Contents

1 GT::Analyzers

Provides some functions that will be used by all analyzer modules.

DESCRIPTION

MANAGE A REPOSITORY OF INDICATORS

```
GT::Analyzers::get_registered_object($name);
GT::Analyzers::register_object($name, $object);
GT::Analyzers::get_or_register_object($name, $object);
GT::Analyzers::manage_object(\@NAMES, $object, $class, $args, $key);
```

DEFAULT FUNCTIONS FOR ANALYZERS

`GT::Analyzers::Module->new($args, $key, $func)`

> Create a new analyzer with the given arguments. $key and $func are optional, they are useful for indicators which can use non-usual input streams.

`$analyzers->initialize()`

> Default method that does nothing.

`GT::Analyzers::Accumulate - Accumulates the Days of arg1`

2 DESCRIPTION

Accumulates the values of the array that is given as arg1.
This means if the array consists of the values (1,2,3,4) the result is (1,3,6,10).

Parameters

First argument: Array reference to be accumulated

Return

Returns an array, not considering first and last parameter.

`GT::Analyzers::Avg - Calculates the Average of arg1`

3 DESCRIPTION

Calculates the Average of arg1

Parameters

First argument: Array reference to be averaged

Return

Returns an array, not considering first and last parameter.

GT::Analyzers::AvgCosts - Average Costs per trade

4 DESCRIPTION

The mean costs of the portfolio

GT::Analyzers::AvgGain - Average Gain per trade

5 DESCRIPTION

This analyzer calculates the mean of the gains.

Parameters

First parameter: Cumulated Gain
Second parameter: Number of Gains

GT::Analyzers::AvgLoss - Average Loss per trade

6 DESCRIPTION

This analyzer calculates the mean of the losses.

Parameters

First parameter: Cumulated Loss
Second parameter: Number of Losses

GT::Analyzers::AvgNZ - Calculates the Average

7 DESCRIPTION

Calculates the Average of Arg1, but only of the days where Arg1 is non-zero.

Parameters

First argument: Array reference to be averaged

`GT::Analyzers::AvgPerformance - Average Performance per trade`

8 DESCRIPTION

The mean performance of the portfolio

Parameters

none

GT::Analyzers::AvgCosts - Average Costs per trade

9 DESCRIPTION

The mean performance of the portfolio

Parameters

none

GT::Analyzers::BuyPrice - The price for which the stock was bought

10 DESCRIPTION

The price for which the stock was bought

Parameters

none

GT::Analyzers::CloseDate - The date where the position was closed

11 DESCRIPTION

The date where the position was closed.

Parameters

none

GT::Analyzers::ClosePricce - The price on the closing date

12 DESCRIPTION

The price on the closing date.

Parameters

none

GT::Analyzers::CompleteCash - Returns the cash-part of the evaluation-history

13 DESCRIPTION

Returns the cash-part of the evaluation-history

Parameters

none

GT::Analyzers::CompleteDate - Returns all dates of the evaulation history

14 DESCRIPTION

Returns all dates of the evaulation history

Parameters

none

GT::Analyzers::CompleteGain - The gain of the evaluation history

15 DESCRIPTION

The gain of the evaluation history

Parameters

none

GT::Analyzers::Complete Value - The values of the portfolio history

16 DESCRIPTION

The values of the portfolio history

Parameters

none

GT::Analyzers::Consec - Maximum of consecutive nonzero-values

17 DESCRIPTION

Calculates the Average of Arg1

Parameters

First argument: Array reference to be averaged

GT::Analyzers::Costs - Costs per trade

18 DESCRIPTION

The costs per trade.

Parameters

none

GT::Analyzers::CumGain - Cummulative Gain

19 DESCRIPTION

Cummulative Gain

Parameters

none

GT::Analyzers::CumLoss - Cummulative Loss

20 DESCRIPTION

Cummulative Loss

Parameters

none

GT::Analyzers::DrwaDown - The Drawdown of the portfolio

21 DESCRIPTION

The Drawdown of the portfolio: The maximum Drawdown can be calculated as: {A:Min {A:DrawDown}}

Parameters

NetGain

Initial Sum of Cash

Maximum net gain

GT::Analyzers::Duration - Duration of the trades

22 DESCRIPTION

Duration of the trades.

