object method that returns the date of the most recent intraday day, or the date of a previous day when the `daysAgo` parameter is used.

ShareScope's intraday data may have missing days (if ShareScope is not connected to the intraday feed every day). The `daysAgo` parameter used by this (and other intraday data methods) allows you to index backwards into the intraday days present in ShareScope's intraday database without knowing the dates for which data is available.

For example, if you connected to the intraday feed on Monday, but not Tuesday, then connected again on Wednesday, a `daysAgo` value of "0" would return data for Wednesday, and `daysAgo` value of "1" would return Monday's data.

See Also

`Share.getIDateNum()`, `Share.getIClose()`, `share.getIOpen()`, `share.getITradeArray()`, `share.getIBidOfferArray()`, `share.getIBarArray()`

Share.getIDateNum() ShareScript v1.1

get the dateNum of an intraday day

Synopsis

share.`getIDateNum()`
share.`getIDateNum(daysAgo)`

Arguments

`daysAgo`	The index back into the set of intraday days (0 is the most recent day, 1 is previous, etc). Note that the previous day will be the previous day for which intraday data is available, not the previous trading day (see `Share.getIDate()`).

Returns

An integer dataNum value giving the date of an intraday day. `Undefined` will be returned if you index beyond the start of the set of intraday days.

See Also

`dateNum()`, `Share.getIDateNum()`

Share.getIMid() ShareScript v1.1

get the latest intraday mid price for the instrument

Synopsis

share.`getIMid()`

Returns

A number giving the latest intraday mid price for the instrument in the minor currency unit. `Undefined` can be returned if a price is not available.

Description

`getIMid()` is a `Share` object method that returns the latest intraday mid price available for the instrument. You can use the `Share.getIDate()` or `Share.getIDateNum()` methods to check the date of the latest intraday data.

If you require a full intraday price history, you can use one of the `Share.getIBidOfferArray()` or `Share.getIBarArray()` methods.

See Also

`Share.getIBid()`, `Share.getIOffer()`, `Share.getIBidOfferArray()`, `Share.getIDate()`, `Share.getCurrency()`

Share.getIMidHigh() ShareScript v1.3

get the intraday mid high price for the instrument

Synopsis

share.`getIMidHigh()`

Returns

A number giving the latest intraday mid high price for the instrument in the minor currency unit. `Undefined` can be returned if a price is not available.

Description

`getIMidHigh()` is a `Share` object method that returns the latest intraday mid high price available for the instrument. You can use the `Share.getIDate()` or `Share.getIDateNum()` methods to check the date of the latest intraday data.

See Also

`Share.getIMidLow()`, `Share.getITradeHigh()`, `Share.getITradeLow()`

Share.getIMidLow() ShareScript v1.3

get the intraday mid low price for the instrument

Synopsis

share.`getIMidLow()`

Returns

A number giving the latest intraday mid low price for the instrument in the minor currency unit. `Undefined` can be returned if a price is not available.

Description

`getIMidLow()` is a `Share` object method that returns the latest intraday mid low price available for the instrument. You can use the `Share.getIDate()` or `Share.getIDateNum()` methods to check the date of the latest intraday data.

See Also

`Share.getIMidHigh()`, `Share.getITradeHigh()`, `Share.getITradeLow()`

Share.getIOffer() ShareScript v1.1

get the latest intraday offer price for the instrument

Synopsis

share.`getIOffer()`

Returns

A number giving the latest intraday offer price for the instrument in the minor currency unit. `Undefined` can be returned if a price is not available.

Description

This method is similar to `Share.getIMid()`. See the entry for that method for more details.

See Also

`Share.getIBid()`, `Share.getIMid()`, `Share.getCurrency()`

Share.getIOpen() ShareScript v1.1

get an intraday opening price for the instrument

Synopsis

```
share.getIOpen()
share.getIOpen(daysAgo)
```

Arguments

daysAgo	The index back into the set of intraday days (0 is the most recent day, 1 is previous, etc). Note that the previous day will be the previous day for which intraday data is available, not the previous trading day (see `Share.getIDate()`).

Returns

A number giving the intraday open price for the instrument in the minor currency unit. `Undefined` can be returned if a price is not available.

Description

`getIOpen()` is a `Share` object method that returns an opening price from the intraday data (by default from the most recent day).

You can use the `Share.getIDate()` method to check the date of an intraday day. See the entry for that method for more details about indexing into the set of intraday days.

See Also

`Share.getIClose()`, `Share.getIBidOfferArray()`, `Share.getIDate()`, `Share.getCurrency()`

Share.getITradeArray() ShareScript v1.1

get the intraday trade history for the instrument

Synopsis

```
share.getITradeArray()
share.getITradeArray(daysAgo)
```

Arguments

daysAgo	The index back into the set of intraday days (0 is the most recent day, 1 is previous, etc). Note that the previous day will be the previous day for which intraday data is available, not the previous trading day (see `Share.getIDate()`).

Returns

An array of `TradeData` objects, or `undefined` if no data is available.

Description

`getITradeArray()` is a `Share` object method that returns a full intraday price history (by default for the most recent day).

Example

The following example prints the size of any uncrossing trades (and the time they took place) in the latest day's intraday history for the Royal Bank of Scotland.

```
var rbs = getShare("LSE:RBS");
var trades = rbs.getITradeArray();
for (var i = 0; i < trades.length; i++)
{
      if (trades[i].type == TradeType.UT)
            print(trades[i].volume + " on " + trades[i].date);
}
```

See Also

`TradeData`, `Share.getITradeArrayOnDate()`

Share.getITradeArrayOnDate() ShareScript v1.1

get the intraday trade history for the instrument for a specific date

Synopsis

share.`getITradeArrayOnDate(date)`

Arguments

`date`	A JavaScript `Date` object (or alternatively a dateNum) specifying the date for which to return the intraday trade history.

Returns

An array of `TradeData` objects, or `undefined` if no data is available.

Description

`getITradeArray()` is a `Share` object method that returns a full intraday trade history for the selected date.

See Also

`TradeData`, `Share.getITradeArray()`

Share.getITradeHigh() ShareScript v1.3

get the intraday trade high price for the instrument

Synopsis

share.`getITradeHigh()`

Returns

A number giving the latest intraday trade high price for the instrument in the minor currency unit. `Undefined` can be returned if a price is not available.

Description

`getITradeHigh()` is a `Share` object method that returns the latest intraday trade high price available for the instrument (across all trade types). You can use the `Share.getIDate()` or `Share.getIDateNum()` methods to check the date of the latest intraday data.

See Also

`Share.getITradeLow()`, `Share.getITMidHigh()`, `Share.getIMidLow()`

Share.getITradeLow() ShareScript v1.3

get the intraday trade low price for the instrument

Synopsis

share.`getITradeLow()`

Returns

A number giving the latest intraday trade low price for the instrument in the minor currency unit. `Undefined` can be returned if a price is not available.

Description

`getITradeLow()` is a `Share` object method that returns the latest intraday trade low price available for the instrument (across all trades types). You can use the `Share.getIDate()` or `Share.getIDateNum()` methods to check the date of the latest intraday data.

See Also

`Share.getITradeHigh()`, `Share.getIMidHigh()`, `Share.getIMidLow()`

Share.getListing() ShareScript v1.3

get the instrument's LSE listing (Full or AIM)

Synopsis

share.`getListing()`

Returns

A string which gives the listing for the instrument (e.g. "AIM" or "Full"). This method is only useful for LSE-listed instruments. You can use the `getMarket()` method to return an instrument's exchange.

See Also

`Share.getMarket()`

Share.getLow()

get a single day's low price for the instrument

Synopsis

share.`getLow()`
share.`getLow(daysAgo)`

Arguments

`DaysAgo`	An optional integer specifying the number of trading days ago that you want to get the day's low price for (0 is the most recent low, 1 is yesterday, etc).

Returns

A number giving the low price of the instrument in the minor currency unit. `Undefined` can be returned if a price is not available (e.g. the requested day lies before the start of the instrument's price history).

Description

`getLow()` is a `Share` object method that returns the requested day's low price. If the `daysAgo` parameter is not specified, `getLow()` returns the most recent day's low. If you require a full OHLCV record for an instrument, consider using `Share.getPrice()` instead.

See Also

`Share.getLowOnDate()`, `Share.getClose()`, `Share.getOpen()`, `Share.getHigh()`, `Share.getVolume()`, `Share.getPrice()`, `Share.getCurrency()`

Share.getLowOnDate()

get a single day's low price for the instrument on a specific date

Synopsis

share.`getLowOnDate(date)`

Arguments

`date`	A JavaScript `Date` object (or alternatively a dateNum) specifying the date on which to return the day's low price.

Returns

A number giving the low price of the instrument in the minor currency unit. `Undefined` can be returned if the requested date lies outside of the instrument's history.

Description

`getLowOnDate()` is a `Share` object method that returns the requested day's low price for an instrument. If `date` falls on a holiday or weekend, then the price on the preceding trading day will be returned.

See Also

`Share.getLow()`, `Share.getCloseOnDate()`, `Share.getOpenOnDate()`, `Share.getHighOnDate()`, `Share.getVolumeOnDate()`, `Share.getPriceOnDate()`, `Share.getCurrency()`

Share.getMarket()

get ShareScope's exchange (or market) code for the instrument

Synopsis

share.`getMarket()`

Returns

A string which gives the Exchange code of the instrument (e.g. "LSE").

See Also

`Share.getName()`, `Share.getEPIC()`

Share.getMarketOpenTime() ShareScript v1.1

get the opening time for the instrument's exchange

Synopsis

share`.getMarketOpenTime()`

Returns

The normal market (exchange) opening time as a timeNum.

See Also

`timeNum`, `Share.getMarketCloseTime()`, `Share.getMarketOffsetGMT()`

Share.getMarketCloseTime() ShareScript v1.1

get the closing time for the instrument's exchange

Synopsis

share`.getMarketCloseTime()`

Returns

The normal market (exchange) closing time as a timeNum.

See Also

`timeNum`, `Share.getMarketOpenTime()`, `Share.getMarketOffsetGMT()`

Share.getMarketOffsetGMT() ShareScript v1.1

get the difference between an instrument's time-zone and GMT

Synopsis

share`.getMarketOffsetGMT()`
share`.getMarketOffsetGMT(date)`

Arguments

`date`	A JavaScript `Date` object specifying the date on which to return the offset. If no date is specified, today is used.

Returns

The difference (in seconds) between the share objects's time-zone and GMT.

Description

The value returned by this method is an number that can be added to the timeNum values obtained from `PriceData`, `TradeData` or `BidOfferData` objects in order to convert them to GMT.

e.g. during British Summer Time, this function will return –3600 for shares listed on the London Stock Exchange.

See Also

`timeNum`, `PriceData`, `BidOfferData`, `TradeData`

Share.getMonthlyBarArray() ShareScript v1.2

get an array of monthly OHLCV price records for the instrument

Synopsis

share.getMonthlyBarArray()
share.getMonthlyBarArray(num)

Arguments

`num`	An optional integer specifying the number of monthly bars required. e.g. 10 will return the most recent 10 monthly bars.

Returns

An array of `PriceData` objects. The oldest record is first (element 0). If more bars are requested than available in the price history, the length of the array may be shorter than `num`.

Description

`getMonthlyBarArray()` is a `Share` object method that returns monthly bars (OHLCV records) from the history. If the optional `num` parameter is not used, the entire price history is returned as monthly bars. The oldest bar is at array[0]. The most recent bar is at array[array.length-1].

Monthly bars are based on calendar months. The most recent bar contains as many days as have elapsed in the current calendar month.

If you require daily bars, use `getPriceArray()` instead. For weekly bars, use `getWeeklyBarArray()`. For intraday bars (e.g. 5 minute bars), use `getIBarArray()`.

See Also

`PriceData`, `Share.getPriceArray()`, `Share.getWeeklyBarArray()`, `Share.getIBarArray()`, `Share.getCurrency()`

Share.getName()

get the full name of a company or instrument

Synopsis

share.getName()

Returns

A string which gives the name of the instrument, or the name of the company, that a `share` object belongs to (e.g. “FTSE 100” or “Lloyds TSB Plc”).

See Also

`Share.getShareName()`, `Share.getEPIC()`

Share.getNotes() ShareScript v1.1

get the "notes column" values for the instrument

Synopsis

share.getNotes()

Returns

An array of strings containing the value of the notes columns 1-10. The first element of the returned array corresponds to Note 1, the last element to Note 10. An element will be `undefined` if the note has not been set.

Share.getNMS()

get the normal market size (NMS) for the instrument

Synopsis

share.getNMS()

Returns

A number giving the normal market size for the instrument (the unit is shares). This information is only available for a company's primary share. Note that this method now returns the EMS, since NMS is so longer used – new code should use `Share.getEMS()` instead.

See Also

`Share.getAssocShares()`, `Share.getEMS()`

Share.getNumShares()

get the number of shares

Synopsis

share.getNumShares()

Returns

The number of shares issued (in Millions).

See Also

`Share.getCap()`

Share.getOpen()

get a single opening price for the instrument

Synopsis

share.getOpen()
share.getOpen(daysAgo)

Arguments

daysAgo	An optional integer specifying the number of trading days ago that you want to get an opening price for (0 is the most recent open, 1 is yesterday, etc).

Returns

A number giving the opening price of the instrument in the minor currency unit. `Undefined` can be returned if a price is not available (e.g. the requested day lies before the start of the instrument's price history).

Description

`getOpen()` is a `Share` object method that returns the requested day's opening price. If the `daysAgo` parameter is not specified, `getOpen()` returns the most recent open. If you require a full OHLCV record, consider using `Share.getPrice()` instead.

See Also

`Share.getOpenOnDate()`, `Share.getClose()`, `Share.getHigh()`, `Share.getLow()`, `Share.getVolume()`, `Share.getPrice()`, `Share.getCurrency()`

Share.getOpenOnDate()

get a single opening price for the instrument on a specific date

Synopsis

share.`getOpenOnDate(date)`

Arguments

`date`	A JavaScript `Date` object (or alternatively a dateNum) specifying the date on which to return the opening price.

Returns

A number giving the opening price of the instrument in the minor currency unit. `Undefined` can be returned if the requested date lies outside of the instrument's history.

Description

`getOpenOnDate()` is a `Share` object method that returns the requested day's opening price. If `date` falls on a holiday or weekend, then the price on the preceding trading day will be returned.

See Also

`Share.getOpen()`, `Share.getCloseOnDate()`, `Share.getHighOnDate()`, `Share.getLowOnDate()`, `Share.getVolumeOnDate()`, `Share.getPriceOnDate()`, `Share.getCurrency()`

Share.getPrice()

get a single OHLCV price record for the instrument

Synopsis

share.`getPrice()`
share.`getPrice(daysAgo)`

Arguments

`daysAgo`	An optional integer specifying the number of trading days ago that you want to get a price for (0 is the most recent price, 1 is yesterday, etc).

Returns

A `PriceData` object. The fields of this object can be `undefined` if a price record is requested that lies before the start of the instrument's history.

Description

`getPrice()` is a `Share` object method that returns a single OHCLV `PriceData` record. If the `daysAgo` parameter is not specified, `getPrice()` returns the most recent price record.

If you require OHLCV data across many days, consider using `Share.getPriceArray()` instead.

Example

The following example gets the last price date (as a JavaScript Date object) for a share:

```
var lastdate = my_share.getPrice().date
```

See Also

`PriceData`, `Share.getPriceOnDate()`, `Share.getPriceArray()`, `Share.getPriceArrayDates()`, `Share.getOpen()`, `Share.getHigh()`, `Share.getLow()`, `Share.getClose()`, `Share.getVolume()`

Share.getPriceArray()

get an array of OHLCV price records for the instrument

Synopsis

share.getPriceArray()
share.getPriceArray(num)

Arguments

`num`	An optional integer specifying the number of prices required. e.g. 10 will return the most recent 10 prices.

Returns

An array of `PriceData` objects. The oldest record is first (element 0). If more days are requested than available in the price history, the length of the array may be shorter than `num`.

Description

`getPriceArray()` is a `Share` object method that returns multiple OHLCV records from the history. If the optional `num` parameter is not used, the entire price history is returned. The oldest price is at array[0]. The most recent price will be at array[array.length-1].

See Also

`PriceData`, `Share.getCloseArray()`, `Share.getPriceArrayDates()`, `Share.getPrice()`, `Share.getPriceOnDate()`, `Share.getCurrency()`

Share.getPriceArrayDates()

get an array of OHLCV price records for the instrument between two dates

Synopsis

```
getPriceArrayDates()
getPriceArrayDates(start)
getPriceArrayDates(start, end)
```

Arguments

`start`	An optional JavaScript `Date` object specifying the date on which to start returning prices. If not specified, the start of the history is used.
`end`	An optional JavaScript `Date` object specifying the date on which to stop returning prices. If not specified, the end of the history is used.

Returns

An array of `PriceData` objects for each of the days between the `start` and `end` dates (inclusive). The oldest record is first (array element 0). Records will not be returned for dates on which trading did not take place (e.g. weekends and holidays). The length of the array may be zero if no records match the criteria.

Description

`getPriceArrayDates()` is a `Share` object method that returns multiple OHLCV records from the history. If the `end` date is not specifed, the end of the history is assumed. If the `start` date is not specified, the start of the history is assumed. The oldest price is at array[0]. The most recent price will be at array[array.length-1].

See Also

`PriceData`, `Share.getCloseArrayDates()`, `Share.getPriceArray()`, `Share.getPrice()`, `Share.getPriceOnDate()`, `Share.getCurrency()`

Share.getPriceOnDate()

get a single OHLCV price record for the instrument on a specific date

Synopsis

share.`getPriceOnDate(date)`

Arguments

`date`	A JavaScript `Date` object (or alternatively a dateNum) specifying the date to return the price for.

Returns

A `PriceData` object. The fields of this object can be `undefined` if the requested date lies outside of the instrument's history.

Description

`getPriceOnDate()` is a `Share` object method that returns a single OHCLV `PriceData` record from an instrument's price history. If `date` falls on a holiday or weekend, then the price on the preceding trading day will be returned.

If you require OHLCV data across many days, consider using `Share.getPriceArray()` instead.

See Also

`PriceData`, `Share.getPrice()`, `Share.getPriceArray()`, `Share.getPriceArrayDates()`, `Share.getCloseOnDate()`

Share.getResult()

provides basic access to the company's historical and forecast results

Synopsis

share.getResult(year, result)

Arguments

year	A number giving the year for which results are desired. A value <0 will access historic results, 0 the current year, >0 for forecasts.
result	A value specifying the result required (e.g. profit). Possible values are defined as static constants by the Result object.

Returns

The requested result, or undefined if that kind of result was not available for the requested year. The return type may be either a number, a date or a string, depending on the kind of result requested. See Result for details.

Description

getResult is a Share object method that gets a specific result (e.g. profit) for a given year. This may be a historical result, an announced result, or a forecast. getResult() always returns year-end figures. If you require quarterly or interim results (or special dividends), use the Share.getResultArray() function instead.

A negative value for year will access historical results (e.g. –1 is one year ago). Positive values access forecasts. Year 0 is the most recent year for which the year end results are available (whether or not they have been analysed and normalised).

The type of the most recent results will change from "Announced" to "Final" after they have been normalised. You can access this result type using getResult(…, Result.Type).

The possible kinds of results that can be requested are defined as static constants by the Result object.

Example

```
var profit = my_share.getResult(0, Result.Profit);
```

See Also

Result, Share.getResultArray(), Share.getCurrencyR()

Share.getResultArray()

provides advanced access to the company's historical and forecast results

Synopsis

share.getResultArray(year, result)

Arguments

year	A number giving the year for which results are desired. A value <0 will access historic results, 0 the current year, >0 for forecasts.
result	A value specifying the result required (e.g. profit). Possible values are defined as static constants by the Result object. Note that only a sub-set of these constants are valid for getResultArray().

Returns

An array providing all values of a given `result` for the specified year. The array can be empty if no results are available for the requested year. The values in the array are cumulative (but see below), and the earliest result will be the first element of the array. An element of the array may be `undefined` to indicate that a value is not relevant or unavailable.

Description

`getResultArray()` is a `Share` object method that can be used to access quarterly, interim and final/forecast results (and any special dividend payments) for a year. The number of elements in the returned array is variable, depending on whether the company reports bi-annually or quarterly (and if any special dividend payments have been made).

For information about the `year` parameter, see `getResult()`.

The possible kinds of results that can be requested are defined as static constants by the `Result` object. Note that only a sub-set of these constants are valid for `getResultArray()`.

Note that the array will always be the same length for a given year, no matter what kind of result is requested. This allows you to call this function multiple times, first to identify the type of result, and subsequently to retrieve the value of specific items. It is probably easiest to think of each call to `getResultArray()` as returning a single row of a table, where the columns represent each reported result across a year –

Result Type	Interim	Special	Final
Profit (£m)	19.54		43.56
EPS (p)	15.60		34.81
Dividend (p)	4.85	1.00	20.15
Ex-div date	29/11/06	15/3/06	6/6/07

Where a numerical result is requested (such as profit or dividend), the values in the array will be cumulative, with the final value corresponding to the total for the year (as returned by the `getResult()` function). An exception to this is the "Special" result type, which is a self-contained record of a special dividend, and is not included in the cumulative total.

Example

`my_share.getResultArray(0, Result.Type) → ["Interim", "Special", "Final"]`

See Also

`Result`, `Share.getResult()`, `Share.getCurrencyR()`

Share.getRiskAnalysis() ShareScript v1.31

get risk analysis metrics for the instrument

Synopsis

share`.getRiskAnalysis(type, config)`

Arguments

`type`	A value specifying the risk analysis required (e.g. alpha). Possible values are defined as static constants on the `RA` object.
`config`	A config object specifying values to configure the risk analysis, or an empty object { } to accept the defaults. Available properties are:

`period`	Integer. The number of return periods (the default is 12). 0 means all data.
`periodType`	Constant. The period type. Possible values are defined by the `RA` object (the default is `RA.Quarterly`).
`periodPrice`	Constant. The price to use from the period. Possible values are defined by the `RA` object (the default is `RA.LastPrice`).
`annualise`	Boolean. Whether to annualise the result (the default is `false`).
`riskFreeRate`	Number. The risk free rate % (the default is 0.5).
`benchmark`	`Share` object. The benchmark (the default is the FTSE100 index).
`sliding`	Boolean (the default is `true`).
`minPeriod`	Number. Minimum periods. Default is 0 (disabled).

Returns

A number giving the value of the metric requested, or `undefined` if the analysis cannot be performed because of insufficient data.

Throws

`RangeError`	If any of the arguments are out of range, or config options are incorrect for the selected analysis type.

Description

`getRiskAnalysis()` is a `Share` object method that provides access to ShareScope's risk analysis metrics (available from the Add Risk Analysis Column… dialog). Because the set of available parameters varies across the analysis types, this method takes a "config" object rather than the more usual approach of multiple parameters separated by commas.

This config object is just a plain JavaScript object to which you add property names and values if you want to specify a non-default setting. See the example below for an illustration. This leads to more readable scripts, and makes it easy to leave something set to its default value.

Note that passing `undefined` as a benchmark to an analysis type requiring a benchmark will not cause the script to fail but simply causes the risk analysis to return `undefined`. This allows you to use `{benchmark: share.getSectorIndex()}` safely without needing to catch exceptions.

Example

```
var alpha = share.getRiskAnalysis(RA.Alpha, {
        period: 20,
        periodType: RA.Monthly,
        benchmark: share.getSectorIndex()
});
```

See Also

`Share.getResult()`

Share.getSector()

get the sector of the company

Synopsis

share.getSector()

Returns

A string which gives the sector that a company is engaged in (e.g. “Real Estate”).

See Also

Share.getIndustry(), Share.getSubSector(), Share.getSuperSector(), Share.getSectorIndex()

Share.getSectorIndex()

get a Share object corresponding to the sector index for the company

Synopsis

share.getSectorIndex()

Returns

A Share object corresponding to the company’s sector index, or undefined if there is no sector index available.

Description

getSectorIndex() is a Share object method that returns (if available) a Share object corresponding to the sector index for the instrument. For UK Equities, this will normally be either as FTSE 350 sector index or All-share sector index. For Unit Trusts, this will be the IMA sector index (this was added in ShareScript v1.31).

See Also

getShare(), Share.getAssocShares(), getList(), getPortfolio()

Share.getSEDOL() **ShareScript v1.3**

get the SEDOL for an instrument

Synopsis

share.getSEDOL()

Returns

A string which gives the instrument’s SEDOL.

See Also

Share.getEPIC(), Share.getName(), Share.getShareScopeID(), Share.getISIN()

Share.getShareName()

get the name of the instrument

Synopsis

share.getShareName()

Returns

A string which gives the name of this particular instrument belonging to a company (e.g. “25p ords”), or undefined if this is not available.

See Also

Share.getName()

Share.getShareNum() ShareScript v1.1

get the share number for the instrument

Synopsis

share.getShareNum()

Returns

The share number.

Description

A company may have one or more different shares. This function returns a number that uniquely identifies a particular share for a given company. If the Share object belongs to a company with only one share (or if the Share object represents e.g. an index) then this number will be 0.

Note that the value returned by this method corresponds to an index into the array returned by Share.getAssocShares().

For more information, see the description of the Share.getShareScopeID() function below.

See Also

Share.getShareScopeID(), getShare(), Share.getAssocShares()

Share.getShareScopeID() ShareScript v1.1

get the ShareScope ID for the instrument

Synopsis

share.getShareScopeID()

Returns

A number giving the ShareScope ID of the instrument.

Description

The ShareScope ID identifies an instrument or the company to which an instrument belongs.

Note that the ShareScope ID may not by itself uniquely identify an instrument, since a company may have more than one type of share (see the getAssocShares() function).

To uniquely identify an instrument, you must also obtain the share number for the instrument using `Share.getShareNum()`. Together, the ShareScopeID and the ShareNum uniquely identify an instrument.

See Also

`Share.getShareNum(), getShare(), Share.getAssocShares()`

Share.getSubSector()

get the sub-sector of the company

Synopsis

share`.getSubSector()`

Returns

A string which gives the sub-sector that the company is engaged in (e.g. "Real Estate Holding & Development").

See Also

`Share.getIndustry()`, `Share.getSector()`, `Share.getSuperSector()`, `Share.getSectorIndex()`

Share.getSuperSector() **ShareScript v1.3**

get the super-sector of the company

Synopsis

share`.getSuperSector()`

Returns

A string which gives the super-sector that the company is engaged in.

See Also

`Share.getIndustry()`, `Share.getSector()`, `Share.getSubSector()`, `Share.getSectorIndex()`

Share.getTradingSystem() **ShareScript v1.3**

get the LSE's trading system for the instrument

Synopsis

share`.getTradingSystem()`

Returns

A string which gives the LSE's trading system for the instrument (e.g. "SETSqx"). This method is only useful for LSE-listed instruments. You can use the `getMarket()` method to return an instrument's exchange.

See Also

`Share.getMarket()`

Share.getType()

get the type of the instrument

Synopsis

share.getType()

Returns

A string which gives the type of the instrument (e.g. “Ord”, “Index”).

See Also

Share.getShareName()

Share.getUncrossingPrice() ShareScript v1.34

get the latest indicative uncrossing price for the instrument

Synopsis

share.getUncrossingPrice()

Returns

If the instrument is in intraday auction, this method returns a number giving the latest intraday auction indicative uncrossing price. The value is in the minor currency unit.

Undefined will be returned if there is no uncrossing price or the share is not currently in intraday auction.

When a share is in auction, ShareScope stops updating the bid/offer values and will only update the uncrossing price. The bid/offer and mid values returned (e.g. by the Share.getIMid() method) will be the values immediately prior to the start of the auction.

See Also

Share.isInAuction(), Share.getIMid(), Share.getIBidOfferArray()

Share.getVolume()

get a single day’s volume for the instrument

Synopsis

share.getVolume()
share.getVolume(daysAgo)

Arguments

daysAgo	An optional integer specifying the number of trading days ago that you want to get the volume figure for (0 is the most recent volume, 1 is yesterday, etc).

Returns

A number giving the day’s volume for the instrument on the requested day. Undefined can be returned if no volume is available (e.g. the requested day lies before the start of the instrument’s price history).

Description

getVolume() is a Share object method that returns the requested day's volume. If the daysAgo parameter is not specified, getVolume() returns the most recent day's volume. If you require a full OHLCV record for an instrument, consider using Share.getPrice() instead.

See Also

Share.getVolumeOnDate(), Share.getClose(), Share.getOpen(), Share.getHigh(), Share.getLow(), Share.getPrice()

Share.getVolumeOnDate()

get a single day's volume for the instrument on a specific date

Synopsis

share.getVolumeOnDate(date)

Arguments

date	A JavaScript Date object (or alternatively a dateNum) specifying the date on which to return the day's volume figure.

Returns

A number giving the day's volume for the instrument. Undefined can be returned if the requested date lies outside of the instrument's history.

Description

getVolumeOnDate() is a Share object method that returns the requested day's volume for an instrument. If date falls on a holiday or weekend, then the volume for the preceding trading day will be returned.

See Also

Share.getHigh(), Share.getCloseOnDate(), Share.getOpenOnDate(), Share.getHighOnDate(), Share.getLowOnDate(), Share.getPriceOnDate()

Share.getWeeklyBarArray() ShareScript v1.2

get an array of weekly OHLCV price records for the instrument

Synopsis

share.getWeeklyBarArray()
share.getWeeklyBarArray(num)

Arguments

num	An optional integer specifying the number of weekly bars required. e.g. 10 will return the most recent 10 weekly bars.

Returns

An array of PriceData objects. The oldest record is first (element 0). If more bars are requested than available in the price history, the length of the array may be shorter than num.

Description

`getWeeklyBarArray()` is a `Share` object method that returns weekly bars (OHLCV records) from the history. If the optional `num` parameter is not used, the entire price history is returned as weekly bars. The oldest bar is at array[0]. The most recent bar is at array[array.length-1].

Weekly bars run from Monday to Friday, with the most recent bar containing as many days as have elapsed in the current week.

If you require daily bars, use `getPriceArray()` instead. For monthly bars, use `getMonthlyBarArray()`. For intraday bars (e.g. 5 minute bars), use `getIBarArray()`.

See Also

`PriceData`, `Share.getPriceArray()`, `Share.getMonthlyBarArray()`, `Share.getIBarArray()`, `Share.getCurrency()`

Share.isInAuction() **ShareScript v1.34**

returns whether the share is in an intraday auction.

Synopsis

share`.isInAuction()`

Returns

A boolean value (true or false) that indicates whether the share is in an intraday auction.

When a share is in auction, ShareScope stops updating the bid/offer values and will only update the uncrossing price. The bid/offer and mid values returned (e.g. by the `Share.getIMid()` method) will be the values immediately prior to the start of the auction.

Note that SETSqx instruments will always return false when this method is called, since instruments on this platform are in permanent auction.

See Also

`Share.getUncrossingPrice()`

Share.isSuspended()

returns whether the company's shares are suspended from trading

Synopsis

share`.isSuspended()`

Returns

A boolean value (true or false) that indicates whether the company has its shares suspended from trading.

timeNum() **ShareScript v1.1**

create a ShareScope timeNum

Synopsis

```
timeNum(dateObj)
timeNum(hour, min, sec)
```

Arguments

`dateObj`	A JavaScript `Date` object to be used to create the timeNum.
`hour`	The hour as an integer from 0 to 23.
`min`	The minute as an integer from 0 to 59.
`sec`	The second as an integer from 0 to 59.

Returns

An integer timeNum representing the time.

Throws

`RangeError`	If any of the arguments are out of range.

Description

`timeNum()` is a global function that you can use to create a timeNum value (which is just an integer that compactly represents a time, in seconds since midnight). Scripts using timeNums will be faster than those using JavaScript `Date` objects.

Normally, you will not need to create timeNums yourself, but will obtain them from a `PriceData` , `BidOfferData` and `TradeData` records. You can then use one of the other timeNum functions below to inspect the time.

There is also a set of `dateNum()` functions that can be used to compactly represent the date part of a JavaScript `Date` object.

See Also

`PriceData`, `TradeData`, `BidOfferData`, `dateNum`, `timeNumGetHour()`, `timeNumGetMin()`, `timeNumGetSec()`, `Share.getMarketOpenTime()`, `Share.getMarketCloseTime()`

timeNumGetHour() ShareScript v1.1

return the hour part of a ShareScope timeNum

Synopsis

`timeNumGetHour(n)`

Arguments

`n`	An integer timeNum obtained e.g. from a `PriceData` record or `timeNum()`.

Returns

An integer providing the hour (0-23).

Description

`timeNumGetHour()` is a global function that returns the hour part of a timeNum value. See `timeNum()` for more information about timeNums.

See Also

`timeNum()`, `timeNumGetMin()`, `timeNumGetSec()`

timeNumGetMin() ShareScript v1.1

return the minute part of a ShareScope timeNum

Synopsis

`timeNumGetMin(n)`

Arguments

`n`	An integer timeNum obtained e.g. from a `PriceData` record or `timeNum()`.

Returns

An integer providing the minutes (0-59).

Description

`timeNumGetMin()` is a global function that returns the minute part of a timeNum value. See `timeNum()` for more information about timeNums.

See Also

`timeNum()`, `timeNumGetHour()`, `timeNumGetSec()`

timeNumGetSec() ShareScript v1.1

return the seconds part of a ShareScope timeNum

Synopsis

`timeNumGetSec(n)`

Arguments

`n`	An integer timeNum obtained e.g. from a `PriceData` record or `timeNum()`.

Returns

An integer providing the seconds (0-59).

Description

`timeNumGetSec()` is a global function that returns the seconds part of a timeNum value. See `timeNum()` for more information about timeNums.

See Also

`timeNum()`, `timeNumGetHour()`, `timeNumGetMin()`

TradeData ShareScript v1.1

details an intraday trade Object→TradeData

Synopsis

`TradeData.`*`property`*

Construction

`TradeData` objects are returned in an array by the `getITradeArray()` series of `Share` object methods. They cannot be created using the normal JavaScript `new()` operator.

Properties

`price`	The price at which the trade took place in the minor currency unit (e.g. Pence)
`volume`	The size of the trade (number of shares).
`type`	The type of trade. Possible values are defined as static constants by `TradeType` (see next entry).
`date`	The date/time (a JavaScript `Date` object), or `undefined` if no value is available.
`dateNum`	An integer representation of the date (see below for details).
`timeNum`	An integer representation of the time (seconds since midnight).
`isPlusMarkets`	A boolean value. **ShareScript v1.3** Indicates whether the trade took place on Plus Markets. This field is only used for instruments where the exchange is LSE.
`index`	An integer giving the index of this event within the second (see below for more information). **ShareScript v1.33**

Description

`TradeData` objects represent an intraday trade.

The `dateNum` & `timeNum` fields provides an integer representation of the date and time. This is made available since JavaScript `Date` objects are relatively costly (in terms of execution speed) to use and compare.

Even though the time resolution provided by the `date` and `timeNum` fields is limited to a second, the records are always returned in the correct sequence by `getITradeArray()`. However, if you need to determine to ordering of trades with respect to bid/offers (which are returned separated by the `getIBidOfferArray()` method), then you must use the `index` fields present in both arrays to determine the proper ordering of trades and prices that occur in the same second. This field starts at zero each second and increments with each trade or bid/offer occurring within that second.

See Also

`dateNum()`, `timeNum()`, `Share.getITradeArray()`, `Share.getITradeArrayOnDate()`, `PriceData`, `BidOfferData`

TradeType **ShareScript v1.1**

constants to identify the different types of trades

Synopsis

`TradeType.`*`constant`*

Constants

`Unknown`	An unknown trade type
`Cum`	Cumulative trade e.g. for index volume, or snapshot data. **ShareScript v1.3**
`O`	Ordinary trade immediate publication
`AT`	Automatic trade
`UT`	Uncrossing trade (total volume SETS auction)

`CT`	Contra Trade	
`OK`	Ordinary trade delayed publication	
`NT`	Negotiated trade immediate publication	
`NK`	Negotiated trade delayed publication	
`PC`	Previous day contra trade	
`LC`	Late correction	
`NM`	Not to mark	
`OT`	OTC trade immediate publication	
`TK`	OTC trade delayed publication	
`IF`	Inter fund cross delayed publication	
`OC`	OTC trade late correction	
`SI`	SI trade immediate publication	
`SK`	SI trade delayed publication	
`SC`	SI trade late correction	
`OM`	Market maker to market maker (Plus markets only)	**ShareScript v1.3**
`OX`	Crossed (Plus markets only)	**ShareScript v1.3**
`OR`	Riskless principal (Plus markets only)	**ShareScript v1.3**
`OP`	Portfolio (Plus markets only)	**ShareScript v1.3**
`NX`	Negiotiated crossed (Plus markets only)	**ShareScript v1.3**
`NB`	Negiotiated broker to broker (Plus markets only)	**ShareScript v1.3**
`NR`	Negiotiated riskless principal (Plus markets only)	**ShareScript v1.3**
`NP`	Negiotiated portfolio (Plus markets only)	**ShareScript v1.3**
`L`	Large in scale (Plus markets only)	**ShareScript v1.3**
`I`	Protected trade (Plus markets only)	**ShareScript v1.3**
`T`	Protected trade confirmation (Plus markets only)	**ShareScript v1.3**
`Z`	Off market (Plus markets only)	**ShareScript v1.3**

Description

`TradeType` is a global object that defines constants that refer to the various types of trades. They can be used to identify the type of trade in a `TradeData` object.

The majority of these codes apply to LSE/Plus Markets data only.

US market trades are always listed as Ordinary trades (`TradeType.O`), or Cumulative trades (`TradeType.Cum`) if, for example, a 15 second snapshot feed is used.

See Also

`TradeData`, `Share.getITradeArray()`, `Share.getITradeArrayOnDate()`

Column Objects Reference

Column

ShareScope column interface Object→Column

Construction

ShareScript `Column` objects are created when the user adds a ShareScript column to a list screen, or creates a ShareScript alarm. They are not created using the normal JavaScript `new()` operator.

Methods

A `Column` object has a few methods you can call (e.g. the `setTitle()` method) and two methods you can supply to create a new ShareScript column (the `init()` and `getVal()` methods).

The `load()` method allows you to make library functions available within the `Column` object, rather than defining them in the global object. This will prevent namespace collisions when your column uses functions defined externally.

`init()`	Optional. ShareScope will call a `Column` object's `init()` method once, before any call to `getVal()`.	
`getVal()`	ShareScope calls a `Column` object's `getVal()` method when it needs to get the value of the column for an instrument. The value returned by this method becomes the value of the column.	
`load()`	Load and execute a ShareScript file in this column object. See the global `load()` method for full details.	
`setTitle()`	Set the column heading.	**ShareScript v1.1**
`setValueForShare()`	Store a value keyed on the share.	**ShareScript v1.3**
`getValueForShare()`	Retrieve a value keyed on the share.	**ShareScript v1.3**

Constants

These constants are defined on the `column` object, and can be used for comparison with the status parameter passed to the `init()` function (see the `init()` method description for details):

`Loading`	ShareScope has started up, and is loading the column from the user configuration files.
`Adding`	The user has added a new column.
`Editing`	The user has indicated they wish to edit the column.

Properties

`storage`	The column's `Storage` object, which provides a permanent storage mechanism for data. See section 5 of this reference for details.	**ShareScript v1.1**
`isAlarmContext`	Indicates whether the column is being used as an alarm (true) or a normal column (false).	**ShareScript v1.3**

Description

When you create a script for a ShareScript column, you need to specify a `getVal()` function, and (optionally) an `init()` function. These, and any other functions or variables you define in the file will become properties of a `Column` object.

The value returned by `getVal()` becomes the value of the column for an instrument. See the reference entry for `Column.getVal()` for more details.

Note that the `Column` object is at the start of the scope chain when executing the code in your column script. Thus a call to e.g. the `load()` method will call the current `Column` object's `load()` method, rather than the `load()` method in the global object.

Any functions you call that are not defined by your `Column` object will be resolved in the next object in the scope chain (i.e. the global object).

Column scripts should be placed in your `ShareScript/Columns/` directory.

Directives

Column directives allow you to tell ShareScope how to treat the column created by your script. They are detailed here for completeness, although they are not technically part of the ShareScript language (directives are placed inside comments, and are used by ShareScope itself, rather than the ShareScript interpreter). Directives take the form:

`@Field:Value`

The available fields and the values they can take are detailed below. Note that field names and values are not case sensitive. Normally, you should place any directives you wish to use in a comment, near the top of your ShareScript file.

Field	Valid values/description
`@Name`	Any text. ShareScope will use this value as the name of the column. This will appear at the top of your column, and in the Add ShareScript Column dialog. If not specified, ShareScope will use the filename of your column script. e.g. `//@Name:My Column`
`@Description`	Any text. A description of what your Script does, which is displayed in the Add ShareScript Column dialog. e.g. `//@Description:The total return over 3 years.`
`@Returns`	Valid values are `Text` or `Number`. Tells ShareScope what kind of values your Script will generate. This will change how the values are sorted, and what options are presented to the user. The default is `Number`. Note that only columns returning a number can be used as Data Mining criteria. e.g. `//@Returns:Text`
`@Width`	A number. Tells ShareScope how wide the column should be (in pixels). e.g. `//@Width:50`

Field	Valid values/description
@Update	Valid values are Normal, Intraday or Periodic. **ShareScript v1.1** **Normal** updating mean the column values will only be re-evaluated when the historical price/fundamentals database is updated. **Intraday** updating will cause a column value to be re-evaluated whenever new intraday data arrives for the share. **Periodic** updating will cause the values to be updated every minute (by default) when you are connected to the intraday feed. You can also specify a different update period in seconds (with a minimum of 15). e.g. `@Update:Periodic,30`
@Env	Valid values are Development or Production **ShareScript v1.1** This directive modifies the environment ShareScope provides to the user to interact with the column. The default is Development. When Production is selected – (i) The "Refresh Script" command is removed from the menu (ii) "Edit column…" will not allow the user to select a different script. It will instead call the Column's init() function with Editing status.
@Editable	Valid values are Yes or No. **ShareScript v1.1** Applicable only with Production environment. This directive tells ShareScope whether to display the "Edit column…" option. The default is Yes.
@DefaultRangeMax	A number. **ShareScript v1.3** When a numeric column is used as an alarm, this value is used to specify the default alarm trigger upper threshold.
@DefaultRangeMin	A number. **ShareScript v1.3** When a numeric column is used as an alarm, this value is used to specify the default alarm trigger lower threshold.
@StandardAlarmOutput	Valid values are Yes or No. **ShareScript v1.3** When a numeric column is used as an alarm, Yes tells ShareScope not to prompt the user for thresholds since the column will output 0 (or undefined) to indicate the alarm has not triggered, and any non-zero value to indicate a trigger condition. The default is No.

Column.init()

method invoked when a column is created

Synopsis

`function init(status)`

Arguments

status	A parameter passed (by ShareScope) telling you why ShareScope is initialising the column. This should be compared to the constants defined on the Column object (e.g. Adding).

Returns

You should return a value of `false` if you don't want the column to be added (or replaced). See below for details.

Description

ShareScope invokes a column's `init()` method when a `Column` object is created by ShareScope. It is guaranteed to be called before any call to `getVal()`. You do not need to supply an `init()` method if you do not need one.

The `init()` method (and its return value) is handled by ShareScope as follows:

When the user **adds a column** for the first time – ShareScope creates a new `Column` object, then calls `init()` with the status set to `Adding`. If `init()` returns `false`, the column is not added, and the `Column` object is discarded.

When the user **edits a column** – ShareScope will create a new `Column` object, and copy any data in the original column's `storage` area to the new object. It then calls the `init()` function of the new `Column` object with the status set to `Editing`. If the `init()` function returns `false`, the new column is discarded, and the user's list table or Data Mining filter is left unchanged.

When ShareScope **starts up**, it will create a `Column` object for any ShareScript columns in the user's list tables (or DM filters). It will then call the `init()` function with a status of `Loading`. If the `init()` function returns `false`, the column will be disabled (but will still be visible to the user).

Finally, note that the "Refresh script" command (available on a column's context menu) creates a new `column` object and calls `init()` with a status of `Adding`. If `init()` returns `false`, the column is disabled.

Example

This simple (but complete) example shows you how to structure your `init()` function when you allow a user to specify a parameter for the column. The user is prompted for a number of days ago, and the column will then display the close price for that date:

```
//@Name:Example

var daysAgo = 5;

function init(status)
{
    if (status == Loading || status == Editing)
        daysAgo = storage.getAt(0);

    if (status == Adding || status == Editing)
    {
        var dlg = new Dialog("Example Column", 200, 45);
        dlg.addOkButton();
        dlg.addCancelButton();
        dlg.addIntEdit("days", -1,-1,-1,-1, "", "trading days", daysAgo, 0, 250);
        if (dlg.show()==Dialog.Cancel)
            return false;
        daysAgo = dlg.getValue("days");
        storage.setAt(0, daysAgo);
    }
    setTitle("Close " + daysAgo + " days ago");
}

function getVal(share)
{
    return share.getClose(daysAgo);
}
```

Column.getVal()

method invoked when ShareScope needs the column value for an instrument

Synopsis

`function getVal(shareObj)`

Arguments

`shareObj`	The `Share` object that ShareScope needs a column value for.

Description

ShareScope invokes a Column's `getVal()` method when it needs a value for the column for an instrument.

The return value of the function will become the value of the column for that instrument. ShareScope will convert the returned value to a number (the default), or a string using the standard JavaScript type conversions. Use a `@Returns` directive in your script to specify which conversion should be used.

In some cases you may not want to return a value (if a value cannot be computed for a particular instrument). In this case, ShareScope will treat the value as being not available, and will display and sort it accordingly. You can also return the `undefined` JavaScript value for this purpose.

When a column returns a string and is used as an alarm, ShareScope checks the first character of the returned string to see if it is ">" or "<". If so, this character is not displayed to the user and is instead used to set the alert colour. The character ">" is used to choose the "Price Up" alert colour, and "<" is used to choose the "Price Down" alert colour.

Example

This simple example returns an instrument's name as the value of the column. Note the @Returns directive which tells ShareScope to treat the return value as a string, not to convert it to a number.

```
//@Returns:Text
function getVal(instr)
{
      return instr.getName();
}
```

Column.setTitle() ShareScript v1.1

sets the column heading

Synopsis

`setTitle(s)`

Arguments

`s`	A string providing a heading for the column

Description

The `setTitle()` function allows you to set the column's heading. You should normally call this method from the `init()` function.

Column.getValueForShare() ShareScript v1.3

gets a user-defined value for an instrument

Synopsis

```
getValueForShare(shareObj)
```

Arguments

`shareObj`	A `share` object

Returns

The value that was set for the instrument, or `undefined` if no value has been set.

Description

The `getValueForShare()` method returns the value for a given instrument. For more information, see the description for the corresponding `setValueForShare()` method below.

See Also

`Column.setValueForShare()`

Column.setValueForShare() ShareScript v1.3

sets a user-defined value for an instrument

Synopsis

```
setValueForShare(shareObj, value)
```

Arguments

`shareObj`	A `share` object
`value`	Any javascript type

Description

Column objects provide a mechanism to set and subsequently retrieve arbitrary user-defined values on a per-instrument basis, using the `getValueForShare()` and `setValueForShare()` column object methods.

Any JavaScript data type can be stored against an instrument, from simple boolean values to objects and arrays.

This mechanism can be useful when you want to store the results of calculations on a per-share basis, especially for columns that are updated frequently. There is an example of this technique in the Alarms tutorial of the ShareScript Guide.

Unlike a column's `storage` area, any values stored by `setValueForShare()` are temporary, and will be lost when the column object is destroyed (e.g. when you quit ShareScope, or edit a column).

These methods are provided mainly for convenience, since internally `setValueForShare`() is simply implemented by adding properties to a plain JavaScript object, where the name of each property is string uniquely identifying the instrument.

Example

```
setValueForShare(shareObj, "hello");
getValueForShare(shareObj); // returns “hello”
```

See Also

`Column.getValueForShare()`

Indicator Objects Reference

Indicator

ShareScope indicator interface Object→Indicator

Construction

Because `Indicator` objects represent ShareScript indicators added to graphs by the user, they are not created using the normal JavaScript `new()` operator.

Methods

An `Indicator` object has several methods you can call and two methods you can supply to create a new ShareScript indicator (the `init()` and `getGraph()` methods).

The `load()` method allows you to make library functions available within the `Indicator` object, rather than defining them in the global object. This will prevent namespace collisions when your indicator uses functions defined externally.

`init()`	Optional. ShareScope will call an `Indicator` object's `init()` method once, before any call to `getGraph()`.
`getGraph()`	ShareScope calls an `Indicator` object's `getGraph()` method when it needs to get the indicator data an instrument. The array(s) of values returned by this method is plotted as one or more indicator data series.
`load()`	Load and execute a ShareScript file in this indicator object. See the global `load()` method for full details.
`clearHorizontalLines()`	Remove all added horizontal lines from the indicator.
`getBackColour()`	Get the background colour of the indicator window.
`getBarLength()`	Returns the current bar length. **ShareScript v1.35**
`setHorizontalLine()`	Draw a horizontal line at a given y-axis level.
`setLayer()`	Determines whether a main graph indicator is drawn above or below the main chart. **ShareScript v1.31**
`setRange()`	Set the y-axis range of the indicator window.
`setSeriesChartType()`	Set the chart type for a data series.
`setSeriesColour()`	Set the colour for plotting a data series.
`setSeriesColourMode()`	Set the colour mode for a data series.
`setSeriesLineStyle()`	Set the line style for a data series.
`setTitle()`	Set the indicator title.

Constants

These constants are defined on the indicator object, and can be used for comparison with the status parameter passed to the `init()` function (see the `init()` method description for details):

`Loading`	ShareScope has started up, and is loading the indicator from the user configuration files.
`Adding`	The user has added a new indicator.

`Loading`	ShareScope has started up, and is loading the indicator from the user configuration files.
`Editing`	The user has indicated they wish to edit the indicator.

Properties

`storage`	The indicator's `Storage` object, which provides a permanent storage mechanism for data. See section 5 of this reference for details. **ShareScript v1.1**
`isIntraday`	A boolean that is `true` if your indicator is on an intraday chart, and `false` if on a historical chart. **ShareScript v1.1**
`ChartType`	Constants for setting the chart type for a data series, for use with the `setSeriesChartType()` function.
`ColourMode`	Constants for setting the colour mode of a data series, for use with the `setSeriesColourMode()` function.
`Layer`	Constants for setting the drawing layer. Only applies to main graph indicators. **ShareScript v1.31**
`Range`	Constants for setting the indicator y-axis range, for use with the `setRange()` function.

Description

When you create a script for a ShareScript indicator, you need to specify a `getGraph()` function, and (optionally) an `init()` function. These, and any other functions or variables you define in the file will become properties of an `Indicator` object.

The values returned by `getGraph()` are plotted as one or more indicator data series for the instrument. See the reference entry for `Indicator.getGraph()` for full details.

Note that the `Indicator` object is at the start of the scope chain when executing the code in your indicator script. Thus a call to e.g. the `load()` method will call the current `Indicator` object's `load()` method, rather than the `load()` method in the global object.

Any functions you call that are not defined by your `Indicator` object will be resolved in the next object in the scope chain (i.e. the global object).

Indicator scripts should be placed in your `ShareScript/Indicators/` directory.

Directives

Indicator directives allow you to tell ShareScope how to treat the indicator created by your script. They are detailed here for completeness, although they are not technically part of the ShareScript language (directives are placed inside comments, and are used by ShareScope itself, rather than the ShareScript interpreter). Directives take the form:

```
@Field:Value
```

The available fields and the values they can take are detailed below. Note that field names and values are not case sensitive. Normally, you should place any directives you wish to use in a comment, near the top of your ShareScript file.

`Field`	Valid values/description

`@Name`	Any text. ShareScope will use this value as the name of the indicator. This will appear as the indicator title, and in the Add ShareScript Indicator dialog. e.g. `//@Name:My Indicator`
`@Description`	Any text. A description of what your Script does, which is displayed in the Add ShareScript Indicator dialog. e.g. `//@Description:Custom MACD indicator`
`@Future`	Valid values are `Yes` or `No`. Tells ShareScope that your indicator `getGraph()` function will return data longer than the data it was provided (i.e. your indicator projects into the future). The default is `No`. e.g. `//@Future:Yes`
`@Type`	Valid values are `Historical`, `Intraday` or `Both`. **ShareScript v1.1** Restricts the use of your indicator to the selected chart type. The default is `Both`.

Indicator.ChartType

constants to specify available chart types for plotting a data series

Synopsis

`ChartType.`*`constant`*

Constants

`Line`	A line graph (the default). ShareScope will draw a line joining each value of the data series. If there are `undefined` values at the beginning (or end) of the data series, the line will start (or end) with the first (or last) valid value. Where `undefined` values appear in the middle of the data, the indicator line will simply join the defined values on either side.
`Clouds`	A filled line graph. The area between the line and the y=0 line is filled.
`Filled`	Another type of filled line graph. The area between the line and bottom of the window is filled.
`Histogram`	A histogram. The bar will be drawn from y=0 to the data series value. If a value is `undefined` no bar is drawn.
`Block`	A block graph. Similar to histogram, but with no gaps between the bars.
`Background`	The data series is not plotted, but instead used to colour the background. The colour mode will determine how a colour is chosen. This mode works well with `ColourMode.Graded`. **ShareScript v1.1**

See Also

`Indicator.setSeriesChartType()`

Indicator.clearHorizontalLines()

remove any horizontal lines

Synopsis

`clearHorizontalLines()`

Description

The `clearHorizontalLines()` function will remove any horizontal lines that have been added to the indicator with `setHorizontalLine()`.

Horizontal lines allow you to illustrate key levels of an indicator to the user.

See Also

`Indicator.setHorizontalLine()`

Indicator.ColourMode

constants to specify available colour modes for plotting a data series

Synopsis

`ColourMode.`*`constant`*

Constants

`Single`	A single colour is used for the plot (the default).
`UpDown`	If the last change of the indicator was up, colour 1 is used. If down, colour 2 is used.
`PosNeg`	If the indicator value is >=0, colour 1 is used. If <0, colour 2 is used.
`Graded`	A gradient of colours will be used for the plot, ranging from colour 1 (for the max value) to colour 2 (for the min value). **ShareScript v1.1**

See Also

`Indicator.setSeriesColourMode()`, `Indicator.setRange()`

Indicator.getBackColour()

get the background colour of the indicator's window

Synopsis

`getBackColour()`

Returns

An integer representing the background colour of the indicator window.

Description

The `getBackColour()` function can be used to choose colours for the indicator data series plot, such that it contrasts with the user-selected background colour of the indicator window.

See Also

`Colour`, `Indicator.setSeriesColour()`

Indicator.getBarLength() ShareScript v1.35

returns the chart's bar period length

Synopsis

`getBarLength()`

Returns

A string encoding the bar period length.

Description

Call `getBarLength()` to find out the period (e.g. daily, weekly) of the bars passed to `getGraph()`. The string returned by this function is in the same format as `Study.getBarLength()`. It is only valid to call this function from an indicator's `getGraph()` method, not from `init()`.

See Also

`Study.getBarLength()`

Indicator.getGraph()

method invoked when ShareScope needs indicator data for an instrument

Synopsis

`function getGraph(shareObj, data)`

Arguments

`shareObj`	The `Share` object that ShareScope needs to generate indicator data for.
`data`	An array of `PriceData` objects corresponding to the bars on the main chart. The length of this array will vary depending on the length of the `Share`'s price history and the user-selected bar width (e.g. daily or weekly).

Description

ShareScope invokes an Indicator's `getGraph()` method when it needs to generate data to plot the indicator for an instrument. It passes the current instrument and OHLCV data that can be used as input for the indicator.

When the indicator is added, the user can choose whether the data source is daily bars, weekly bars, or uses the main graph time period (which can be a range of bar widths). The length of the `data` array passed to `getGraph()` will reflect this choice.

Normally, the `getGraph()` function you write should return one (or more) data series of exactly the same length as the input `data`. ShareScope will report an error if the length is different. However, you can use the `@Future` directive to disable this check and tell ShareScope that you will return data *longer* than the input data. This extra data will be displayed if "Show Future" is enabled on the graph.

If you return a single array, this will be referred to as data series 0. You can return a single array using code like this:

```
function getGraph(instr, data)
{
        var x = new Array();
        (add code that populates elements 0 to data.length-1 of x)
        return x;
}
```

You can also return multiple sets of data from `getGraph()`. In this case you should return an array of arrays. Each array will represent a separate data series, and can have its own chart type and colour. However, all will share the same y-axis range. The first array returned will be referred to as data series 0, the next is series 1, and so on. The maximum number of data series allowed is now 32 (this was increased from 8).

When multiple data series are returned by an indicator, data series 0 is considered to be the 'primary' data series: it will be plotted on top of any others, and the indicator value (e.g. when displayed as a watermark) will be the last data series 0 value.

In the following example, x is data series 0, y is data series 1 and z is data series 2.

```
function getGraph(instr, data)
{
        var x = new Array();
        var y = new Array();
        var z = new Array();

        (add code that initialises elements of x, y & z)

        return [x,y,z];
}
```

An element of an array may be left (or explicitly set to) `undefined`. In this case the value will treated as undefined by ShareScope. See `Indicator.ChartType` for more details.

See Also

`PriceData`, `Indicator.setSeriesChartType()`, `Indicator.setSeriesColour()`, `Indicator.setSeriesColourMode()`, `Indicator.setSeriesLineStyle()`

Indicator.init()

method invoked when an indicator is created

Synopsis

`function init(status)`

Arguments

`status`	A parameter passed (by ShareScope) telling you why ShareScope is initialising the indicator. This should be compared to the constants defined on the `Indicator` object (e.g. `Adding`).

Returns

You should return a value of `false` if you don't want the indicator to be added (or replaced). See below for details.

Description

ShareScope invokes an indicator's `init()` method when an `Indicator` object is created by ShareScope. It is guaranteed to be called before any call to `getGraph()`. You do not need to supply an `init()` method if you do not need one.

The `init()` method (and its return value) is handled by ShareScope as follows:

When the user **adds an indicator** for the first time – ShareScope creates a new `Indicator` object, then calls `init()` with the status set to `Adding`. If `init()` returns `false`, the indicator is not added, and the `Indicator` object is discarded.

When the user **edits an indicator** – ShareScope will create a new `Indicator` object, and copy any data in the original indicator's `storage` area to the new object. It then calls the `init()` function of the new `Indicator` object with the status set to `Editing`. If the `init()` function returns `false`, the new indicator is discarded, and the user's graph is left unchanged.

When ShareScope **loads**, it will create an `Indicator` object for any ShareScript indicators on the user's graph. It will then call the `init()` function with a status of `Loading`. If the `init()` function returns `false`, the indicator will be disabled (but will still be visible to the user).

Example

The following `init()` function sets the indicator title. Please refer to the `Column.init()` entry for a more detailed example, including the use of dialog boxes for user input, the storage area, and appropriate handling of the status parameter.

```
function init()
{
	setTitle("My indicator");
}
```

Indicator.Layer — ShareScript v1.31

constants to specify a drawing layer

Synopsis

`Layer.`*`constant`*

Constants

`Bottom`	The bottom drawing layer (beneath the bars and any analytics).
`Top`	The top drawing layer (above the bars and any analytics).

See Also

`Indicator.setLayer()`

Indicator.Range

constants to control the indicator y-axis range

Synopsis

`Range.`*`constant`*

Constants

`Dynamic`	ShareScope will determine the y-axis range dynamically (the default). The y-axis will accommodate the maximum and minimum values present across all the data series.
`CentreZero`	Range is dynamically determined, but y=0 will be in the centre.
`MinMax`	The y-axis range is specified by a minimum and maximum value. You can also leave either the minimum or maximum value `undefined` to have only it dynamically determined (from the data). If `Range.MinMax` is used and both minimum and maximum are left `undefined`, the behaviour is the same as `Range.Dynamic`.
`Parent`	ShareScope should use the y-axis belonging to the parent graph. If no parent graph exists, a `Dynamic` range is used.

`Dynamic`	ShareScope will determine the y-axis range dynamically (the default). The y-axis will accommodate the maximum and minimum values present across all the data series.
`ParentMerge`	As `Range.Parent` but the range of the parent graph y-axis will be extended (if necessary) to accommodate the values in the indicator data series.

See Also

`Indicator.setRange()`

Indicator.setHorizontalLine()

add a horizontal line to the indicator display at a given y-axis value

Synopsis

`setHorizontalLine(value)`

Arguments

`value`	The y-axis value where a line should be placed.

Description

The `setHorizontalLine()` function adds a new horizontal line to the indicator display. You can add as many lines as are required. These lines can illustrate key levels of an indicator to the user. They can persist across multiple calls to `getGraph()`, or you can create new levels individually for each new indicator plot.

Example

```
clearHorizontalLines();
for (var i = 0; i<=100; i+=10)
      setHorizontalLine(i);
```

See Also

`Indicator.clearHorizontalLines()`

Indicator.setLayer() ShareScript v1.31

determines whether a main graph indicator is drawn above or below the chart

Synopsis

`setLayer(layer)`

Arguments

`layer`	A constant from the `Indicator.Layer` object specifying the target layer.

Description

The `setLayer()` function allows you to specify if a main graph indicator should be drawn beneath the other main chart features (the default) or on top of them. Note that this setting applies to the whole indicator, not to an individual data series. The draw ordering of data-series within the indicator remains unchanged (see `Indicator.getGraph` for details).

Example

```
setLayer(Layer.Top); // draw above everything
```

See Also

`Indicator.Layer`

Indicator.setRange()

controls the y-axis range for the indicator window

Synopsis

```
setRange(rangeMode)
setRange(rangeMode, min, max)
```

Arguments

`rangeMode`	A constant from the `Indicator.Range` object to select the desired y-axis range.
`min`	Only required for `Range.MinMax`. A number specifying the minimum y-axis value. If `undefined`, the minimum is determined dynamically.
`max`	Only required for `Range.MinMax`. A number specifying the maximum y-axis value. If `undefined`, the maximum is determined dynamically.

Description

By default, ShareScope will calculate a y-axis range which will accommodate the values from all the data series you return from `getGraph()`. You can use this function if you wish to specify a different y-axis range behaviour.

Note that ShareScope normally adds a small margin to the y-axis range, so the maximum and minimum data values do not lie on the top and bottom edges of the graph. You can use the `Range.MinMax` setting to switch off this behaviour and specify exactly the range required.

See `Indicator.Range` for more information about the available options for y-axis behaviour.

Example

This example sets the minimum y-axis value explicitly to 0, whilst allowing the maximum y-axis value to be determined from the data being graphed:

```
setRange(Range.MinMax, 0, undefined);
```

See Also

`Indicator.Range`

Indicator.setSeriesChartType()

sets the chart type for a data series

Synopsis

```
setSeriesChartType(series, chartType)
```

Arguments

`series`	An integer specifying the data series to change the chart type for (see `Indicator.getGraph()` for information about data series numbering).
`chartType`	A constant from the `Indicator.ChartType` object to select the desired chart type.

Description

An indicator can consist of several data series plotted in different ways. `setSeriesChartType()` allows you to specify how each individual data series is drawn by ShareScope.

See `Indicator.ChartType` for details of the available chart types.

Example

```
setSeriesChartType(0, ChartType.Histogram);
```

See Also

`Indicator.ChartType`, `Indicator.setSeriesColour()`, `Indicator.setSeriesColourMode()`, `Indicator.setSeriesLineStyle()`

Indicator.setSeriesColour()

sets the colour(s) for a data series

Synopsis

```
setSeriesColour(series, colour1)
setSeriesColour(series, colour1, colour2)
```

Arguments

`series`	An integer specifying the data series to change the plot colour for (see `Indicator.getGraph()` for information about data series numbering).
`colour1`	An integer specifying the colour for the plot.
`colour2`	Optional. An integer specifying an alternative colour for the plot. This is used when certain `Indicator.ColourMode` settings are active for the series.

Description

The `setSeriesColour()` function allows you to set the colour that a given data series should be plotted in. The alternative colour is only used when certain colour modes are active. See `Indicator.setSeriesColourMode()` for more details.

You can use the `Indicator.getBackColour()` to ensure that a colour is used that will contrast with the indicator window background.

Example

```
setSeriesColour(0, Colour.Red);
setSeriesColour(1, Colour.RGB(128,255,192));
```

See Also

`Colour`, `Indicator.setSeriesColourMode()`, `Indicator.getBackColour()`

Indicator.setSeriesColourMode()

sets the colour mode for a data series

Synopsis

```
setSeriesColourMode(series, colourMode)
```

Arguments

`series`	An integer specifying the data series to change the colour mode for (see `Indicator.getGraph()` for information about data series numbering).
`colourMode`	A constant from the `Indicator.ColourMode` object to select the desired colour mode.

Description

The `setSeriesColourMode()` function allows you to set the colour mode for a data series. A data series plot can be a single colour, or two colours which alternate. See `Indicator.ColourMode` for the available colour mode options.

See Also

`Indicator.ColourMode`, `Indicator.setSeriesColour()`

Indicator.setSeriesLineStyle()

sets the line style for a data series

Synopsis

```
setSeriesLineStyle(series, pen)
setSeriesLineStyle(series, pen, width)
```

Arguments

`series`	An integer specifying the data series to change the line style for (see `Indicator.getGraph()` for information about data series numbering).
`pen`	A constant from the `Pen` object specifying the type of pen to use.
`width`	Optional. An integer giving the width of the pen. Valid values are 0 to 7. If not specified this defaults to 0 (the thinnest line). Greater widths are only allowed for a pen type of `Pen.Solid`.

Description

The `setSeriesLineStyle()` function allows you to select a pen that a data series plot will be drawn with.

Line styles are only supported for a chart type of `ChartType.Line`. See `Indicator.setSeriesChartType()` for more details.

Example

```
setSeriesLineStyle(0, Pen.Solid, 3);
setSeriesLineStyle(1, Pen.Dash);
```

See Also

`Pen`, `Indicator.setSeriesColour()`, `Indicator.setSeriesChartType()`

Indicator.setTitle()

sets the indicator window title

Synopsis

```
setTitle(s)
```

Arguments

`s`	A string providing a name for the indicator window title

Description

The `setTitle()` function allows you to set the indicator's title, displayed at the top of an indicator window. You should normally call this method from the `init()` function.

Chart Study Objects Reference

ChartStudy — ShareScript v1.2

ShareScope chart study interface — Object→ChartStudy

Construction

ShareScope manages the creation and destruction of `ChartStudy` objects after the user adds a chart study script to a graph setting. They are not created using the normal JavaScript `new()` operator.

Methods

A `ChartStudy` object has many methods you can call and six methods you can supply to create a new ShareScript chart study (these are shown first below).

The `load()` method allows you to make library functions available within the `ChartStudy` object, rather than defining them in the global object. This will prevent namespace collisions when your study uses functions defined externally.

Methods You Supply

`init()`	Optional. ShareScope will call the `ChartStudy` object's `init()` method once when the object is first created.
`onNewChart()`	Optional. ShareScope will call the `ChartStudy` object's `onNewChart()` method whenever a new chart is about to be drawn.
`onBarClose()`	Optional. ShareScope will call the `ChartStudy` object's `onBarClose()` method once for each complete bar added to the chart.
`onNewBarUpdate()`	Optional. ShareScope will call the `ChartStudy` object's `onNewBarUpdate()` method whenever a new partial bar is created, or the existing partial bar's OHLCV values change.
`onMouseClick()`	Optional. ShareScope will call the `ChartStudy` object's `onMouseClick()` method if the study has input focus and the user clicks on the chart.
`onZoom()`	Optional. ShareScope will call the `ChartStudy` object's `onZoom()` method when the user changes the range of visible chart bars.

General Methods

`load()`	Load and execute a ShareScript file in this study object. See the global `load()` method for full details.
`getBackColour()`	Returns the chart's background colour.
`getBarLength()`	Returns the chart's bar period length.
`getCurrentShare()`	Returns the charted instrument.
`getMinVisibleBarIndex()`	Returns the index of the left-most (earliest) visible bar on the chart.
`getMaxVisibleBarIndex()`	Returns the index of the right-most (latest) visible bar on the chart.

Study Panel Methods

`setTitle()`	Set the study's title.
`setInfoText()`	Set the study's info text (displayed under the title).
`createButton()`	Create a button on the study's panel.
`setButtonText()`	Set a panel button's caption.
`deleteButton()`	Delete a panel button.
`deleteButtons()`	Delete all the panel buttons.

Drawing Control Methods

`setAltRange()`	Set up an alternative y-axis range for drawing.
`useAltRange()`	Enable or disable drawing using the alternative y-axis range.
`setPenColour()`	Change the pen colour.
`setPenStyle()`	Change the pen line style.
`setBrushColour()`	Change the brush colour.
`setFontColour()`	Change the font colour.
`setFontStyle()`	Change the font style.
`setFillMode()`	Change the fill mode.
`setFrame()`	Set the target frame for drawing (i.e. main chart or volume area)
`setLayer()`	Set the layer for drawing (i.e. above or below the main chart).

Drawing Methods

`clearDisplay()`	Remove all drawn objects from the chart.
`undrawItem()`	Remove a single drawn object from the chart.
`moveTo()`	Move the cursor position.
`lineTo()`	Draw a line from the cursor to a new position.
`beginPath()`	Begin a path, for grouping together a series of lineTo() commands.
`endPath()`	End a path.
`drawPath()`	Draw a completed path, using the current pen style.
`fillPath()`	Fill a completed path, using the current brush.
`drawAndFillPath()`	Draw and fill a completed path.
`drawText()`	Add text to the chart.
`drawTextEx()`	Add text, with additional control over the position.
`drawSymbol()`	Draw a symbol on the chart.
`drawSymbolEx()`	Draw a symbol, with additional control over the position.
`drawShape()`	Draw a shape, using the current pen style.
`fillShape()`	Fill a shape, using the current brush.
`drawAndFillShape()`	Draw and fill a shape.

Constants

These constants are defined on the study object, and can be used for comparison with the status parameter passed to the `init()` function (see the `init()` method description for details):

`Loading`	ShareScope is creating a new `ChartStudy` object belonging to a chart window.
`Adding`	The user has added a new study to the setting.
`Editing`	The user has indicated they wish to edit the study.

Properties

`bars`	An array of `Bar` objects representing the chart bars.
	This property is available in all `ChartStudy` methods, with the exception of `init()` where it is `undefined`.
`bar`	A `Bar` object representing the current bar - equivalent to `bars[barIndex]`.
	This property is available in the `onBarClose()`, `onNewBarUpdate()` and `onMouseClick()` methods. It is `undefined` elsewhere.
`barIndex`	An integer, giving the index of the current bar in the `bars` array.
	This property is available in the `onBarClose()`, `onNewBarUpdate()` and `onMouseClick()` methods. It is `undefined` elsewhere.
`isIntraday`	A boolean value that is `true` if your study is on an intraday chart, and `false` if on a historical chart.
`storage`	The study's `Storage` object, which provides a permanent storage mechanism for data. See section 5 of this reference for details.
`BoxAlign`	Constants for positioning the bounding boxes of drawn objects, for use with the `drawText()` and `drawSymbol()` functions.
`FillMode`	Constants for specifying a fill mode, for use with the `setFillMode()` function.
`Frame`	Constants for specifying a particular frame of the chart, for use with the `setFrame()` function.
`Layer`	Constants for specifying a drawing layer, for use with the `setLayer()` function.
`TextAlign`	Constants for positioning text within its bounding box, for use with the `drawText()` function.
`Shape`	Constants for specifying a shape, for use with the `drawShape()` function.
`Symbol`	Constants for specifying a symbol, for use with the `drawSymbol()` function.

Description

When you create a script for a ShareScript chart study, the functions and variables you define in the script become methods and properties of a `ChartStudy` object. ShareScope calls certain well-defined `ChartStudy` methods in response to particular chart events, so by adding your own code to these "event handler" methods, you can customise the behaviour of the chart.

`ChartStudy` objects provide access to the chart bars through the `bars`, `barIndex` and `bar` properties and provide methods that allow you to draw text, lines, shapes and symbols on the chart.

Although the user adds a chart study script to a particular graph *setting*, ShareScope will create a new `ChartStudy` object for each *chart window* visible on the screen. This behaviour differs from ShareScript columns (and indicators), where only one object is created from the script, and this single object is responsible for returning the calculated values to all list (or chart) windows sharing the setting.

This architecture, with individual study objects for each chart window, allows a much richer customisation of charts, since studies can react to events (such as mouse or button clicks) within specific charts, and maintain individual states in response to these and other events.

When the user first adds a study script to a graph setting, ShareScope will first create a temporary `ChartStudy` object that belongs to the setting, rather than a particular chart. This allows a script to present a dialog box to the user to set up any custom parameters for the study. Your script can store the user parameters to the `ChartStudy` storage area. This allows subsequent `ChartStudy` objects, which will be created for each chart window, to retrieve the user's parameters from the storage area, rather than displaying the dialog again. Please refer to the entry for `ChartStudy.init()` for more information.

Note that the `ChartStudy` object is at the start of the scope chain when executing the code in your study script. Thus a call to e.g. the `load()` method will call the current `ChartStudy` object's `load()` method, rather than the `load()` method in the global object.

Any functions you call that are not defined by your `ChartStudy` object will be resolved in the next object in the scope chain (i.e. the global object).

Chart study scripts should be placed in your `ShareScript/Studies/` directory.

Directives

Study directives allow you to tell ShareScope how to treat the study created by your script. They are detailed here for completeness, although they are not technically part of the ShareScript language (directives are placed inside comments, and are used by ShareScope itself, rather than the ShareScript interpreter). Directives take the form:

`@Field:Value`

The available fields and the values they can take are detailed below. Note that field names and values are not case sensitive. Normally, you should place any directives you wish to use in a comment, near the top of your ShareScript file.

Field	**Valid values/description**
`@Name`	Any text. ShareScope will use this value as the name of the study. This will appear in the study's panel, and in the Add ShareScript Study dialog. e.g. `//@Name:My Study`
`@Description`	Any text. A description of what your Script does, which is displayed in the Add ShareScript Study dialog. e.g. `//@Description:My customised bar colouring`

`@Type`	Valid values are `Historical`, `Intraday` or `Both`. Restricts the use of your study to the selected chart type. The default is `Both`.
`@Env`	Valid values are `Development` or `Production` This directive modifies the environment ShareScope provides to the user to interact with the study. The default is `Development`. When `Production` is selected – (i) The "Refresh Script" command is removed from the menu (ii) "Edit study" will not allow the user to select a different script. It will instead call the Study's `init()` method with `Editing` status.
`@Editable`	Valid values are `Yes` or `No`. Applicable only with `Production` environment. This directive tells ShareScope whether to display the "Edit study" option. The default is `Yes`.

ChartStudy.bar

Built-in property referring to the current bar

Synopsis

`bar`

Description

This property provides access to a `Bar` object representing the current chart bar. This property is set during `onBarClose()`, `onNewBarUpdate()` and `onMouseClick()`. It is `undefined` elsewhere.

This property is functionally equivalent to `bars[barIndex]`.

See Also

`Bar`, `ChartStudy.bars`, `ChartStudy.barIndex`

ChartStudy.barIndex

Built-in property giving the index of the current bar

Synopsis

```
barIndex
bars[barIndex]
```

Description

This property is of integer type and provides the index of the current bar. This property is set during `onBarClose()`, `onNewBarUpdate()` and `onMouseClick()`. It is `undefined` elsewhere.

`barIndex` is always a valid index for the `bars` array (i.e. `bars[barIndex]` is always valid) which will yield a `Bar` object representing the current bar.

See Also

`Bar`, `ChartStudy.bar`, `ChartStudy.bars`

ChartStudy.bars

Built-in array property providing access to all chart bars

Synopsis

`bars[]`

Description

This property provides access an array of `Bar` objects representing all the chart bars. This property is available in all methods, except `init()` where it is `undefined`.

The left-most bar on the chart is `bars[0]`, the right-most is `bars[bars.length-1]`. The last bar in the `bars` array may be a partial rather than a complete bar (you can use the `Bar` object's `isComplete` property to check).

See Also

`Bar`, `ChartStudy.bars`, `ChartStudy.barIndex`, `ChartStudy.getBarLength()`

ChartStudy.beginPath()

start accumulating line drawing commands into a path

Synopsis

`beginPath()`

Description

A path is a set of line segments, where the end of each line segment is the starting point of the next (i.e. there are no breaks in a path). ShareScope can draw a path much faster than it could draw the individual line segments.

The `beginPath()` function tells the drawing system to start building a path. Until `endPath()` is called, all subsequent `lineTo()` commands will add the line segment to the current path, rather than drawing a line. When a path has been defined, it can be drawn using the `drawPath()`, `fillPath()` or `drawAndFillPath()` commands.

When `beginPath()` is called, the cursor positon will become the starting point of the path. If a path already exists when `beginPath()` is called, any existing contents will be removed.

Example

The following extract from a script draws a box around the second bar, using a path.

```
beginPath();
moveTo(0.5, bars[1].high);
lineTo(1.5, bars[1].high);
lineTo(1.5, bars[1].low);
lineTo(0.5, bars[1].low);
lineTo(0.5, bars[1].high);
endPath();
drawPath();
```

See Also

`ChartStudy.moveTo()`, `ChartStudy.lineTo()`, `ChartStudy.endPath()`, `ChartStudy.drawPath()`

ChartStudy.BoxAlign

constants for bounding box alignment

Synopsis

`BoxAlign.`*`constant`*

Constants

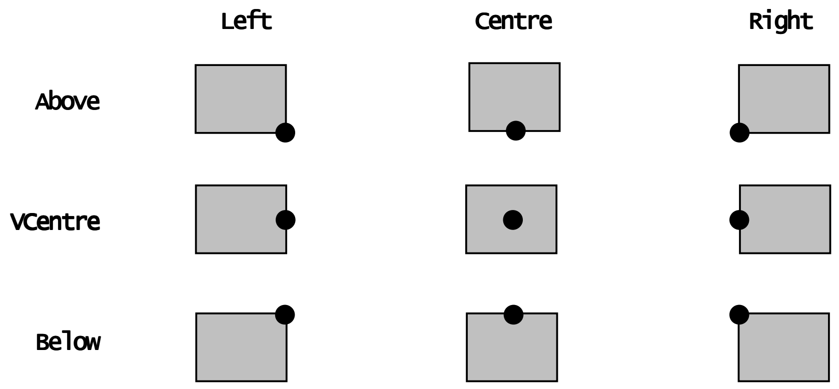
`Left`	Horizontally aligns the box to the left of the specified x position.
`Right`	Horizontally aligns the box to the right of the specified x position.
`Center/Centre`	Horizontally aligns the box centred on the specified x position.
`Above`	Vertically aligns the box above the specified y position.
`Below`	Vertically aligns the box below the specified y position.
`VCenter/VCentre`	Vertically aligns the box centred on the specified y position.

Description

These constants determine the how the bounding box for text or a symbol will be positioned relative to the point you specify when drawing the object.

You can combine these constants using the bitwise OR operator (|) to specify both horizontal and vertical positioning. If you fail to specify horizontal and/or vertical alignment then the default behaviour is to centre on that axis.

The figure below illustrates the positioning of a text box (grey rectangle) relative to the (x,y) position specified (black dot) for the `drawText()` function under all valid combinations of the above constants:



See Also

`ChartStudy.drawText()`, `ChartStudy.drawSymbol()`

ChartStudy.clearDisplay()

remove all drawn objects from the chart

Synopsis

`clearDisplay()`

Description

A script can use the `clearDisplay()` function to remove anything previously drawn on the chart. This function does not effect the study's panel, which will remain unchanged.

Note that ShareScope automatically removes all script-drawn objects from the chart before it calls a study's `onNewChart()` method, so `clearDisplay()` is only needed if you want clear the display in response to a different event.

If you want to only remove a subset of the drawn objects, the `undrawItem()` function can be used instead.

See Also

`ChartStudy.undrawItem()`

ChartStudy.createButton()

adds a button to the study's panel

Synopsis

`createButton(caption, func)`

Arguments

`caption`	The label for the button (a string).
`func`	A function (defined in the study object) to be called when the button is clicked.

Returns

The newly created button's handle (an integer).

Description

The `createButton()` function creates and adds a button to the study's panel, and associates it with a study object method that will be called whenever the user clicks on the button. It returns a handle which can later be used to change the button's caption or remove the button from the panel. The return value can be ignored if your script has no need to refer to the button later in the script.

The callback method you define is expected to have an arity of zero (i.e. it should not accept any parameters) – see the buttonCallback function in the example below.

Example

The following script adds a button to the study's panel and prints a message to the console when that button clicked. The button handle is retained using a variable with study object scope, so it could be used to refer to the button at a later time (not illustrated in this example).

```
var handle;

function init()
{
	handle = createButton("my label", buttonCallback);
}

function buttonCallback()
{
	print("button was clicked!");
}
```

See Also

`ChartStudy.setButtonText()`, `ChartStudy.deleteButton()`, `ChartStudy.setInfoText()`

ChartStudy.deleteButton()

remove a button from the study's panel

Synopsis

`deleteButton(handle)`

Arguments

`handle`	The handle of the button to delete from the panel (an integer).

Returns

A boolean value – `true` if the button was found and removed, `false` otherwise.

Description

The `deleteButton()` function removes a previously created button from the study's panel. To remove all the buttons from a panel at once, consider using `deleteButtons()` instead.

See Also

`ChartStudy.createButton()`, `ChartStudy.deleteButtons()`

ChartStudy.deleteButtons()

remove all buttons from the study's panel

Synopsis

`deleteButtons()`

Description

The `deleteButtons()` function removes all previously created buttons from the study's panel.

See Also

`ChartStudy.createButton()`, `ChartStudy.deleteButton()`

ChartStudy.drawAndFillPath()

draws and fills a path on the chart

Synopsis

`drawAndFillPath()`

Returns

An integer giving the handle of the path (which can be used to later undraw the path), or `undefined` if the path was invalid.

Description

The `drawAndFillPath()` function is almost identical to `drawPath()`, except that the path is additionally filled using the current brush and fill-mode. Please see the entry for `drawPath()` below for further information.

See Also

`ChartStudy.drawPath()`, `ChartStudy.fillPath()`, `ChartStudy.FillMode`

ChartStudy.drawAndFillShape()

draws and fills a shape on the chart

Synopsis

```
drawAndFillShape(x1, y1, x2, y2, shape)
drawAndFillShape(x1, y1, x2, y2, shape, ratio)
```

Arguments

All arguments as for `drawShape()`.

Returns

As for `drawShape()`.

Description

The `drawAndFillShape()` function is almost identical to `drawShape()`, except that the shape is additionally filled using the current brush and fill-mode. Please see the entry for `drawShape()` below for further information.

See Also

`ChartStudy.Shape`, `ChartStudy.drawShape()`, `ChartStudy.fillShape()`, `ChartStudy.FillMode`

ChartStudy.drawPath()

draw a path on the chart

Synopsis

```
drawPath()
```

Returns

An integer giving the handle of the path (which can be used to later undraw the path), or `undefined` if the path was invalid.

Description

The `drawPath()` function draws the current path onto the chart using the current pen. Use `fillPath()` or `drawAndFillPath()` to draw filled paths (using the current brush).

The `drawPath()` function can return `undefined` if the path is invalid i.e. if it contains no line segments.

For an full path drawing example please see the `beginPath()` function.

See Also

`ChartStudy.beginPath()`, `ChartStudy.fillPath()`, `ChartStudy.drawAndFillPath()`

ChartStudy.drawShape()

draws a shape on the chart

Synopsis

```
drawShape(x1, y1, x2, y2, shape)
drawShape(x1, y1, x2, y2, shape, ratio)
```

Arguments

`x1`	The start point x-coordinate. See `ChartStudy.moveTo()` for the different ways an x-coordinate can be specified.
`y1`	The start point y-coordinate.
`x2`	The end point x-coordinate
`y2`	The end point y-coordinate
`shape`	A constant from the `ChartStudy.Shape` object specifying the shape to draw.
`ratio`	A number giving the ratio of width to length. If not specified, the default is 0.5.

Returns

An integer giving the handle of the shape (which can be used to later undraw the shape).

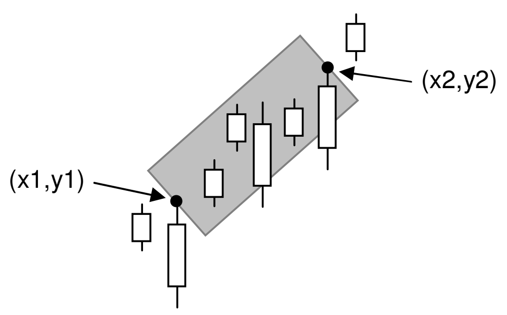
Description

The `drawShape()` function draws a shape on the chart. Unlike symbols, which are drawn at a single point, shapes can be used to highlight a series of bars or other features that stretch across time.

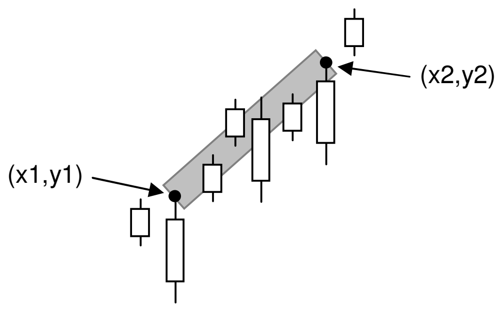
`drawShape()` is used to draw the outline of the shape using the current pen. Use `fillShape()` or `drawAndFillShape()` to draw filled shapes (using the current brush).

The shape is specified by providing a start point and an end point. These points specify the mid-point of each end of the bounding rectangle defining the shape (see below for an illustration). The ratio parameter is the ratio of the width to length of the rectangle.

Width/Length Ratio 0.5



Width/Length Ratio 0.1



Shapes are a useful and easy way for a script to highlight a trend or period on the chart. For more flexible polygon drawing, consider using a path instead.

Example

The following example draws an arrow from the bar with index 5 to the bar with index 10 (similar to the illustration above).

```
drawShape(5, bars[5].high, 10, bars[10].high, Shape.Arrow, 0.5);
```

See Also

`ChartStudy.Shape`, `ChartStudy.fillShape()`, `ChartStudy.drawAndFillShape()`

ChartStudy.drawSymbol()

draws a symbol on the chart

Synopsis

```
drawSymbol(x, y, symbol)
drawSymbol(x, y, symbol, char, boxAlign, size, isFilled)
```

Arguments

`x`	The symbol's x-coordinate. See `ChartStudy.moveTo()` for the different ways an x-coordinate can be specified.
`y`	The symbol's y-coordinate.
`symbol`	A constant from the `ChartStudy.Symbol` object specifying the symbol to draw.
`char`	An optional single character string which will be shown inside the symbol. Pass `null` or `undefined` if you don't want a character inside (the default).
`boxAlign`	An optional integer value specifying the alignment of the symbol relative to the point specified by the (x,y) parameters. This should normally be two of the constants defined by the `ChartStudy.BoxAlign` object combined using the bitwise "or" operator (|). If not specified, the symbol will be centred on the point.
`size`	An optional integer giving the size of the symbol in pixels (the default is 16).
`isFilled`	An optional boolean value specifying whether the symbol should be filled using the current brush colour (the default is `true`).

Returns

An integer giving the handle of the symbol (which can be used to later undraw the symbol).

Description

The `drawSymbol()` function draws a symbol on the chart. Symbols can be used to annotate the chart or highlight certain features. ShareScope uses symbols to show events, swing and candlestick patterns. Unlike shapes, the size of a symbol is specified in pixels, and the drawn size remains constant irrespective of the size of the chart window.

Symbols are drawn with the current pen and (optionally) filled using the current brush. If you specify a character to be displayed in the symbol, this will be drawn using the current font style and colour (note that the font size is not used – ShareScope will scale the font such that the character fits inside the symbol). Not all symbols are designed to have text drawn inside (e.g. the cross symbol does not work well with text).

Symbols are positioned relative to the specified (x,y) position using the `boxAlign` parameter. See `ChartStudy.drawText()` for more information about this parameter.

Example

The example below draws a red downwards pointing flag symbol with an "A" character inside, centred above the high of bar index 5. By default the symbol is 16 pixels in size.

```
setBrushColour(Colour.Red);
drawSymbol(5, bars[5].high, Symbol.FlagDown, "A", BoxAlign.Centre|BoxAlign.Above);
```

See Also

`ChartStudy.Symbol`, `ChartStudy.BoxAlign`, `ChartStudy.drawText()`

ChartStudy.drawSymbolEx()

draws a symbol on the chart with additional control over the position

Synopsis

```
drawSymbolEx(x, y, xOffset, yOffset, symbol)
drawSymbolEx(x, y, xOffset, yOffset, symbol, char, boxAlign, size, isFilled)
```

Arguments

All arguments as for `drawSymbol()`, with the addition of:

`xOffset`	An integer giving a pixel offset to the x-coordinate. Negative values offset to the left, positive values offset to the right.
`yOffset`	An integer giving a pixel offset to the y-coordinate. Negative values offset upwards, positive values downwards.

Returns

An integer giving the handle of the symbol (which can be used to later undraw the symbol).

Description

The `drawSymbolEx()` function is almost identical to the `drawSymbol()` function, but takes two additional arguments which allow you to refine the symbol position on the chart by adding pixel-based offsets to the x and y positions (which are based on date/time and value). This allows you to position the symbol e.g. just above a bar, rather than exactly aligned with it.

See Also

`ChartStudy.drawSymbol()`

ChartStudy.drawText()

draws text on the chart

Synopsis

```
drawText(x, y, str)
drawText(x, y, str, boxAlign, textAlign, isFilled, isBoxed)
```

Arguments

`x`	The text's x-coordinate. See `ChartStudy.moveTo()` for the different ways an x-coordinate can be specified.
`y`	The text's y-coordinate.
`str`	The text to draw (a string).
`boxAlign`	An optional integer value specifying the alignment of the text box relative to the point specified by the (x,y) parameters. This should normally be two of the constants defined by the `ChartStudy.BoxAlign` object combined using the bitwise "or" operator (`|`). If not specified, the text box will be centred on the point.
`textAlign`	An optional integer value specifying how the draw should be aligned within

	its bounding box. This value should be a constant defined by the `ChartStudy.TextAlign` object. If not specified, the text will be left-aligned within its bounding box.
`isFilled`	An optional boolean value specifying whether the text's bounding box should be filled using the current brush colour (the default is `false`).
`IsBoxed`	An optional boolean value specifying whether the edge of the text's bounding box should be drawn using the current pen (the default is `false`).

Returns

An integer giving the handle of the text object (which can be used to later undraw the text).

Description

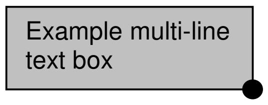
The `drawText()` function draws text to the chart using the current font. The text can be single line of text, or can span multiple lines if the `str` parameter includes '\n' (newline) characters.

ShareScope calculates a bounding box that will exactly fit around the text, which can (optionally) be drawn and filled using the current pen and brush. This box is positioned relative to the specified (x,y) position using the `boxAlign` parameter.

If you are drawing text that spans multiple lines, you can also decide how the text is formatted within the bounding box using the `textAlign` parameter (a single line of text will fit exactly within the bounding box).

Example

The following example draws the multi-line text box illustrated below. The box will be positioned above and left of the specified position (shown as a black circle in the illustration).



```
drawText(20, bars[20].high, "Example multi-line\ntext box",
                    BoxAlign.Left|BoxAlign.Above, TextAlign.Left, true, true);
```

See Also

`ChartStudy.setFontStyle()`, `ChartStudy.BoxAlign`, `ChartStudy.TextAlign`, `ChartStudy.drawTextEx()`

ChartStudy.drawTextEx()

draws text on the chart with additional control over the position

Synopsis

```
drawTextEx(x, y, xOffset, yOffset, str)
drawTextEx(x, y, xOffset, yOffset, str, boxAlign, textAlign, isFilled, isBoxed)
```

Arguments

All arguments as for `drawText()`, with the addition of:

`xOffset`	An integer giving a pixel offset to the x-coordinate. Negative values offset to the left, positive values offset to the right.
`yOffset`	An integer giving a pixel offset to the y-coordinate. Negative values offset upwards, positive values downwards.

Returns

An integer giving the handle of the text object (which can be used to later undraw the text).

Description

The `drawTextEx()` function is almost identical to the `drawText()` function, but takes two additional arguments which allow you to refine the text bounding box position on the chart by adding pixel-based offsets to the x and y positions (which are based on date/time and value). This allows you to position text e.g. just above a bar, rather than exactly aligned with it.

See Also

`ChartStudy.drawText()`

ChartStudy.endPath()

stop accumulating line drawing commands into a path

Synopsis

`endPath()`

Description

The `endPath()` function tells the drawing system to stop accumulating `lineTo()` commands into the current path. When a path has been defined by a call to `endPath()`, it can then be drawn using the `drawPath()`, `fillPath()` or `drawAndFillPath()` commands.

For an full path drawing example please see the `beginPath()` function.

See Also

`ChartStudy.beginPath()`, `ChartStudy.drawPath()`, `ChartStudy.fillPath()`, `ChartStudy.drawAndFillPath()`

ChartStudy.FillMode

constants to specify a fill mode

Synopsis

`FillMode.`*`constant`*

Constants

`Solid`	Solid fill mode. Anything underneath the filled object will not be visible.
`Transparent`	Transparent fill mode. The background will partially show through the filled object. Note that a logical operation (rather than alpha-blending) is used to combine the background and fill colour.

See Also

`ChartStudy.setFillMode()`, `ChartStudy.setBrushColour()`

ChartStudy.fillPath()

draws a filled path on the chart

Synopsis

```
fillPath()
```

Returns

An integer giving the handle of the path (which can be used to later undraw the path), or `undefined` if the path was invalid.

Description

The `fillPath()` function is almost identical to `drawPath()`, except that the path's outline is not drawn – instead the path is filled using the current brush and fill-mode. Please see the earlier entry for `drawPath()` for further information.

See Also

`ChartStudy.drawPath()`, `ChartStudy.drawAndFillPath()`, `ChartStudy.FillMode`

ChartStudy.fillShape()

draws a filled shape on the chart

Synopsis

```
fillShape(x1, y1, x2, y2, shape)
fillShape(x1, y1, x2, y2, shape, ratio)
```

Arguments

All arguments as for `drawShape()`.

Returns

As for `drawShape()`.

Description

The `fillShape()` function is almost identical to `drawShape()`, except that the shape's outline is not drawn – instead the shape is filled using the current brush and fill-mode. Please see the earlier entry for `drawShape()` for further information.

See Also

`ChartStudy.Shape`, `ChartStudy.drawShape()`, `ChartStudy.drawAndFillShape()`, `ChartStudy.FillMode`

ChartStudy.Frame

constants to specify a chart frame

Synopsis

`Frame.`*`constant`*

Constants

`Chart`	The main share price frame of the chart window.

`Volume`	The volume frame of the chart window (beneath the price frame).

See Also

`ChartStudy.setFrame()`

ChartStudy.getBackColour()

get the background colour of the chart window

Synopsis

`getBackColour()`

Returns

An integer representing the background colour of the chart window.

Description

The `getBackColour()` function can be used to choose colours for a study such that they contrast with the user-selected background colour of the chart window. Note that ShareScope will call `onNewChart()` if the user changes the background colour after the study has been added to the chart.

See Also

`Colour`, `ChartStudy.setPenColour()`, `ChartStudy.setBrushColour()`, `ChartStudy.setFontColour()`

ChartStudy.getBarLength()

returns the chart's bar period length

Synopsis

`getBarLength()`

Returns

A string encoding the bar period length.

Description

The `getBarLength()` function returns the chart's bar period length. This is encoded as string, with the period length (a number) first, followed by a single character that describes the units (e.g. "2w" for 2 week bars).

The possible unit values are:

s	Seconds. Intraday charts always report the bar period length in seconds. e.g. "60s" will be returned for one minute bars.
d	Days.
w	Weeks.
m	Months.

Example

The example below shows you how to extract the period and unit from the string returned by `getBarLength()`:

```
var period = parseInt(getBarLength());
var unit = getBarLength().slice(-1);
```

See Also

Bar, ChartStudy.bars

ChartStudy.getCurrentShare()

returns a study's current charted instrument

Synopsis

getCurrentShare()

Returns

A Share object representing the study's current instrument.

Description

The getCurrentShare() function returns the current instrument for the chart window that the study is attached to. The current share is undefined when ShareScope calls the ChartStudy's init() method.

Note that this function can also return undefined when the chart is not displaying a valid instrument (e.g. synthetic data mode).

See Also

Share

ChartStudy.getMaxVisibleBarIndex()

returns the index of the right-most visible bar

Synopsis

getMaxVisibleBarIndex()

Returns

An integer, giving the maximum bar index currently displayed.

Description

The getMaxVisibleBarIndex() function returns the index of the right-most bar in the window. The associated Bar object can be retrieved by using the return value of this function as in index into the bars array.

This function is generally useful when your script defines an onZoom() method.

Example

The following example prints the close value for the last visible bar to the console:

```
var lastBar = bars[getMaxVisibleBarIndex()];
print(lastBar.close);
```

See Also

ChartStudy.bars, ChartStudy.onZoom(), ChartStudy.getMinVisibleBarIndex()

ChartStudy.getMinVisibleBarIndex()

returns the index of the left-most visible bar

Synopsis

`getMinVisibleBarIndex()`

Returns

An integer, giving the minimum bar index currently displayed.

Description

The `getMinVisibleBarIndex()` function returns the index of the left-most bar in the window. This will be 0 if the user has not zoomed into the chart. The associated `Bar` object can be retrieved by using the return value of this function as in index into the `bars` array.

This function is generally useful when your script defines an `onZoom()` method.

See Also

`ChartStudy.bars`, `ChartStudy.onZoom()`, `ChartStudy.getMaxVisibleBarIndex()`

ChartStudy.init()

method invoked when a study is created

Synopsis

`function init(status)`

Arguments

`status`	A parameter passed (by ShareScope) telling you why ShareScope is initialising the study. This should be compared to the constants defined on the `ChartStudy` object (e.g. `Adding`).

Returns

You should return a value of `false` if you don't want the study to be added (or replaced). See below for details.

Description

ShareScope invokes a study's `init()` method when an `ChartStudy` object is created by ShareScope. It is guaranteed to be called before any other method call. You do not need to supply an `init()` method if you do not need one.

In general there are few practical differences in the use of a study's `init()` method compared to the `init()` method of columns and indicators. The `status` parameter passed to `init()` will be:

- `Adding` when the user adds a new study to the chart setting. Returning `false` from `init()` will cause the study not to be added.
- `Editing` when the user has indicated they wish to edit a study. Returning `false` will cause the edit operation to be cancelled.
- `Loading` when ShareScope is creating a new `ChartStudy` object belonging to a chart window. A study should not return `false` from `init()` in this case.

Example

Please refer to the `Column.init()` entry for a detailed example, including the use of dialog boxes for user input, the storage area, and appropriate handling of the status parameter.

ChartStudy.Layer

constants to specify a drawing layer

Synopsis

`Layer.`*`constant`*

Constants

`Bottom`	The bottom drawing layer (beneath the bars and any analytics).
`Top`	The top drawing layer (above the bars and any analytics).

See Also

`ChartStudy.setLayer()`

ChartStudy.lineTo()

draws a line, or adds a line segment to a path

Synopsis

`lineTo(x, y)`

Arguments

`x`	The line's end point (x-coordinate). See `ChartStudy.moveTo()` for the different ways an x-coordinate can be specified.
`y`	The line's end point (y-coordinate).

Returns

An integer giving the handle of the line (which can be used to later undraw the line). If you are currently drawing a path, the return value is `undefined`.

Description

The `lineTo()` function draws a line from the current cursor position to the point specified by the `lineTo()` parameters. The line is drawn using the current pen.

`lineTo()` is also used when drawing a path (see the `beginPath()` function) to add a line segment to the current path.

Examples

The example below draws a line joining the high of the first bar and the low of the second bar:

```
moveTo(0, bars[0].high);
lineTo(1, bars[1].low);
```

See Also

`ChartStudy.moveTo()`, `ChartStudy.beginPath()`, `ChartStudy.setPenColour()`

ChartStudy.moveTo()

moves the drawing cursor

Synopsis

`moveTo(x, y)`

Arguments

`X`	The new x-axis position can be specified in any one of the following ways: (i) as a number, giving the position as a bar index (ii) as a JavaScript `Date` object (iii) as a 2 element array of [dateNum, timeNum]
`Y`	A number giving the new y-axis position

Description

The `moveTo()` function moves the cursor position which acts as a starting point for a line drawn using the `lineTo()` function. Note that there is only a single cursor across all chart frames.

The x-coordinate of the new position can be specified using any one of three methods. The simplest (and fastest for ShareScope to plot) is to use a bar index. If this is an integer value, it will correspond to the centre of a bar. e.g. 0 is the centre of the first bar, 1 is the centre of the second bar, and 0.5 is the point midway between the centres of the two bars (note that, consistent with this scheme, a value of –0.5 will give you a point to the left of the first bar, should you need it).

The other two methods allow date/time based plotting. Firstly, you can use JavaScript `Date` objects – unfortunately these are slow. Alternatively use a 2-element array of dateNum and timeNum – this is a much faster way to plot date/time based data.

Examples of all 3 methods are given below.

Note that if you are currently drawing a path (see the `beginPath()` function), issuing a `moveTo()` command will not end the path, but will delete any existing path contents.

Examples

The examples below move the cursor to the high of the first bar on the chart, using the 3 alternative methods for specifying the x-coordinate.

```
moveTo(0, bars[0].high);                              // using a bar index
moveTo(bars[0].date, bars[0].high);                   // using a Date object
moveTo([bars[0].dateNum, bars[0].timeNum], bars[0].high);// using dateNum,timeNum
```

See Also

`ChartStudy.lineTo()`, `ChartStudy.beginPath()`

ChartStudy.onBarClose()

method invoked when a complete bar is formed on the chart

Synopsis

`function onBarClose(preExisting)`

Arguments

`preExisting`	This parameter will be `true` if the bar existed in ShareScope's database when

onNewChart() was called.

Description

ShareScope will call a study's onBarClose() method once, in time order, for every complete bar on the chart. You can access information about the current bar using the built-in bar and barIndex properties which are defined when this method is called. You can also use the built-in bars array to access information about any bar on the chart from onBarClose().

A *complete* bar is a bar which has stopped accumulating data – i.e. its OHLCV values are fixed and will not change. During market hours, the rightmost (i.e. latest) bar on an intraday chart is generally not a complete bar. onBarClose() will be called for these bars only when they become complete (you can use onNewBarUpdate() if you want to be informed about new and updated *partial* bars).

After calling onNewChart(), ShareScope will call onBarClose() for each of the complete bars that already exist in ShareScope's database. ShareScope passes a boolean value (preExisting) to onBarClose() which will be true, to indicate that these bars were already present in the bars array when ShareScope called onNewChart().

If you are connected to the intraday feed, additional bars will be formed on the chart every few minutes (depending on the bar period selected). ShareScope will call onBarClose() for these new bars as they become complete. In this case, preExisting will be false, to indicate that this is a new bar, that was not present in the bars array when onNewChart() was called.

A full introduction to using the onNewChart(), onBarClose() and onNewBarUpdate() methods together can be found in the *ShareScript User Guide*.

Example

The following script shows how to use bar, bars and barIndex in onBarClose() to colour each bar based on its close relative to the previous bar's close.

```
function onBarClose()
{
        if (barIndex == 0)    // the first bar has no previous bar, so return
              return;
        if (bar.close > bars[barIndex-1].close)
              bar.colour = Colour.Green;
        else
              bar.colour = Colour.Black;
}
```

See Also

ChartStudy.onNewBarUpdate(), ChartStudy.bar, ChartStudy.barIndex, ChartStudy.bars

ChartStudy.onMouseClick()

method invoked when the user clicks on the chart

Synopsis

```
function onMouseClick(frame, date, value, altValue)
```

Arguments

frame	A parameter passed (by ShareScope) telling you which frame of the chart the user clicked in. This can be compared to the constants defined by ChartStudy.Frame.
date	A JavaScript date object indicating the date/time where the click occurred (x-

	axis value).
`value`	A number indicating where on the value (y-axis) the user clicked. This will normally be a price (when the frame is `Frame.Chart`).
`altValue`	If the script defines an alternative y-axis range, this will return the alternative y-axis value. Otherwise, this value will be `undefined`.

Returns

You should return `true` if you handled the user's mouse click in your script, `false` if you didn't. Returning `false` allows ShareScope to respond to the mouse click.

Description

ShareScope will call a study's `onMouseClick()` method if the user clicks in the chart, and the study has input focus. The user can give a study input focus by clicking on the study's panel.

ShareScope passes a number of parameters to the method call that will inform the script exactly where the user clicked.

Additionally, the study's `bar` and `barIndex` properties will also be defined if the user clicked on one of the chart's bars (or `undefined` if not). See `ChartStudy.bar` for more information and the example below.

If your script handles the click, you should usually return `true` from this method. Your script should also normally provide some sort of visual feedback to the user in response to the click. Otherwise, return `false` to allow ShareScope's default click handler to be called.

Example

The following example shows a bar's close in the study's panel when the user clicks on a bar. Since it does not use any of the parameters passed to the `onMouseClick()` method, they are not required to be listed by the function declaration.

```
function onMouseClick()
{
	if (!bar)
		return false; // if the user didn't click on a bar
	else
	{
		setInfoText(bar.close);
		return true;  // tell ShareScope we handled the click
	}
}
```

See Also

`ChartStudy.Frame`, `ChartStudy.bar`, `ChartStudy.setAltRange()`

ChartStudy.onNewBarUpdate()

method invoked when a partial bar is added to the chart or the partial bar changes

Synopsis

`function onNewBarUpdate(preExisting)`

Arguments

`preExisting`	This parameter will be `true` if the bar existed in ShareScope's database when `onNewChart()` was called, and it has not changed since then.

Description

A study's `onNewBarUpdate()` method is called to tell you about changes to the newest partial bar at the right-hand side of the chart. It is called when a new partial bar is created or when that bar changes. It will be followed by a call to `onBarClose()` when the bar completes.

You can access information about the current partial bar using the built-in `bar` and `barIndex` properties which are defined when this method is called. You can also use the built-in `bars` array to access information about any bar on the chart from `onNewBarUpdate()`.

A full introduction to using the `onNewChart()`, `onBarClose()` and `onNewBarUpdate()` methods together can be found in the *ShareScript User Guide*.

See Also

`ChartStudy.onBarClose()`, `ChartStudy.bar`, `ChartStudy.barIndex`, `ChartStudy.bars`

ChartStudy.onNewChart()

method invoked when a new chart is about to be displayed

Synopsis

`function onNewChart()`

Description

ShareScope invokes an study's `onNewChart()` method when a new chart is about to be displayed. This happens when the user changes the charted instrument, or if the bar period changes (e.g. from 1 day bars to 1 week bars).

The chart is cleared of any previously drawn objects when ShareScope invokes `onNewChart()`. The study's panel however, is left unchanged.

The new chart's bars are available for inspection or manipulation through the `ChartStudy`'s `bars` property.

After `onNewChart()` returns it will be immediately followed by a call to `onBarClose()` for each complete bar in the `bars` array. If there is a partial (i.e. incomplete) bar at the end of the `bars` array, a call to `onNewBarUpdate()` will follow the calls to `onBarClose()`.

Example

The following example uses the `ChartStudy`'s `bars` array to search for highest price, then displays it on the study's panel:

```
function onNewChart()
{
	var max = 0;
	for (var i=0; i<bars.length; i++)
	{
		if (bars[i].high > max)
			max = bars[i].high;
	}
	setInfoText("max price="+max);
}
```

See Also

`ChartStudy.bars`, `ChartStudy.onBarClose()`

ChartStudy.onZoom()

method invoked when the user zooms into or out of the chart

Synopsis

`function onZoom()`

Description

ShareScope invokes a study's `onZoom()` method when the user changes the displayed bars by either zooming into (or out of) the chart using the mouse, or by selecting the "limit date range" command.

You can retrieve the index of the left-most visible bar using `getMinVisibleBarIndex()` and the right-most bar using `getMaxVisibleBarIndex()`.

Note that `onZoom()` is not called when you are connected to the intraday feed and new bars are added to an intraday chart. `onBarClose()` and `onNewBarUpdate()` will be called as the new bars are formed, and this will be reflected in the right-most bar index returned by `getMaxVisibleBarIndex()` if it is called from these methods.

See Also

`ChartStudy.getMinVisibleBarIndex()`, `ChartStudy.getMaxVisibleBarIndex()`

ChartStudy.setAltRange()

set an alternative y-axis for the current chart frame

Synopsis

`setAltRange(min, max)`

Arguments

`min`	A number giving the minimum value to be accommodated on the y-axis (corresponding to the bottom of the frame).
`max`	A number giving the maximum value to be accommodated on the y-axis (corresponding to the top of the frame).

Description

Each frame has a distinct primary y-axis, which ShareScope determines automatically from the range of data in the frame. You can also specify an alternative y-axis for the current target frame using the `setAltRange()` function.

Objects drawn by the script are normally plotted with reference to the primary y-axis of the target frame. When the alternative range is enabled (using the `useAltRange()` function), the y-coordinates of any drawn object are interpreted relative to the alternative y-axis, instead of the primary y-axis.

Note that the alternative y-axis is not shown on the chart, it is simply used to position objects drawn by the script within the chart frame.

Example

The example below sets up an alternative y-axis, running from 0 to 1. It then draws a line (centred on the first bar) from 10% above the bottom of the chart to 10% below the top.

The position of this line will remain constant within the frame, even when ShareScope chooses a new range for the frame's primary y-axis e.g. when zooming into the chart.

```
setAltRange(0, 1);
useAltRange(true);
moveTo(0, 0.1);
lineTo(0, 0.9);
```

See Also

`ChartStudy.useAltRange()`, `ChartStudy.setFrame()`

ChartStudy.setBrushColour()

set the brush colour for drawing

Synopsis

`setBrushColour(colour)`

Arguments

`colour`	An integer specifying the new brush colour.

Description

The `setBrushColour()` function sets the brush colour to be used for subsequent drawing commands. The brush is used to fill the interior of paths, shapes, symbols and text boxes. A study's default brush is a medium grey colour.

Examples

```
setBrushColour(Colour.Red);
setBrushColour(Colour.RGB(128,255,192));
```

See Also

`Colour`, `ChartStudy.setPenColour()`, `ChartStudy.setFillMode()`

ChartStudy.setButtonText()

change the caption of an existing button on the study's panel

Synopsis

`setButtonText(handle, caption)`

Arguments

`handle`	The handle of the panel button to change (an integer).
`caption`	The new label for the button (a string).

Returns

A boolean value – `true` if the button was found and changed, `false` otherwise.

Description

The `setButtonText()` function changes the caption of a previously created button on the study's panel.

See Also

`ChartStudy.createButton()`, `ChartStudy.deleteButton()`

ChartStudy.setFillMode()

selects whether a solid or transparent fill is used for drawing filled objects

Synopsis

`setFillMode(mode)`

Arguments

`mode`	A constant from the `ChartStudy.FillMode` object specifying the fill mode for subsequent drawing commands.

Description

When you use a drawing command that produces a filled shape or path, it will be filled using the current brush colour. The `setFillMode()` function allows you to also specify whether the fill is solid (i.e. opaque) or transparent.

By default, a study's fill mode is transparent, which allows other chart elements (e.g. the bars) to show through any filled objects you draw in your script.

Example

```
setFillMode(FillMode.Solid);
```

See Also

`ChartStudy.FillMode`, `ChartStudy.setBrushColour()`, `ChartStudy.fillShape()`, `ChartStudy.fillPath()`

ChartStudy.setFontColour()

set the font colour for drawing text

Synopsis

`setFontColour(colour)`

Arguments

`colour`	An integer specifying the new font colour.

Description

The `setFontColour()` function sets the font colour to be used for subsequent drawing commands. To change the font face and size, use `setFontStyle()`. A study's default font is black, Verdana, 9pt.

Examples

```
setFontColour(Colour.Red);
```

See Also

`Colour`, `ChartStudy.setFontStyle()`, `ChartStudy.drawText()`

ChartStudy.setFontStyle()

set the font style for drawing text

Synopsis

```
setFontStyle(name)
setFontStyle(name, size)
setFontStyle(name, size, colour)
```

Arguments

`name`	A string specifying the new font face.
`size`	Optional. An integer specifying the point size of the font (valid values range from 4 to 20). If not specified, the current font size is left unchanged.
`colour`	Optional. An integer specifying the new font colour. If not specified, the current font colour is left unchanged.

Description

The `setFontStyle()` function sets the font face, size and colour to be used for subsequent drawing commands. A study's default font is black, Verdana, 9pt.

Examples

```
setFontStyle("Times New Roman", 12, Colour.Black);
```

See Also

`Colour`, `ChartStudy.setFontColour()`, `ChartStudy.drawText()`

ChartStudy.setFrame()

set the target chart frame for drawing commands

Synopsis

```
setFrame(frame)
```

Arguments

`frame`	A constant from the `ChartStudy.Frame` object specifying the new target frame for subsequent drawing commands.

Description

The chart window is conceptually divided into multiple frames. The main price chart (containing the bars) is a frame, with the volume chart drawn beneath it in a separate frame. Indicators occupy further frames (though these cannot currently be drawn into from a study script). Each frame has a distinct primary y-axis, which ShareScope determines automatically from the range of data in the frame. The x-axis (date/time) is common to all frames.

The `setFrame()` function allows you to specify which frame subsequent drawing commands will target. The main price chart is selected by default.

Example

```
setFrame(Frame.Volume);      // target the volume frame
```

See Also

`ChartStudy.Frame`, `ChartStudy.setLayer()`

ChartStudy.setInfoText()

set the informational text on the study's panel

Synopsis

```
setInfoText()
setInfoText(s)
```

Arguments

`s`	An optional string used to set the study's informational text (displayed in the study's panel).

Description

The `setInfoText()` function allows you to set the study's informational text, which is displayed in the study's panel at the top left of the chart window. The text will wrap automatically in the available space, but can contain newline characters ('\n') if you wish to manually insert line breaks.

To remove the current text, either call `setIntoText()` with no parameter, or pass an empty string.

See Also

`ChartStudy.setTitle()`

ChartStudy.setLayer()

set the target layer for drawing commands

Synopsis

```
setLayer(layer)
```

Arguments

`layer`	A constant from the `ChartStudy.Layer` object specifying the new target layer for subsequent drawing commands.

Description

A study can draw objects on the chart on a layer either below or above the objects drawn by ShareScope.

The `setLayer()` function allows you to specify which layer subsequent drawing commands will target. By default, anything a study script draws will be drawn *on top* of the chart objects drawn by ShareScope.

Example

```
setLayer(Layer.Bottom);      // draw beneath the bars
```

See Also

`ChartStudy.Layer`, `ChartStudy.setFrame()`

ChartStudy.setPenColour()

set the pen colour for drawing

Synopsis

```
setPenColour(colour)
```

Arguments

`colour`	An integer specifying the new pen colour.

Description

The `setPenColour()` function sets the pen colour to be used for subsequent drawing commands. The pen is used to draw lines. It is also used to draw the outlines of paths, shapes, symbols and text boxes, with the interiors of these objects being filled using the current brush colour.

To change the pen's line-style as well as the colour, see `setPenStyle()`.

A study's default pen is solid black, minimum width. You can use `ChartStudy.getBackColour()` to ensure that any colours used will contrast with the chart window background.

Examples

```
setPenColour(Colour.Red);
setPenColour(Colour.RGB(128,255,192));
```

See Also

`Colour`, `ChartStudy.setPenStyle()`, `ChartStudy.getBackColour()`

ChartStudy.setPenStyle()

sets the pen style for drawing

Synopsis

```
setPenStyle(pen)
setPenStyle(pen, width)
setPenStyle(pen, width, colour)
```

Arguments

`pen`	A constant from the `Pen` object specifying the type of pen to use.
`width`	Optional. An integer giving the width of the pen. Valid values are 0 to 7. If not specified this defaults to 0 (the thinnest line). Greater widths are only allowed for a pen type of `Pen.Solid`.
`colour`	Optional. An integer specifying the new pen colour. If not specified, the current pen colour is left unchanged.

Description

The `setPenStyle()` function sets the pen to be used for subsequent drawing commands. To change only the pen's colour, use `setPenColour()` instead. A study's default pen is solid black, minimum width.

Example

```
setPenStyle(Pen.Dash, 0, Colour.RGB(128,255,255));
```

See Also

`Colour`, `Pen`, `ChartStudy.setPenColour()`

ChartStudy.setTitle()

sets the study's title

Synopsis

`setTitle(s)`

Arguments

`s`	A string providing a name for the study

Description

The `setTitle()` function allows you to set the study's title, which is displayed in the study's panel at the top left of the chart window.

See Also

`ChartStudy.setInfoText()`

ChartStudy.Shape

constants to specify a shape to draw

Synopsis

`Shape.`*`constant`*

Constants

`Ellipse`	An ellipse.
`Rectangle`	A rectangle.
`Diamond`	A diamond.
`Arrow`	An arrow.

See Also

`ChartStudy.drawShape()`

ChartStudy.Symbol

constants to specify a symbol to draw

Synopsis

`Symbol.`*`constant`*

Constants

`Circle`	A circle.
`Square`	A square.
`TriangleUp`	A upwards pointing triangle.

`TriangleDown`	A downwards pointing triangle.
`Cross`	A cross.
`FlagUp`	An upwards pointing “flag” symbol, designed to be drawn with a single letter inside (used in ShareScope to indicate CandleStick patterns).
`FlagDown`	An downwards pointing “flag” symbol, designed to be drawn with a single letter inside (used in ShareScope to indicate CandleStick patterns).

See Also

`ChartStudy.drawSymbol()`

ChartStudy.TextAlign

constants to specify multi-line text alignment within a text box

Synopsis

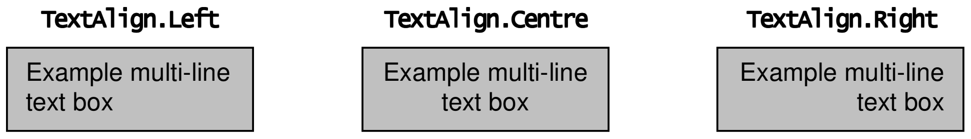
`TextAlign.`*`constant`*

Constants

`Left`	The text is left-aligned within its bounding box.
`Centre/Center`	The text is centred within its bounding box.
`Right`	The text is right-aligned within its bounding box.

Description

These constants specify text alignment with the text’s bounding box, as illustrated below:


TextAlign.Left
Example multi-line text box
TextAlign.Centre
Example multi-line text box
TextAlign.Right
Example multi-line text box

See Also

`ChartStudy.drawText()`

ChartStudy.undrawItem()

removes a single previously drawn object

Synopsis

`undrawItem(handle)`

Arguments

`handle`	An integer providing the handle of a previously drawn item

Description

The `undrawItem()` function can be used to remove a single drawn item from the chart. It requires the handle returned when the item was originally drawn e.g. by the `lineTo()` or `drawText()` functions.

If you are drawing complex figures composed of multiple drawn objects, and you later wish to remove a whole figure, consider accumulating individual handles in an array.

To remove all drawn items from the chart, use `clearDisplay()` instead.

Example

```
moveTo(0, bars[0].close);
var lineHandle = lineTo(1, bar[1].close);  // store the line's handle for later
    :       :
undrawItem(lineHandle);                    // remove the previously drawn line
```

See Also

`ChartStudy.clearDisplay()`

ChartStudy.useAltRange()

enable/disable plotting using an alternative y-axis

Synopsis

`useAltRange(enable)`

Arguments

`enable`	A boolean value: `true` to enable the use of any alternative y-axis, `false` to use the primary y-axis. Applies to all subsequent drawing commands.

Description

The `useAltRange()` function enables (and disables) plotting using the alternative y-axes across all frames. When this mode is enabled, all subsequent drawing commands are assumed to be specifying y-coordinates using the current frame's alternative y-axis range, specified using the `setAltRange()` function.

See Also

`ChartStudy.setAltRange()`, `ChartStudy.setFrame()`

Storage Objects Reference

Storage

ShareScript v1.1

persistent storage for columns, indicators and chart studies

Object→Storage

Construction

A `Storage` object is defined as a property of ShareScript `Column`, `Indicator` and `ChartStudy` objects (note that the property name is lower case). `Storage` objects can not be created using the normal JavaScript new() operator.

Methods

`getSize()`	Returns the number of slots in the storage area.
`getAt()`	Returns the number in a slot.
`setAt()`	Stores a number in a slot.

Description

`Storage` objects provide access to a persistent storage area for ShareScript columns, indicators and studies. Data stored in this area is available to your scripts even when ShareScope has been closed and then restarted. Along with `Dialog` objects, `Storage` objects allow you to create columns and indicators that behave just like the ones built-in to ShareScope.

At present eight slots are provided for storing data in columns and indicators. Thirty-two slots are provided in studies. Each slot can hold a JavaScript `Number` value. It is not possible to store strings. The `File` class provides an alternative, if you need to write textual data to a file.

Note that since ShareScope stores the data in its configuration files using IEEE floats, the values may lose some precision during the storage and retrieval process. However, this is unlikely to be encountered with normal use.

Example

A full example of storage area usage can be found in the `Column` object section.

See Also

`Column`, `Indicator`, `ChartStudy`, `File`

Storage.getSize()

get the total number of slots

Synopsis

storage.`getSize()`

Returns

An integer giving the number of slots in the storage object.

Storage.getAt()

get the value at a specified slot

Synopsis

storage.getAt(i)

Arguments

i	An integer specifying the slot number. Numbering runs from 0 (the first slot) to getSize()-1.

Throws

RangeError	If an invalid slot is requested.

Returns

A Number giving the current value at slot i. The value will be undefined if the slot has not yet been assigned a value.

Storage.setAt()

set the value at a specified slot

Synopsis

storage.setAt(i)

Arguments

i	An integer specifying the slot number. Numbering runs from 0 (the first slot) to getSize()-1.

Throws

RangeError	If an invalid slot is requested.