Parameters

none

GT::Analyzers::First - First value of the arrray #arg1

23 DESCRIPTION

First value of the arrray #arg1

Parameters

First argument: Array reference

GT::Analyzers::Gain - Gain of each trade

24 DESCRIPTION

Gain of each trade

Parameters

none

GT::Analyzers::GrossGain - The gross gain

25 DESCRIPTION

The gross gain

Parameters

none

GT::Analyzers::InitSum - The initial amount of cash

26 DESCRIPTION

The initial amount of cash

Parameters

none

GT::Analyzers::AvgCosts - Average Costs per trade

27 DESCRIPTION

The mean performance of the portfolio

Parameters

none

GT::Analyzers::AvgCosts - Average Costs per trade

28 DESCRIPTION

The mean performance of the portfolio

Parameters

none

`GT::Analyzers::IsLoss - Boolean value: True if it is a loosing trade`

29 DESCRIPTION

Boolean value: True if it is a loosing trade

Parameters

none

GT::Analyzers::Last - Last value of array #arg1

30 DESCRIPTION

Last value of array #arg1

Parameters

First argument: Array reference

`GT::Analyzers::Long - Boolean value: True if trade is long`

31 DESCRIPTION

Boolean value: True if trade is long

Parameters

none

GT::Analyzers::Losses - The losses of the trades

32 DESCRIPTION

The losses of the trades

Parameters

none

GT::Analyzers::Max - Calculates the Maximum of Arg1

33 DESCRIPTION

Calculates the Maximum of Arg1

Parameters

First argument: Array reference

GT::Analyzers::MeanPerformace - The Mean Performance of a Portfolio

34 DESCRIPTION

The mean performance of the portfolio

Parameters

none

GT::Analyzers::Min - Calculates the Minimum of Arg1

35 DESCRIPTION

Calculates the Minimum of Arg1

Parameters

First argument: Array reference

GT::Analyzers::NB - Number of trades

36 DESCRIPTION

Number of trades

Parameters

none

GT::Analyzers::NetGain - The net gain of the positions

37 DESCRIPTION

The net gain of the positions

Parameters

none

GT::Analyzers::AvgCosts - Average Costs per trade

38 DESCRIPTION

The mean performance of the portfolio

Parameters

none

`GT::Analyzers::OpenDate - The date where the trade was opened`

39 DESCRIPTION

The date where the trade was opened

Parameters

none

GT::Analyzers::OpenPrice - The price at the opening

40 DESCRIPTION

The price at the opening

Parameters

none

GT::Analyzers::PerShare - Normalizes a value per share of a position

41 DESCRIPTION

Normalizes a value per share of a position

Parameters

First argument: Array reference to be normalized

42 DESCRIPTION

The Performance of the trades

Parameters

The net gain in percent

The price ath the opening of a trade

GT::Analyzers::Process

43 DESCRIPTION

This module offes all those functions that are needed by the analyzers shell to interactively analyze and test portfolios.

FUNCTIONS

`GT::Analyzers::Process->new()`

`GT::Analyzers::Process->parse( $cmd )`

This function parses the command $cmd. If the shell is set in expert mod it tries first to map $cmd to an internat command and otherwise evaluates it using eval.

`GT::Analyzers::Process->bye()`

Exits the program after asking if the history should be stored.

`GT::Analyzers::Process->set( [ $key ] )`

Set a configuration-parameter. If key is not given, the list of parameters is given. The variable key consits of the real key and the value separated by a space. If you want to set an array, you can either use key[x] to teh xth element or +key to add the value to the array.

`GT::Analyzers::Process->set_code( $code )`

Set the code and do the necessary initialization-stuff.

`GT::Analyzers::Process->load( $sys, $dir, $code )`

Loads $sys from $dir and $code (optional).

`GT::Analyzers::Process->save( $sys, $dir )`

Saves the portfolio with name $sys to directory $dir.

`GT::Analyzers::Process->list( $dir )`

Lists the systems in directory $dir.

`GT::Analyzers::Process->btest()`

start the backtest.

`GT::Analyzers::Process->calc($args)`

Calculates the expression given as argument(s).

`GT::Analyzers::Process->calc_array($arg1, $arg2, ...)`

Calculates each array and prints out/returns a list.

The arry should have the same length.

`GT::Analyzers::Process->p($arg)`

Prints out the string $arg and replaces the variable elements

`GT::Analyzers::Process->help()`

Print the help screen

`GT::Analyzers::Process->licence()`

Print the licence

`GT::Analyzers::Process->license()`

Prints the license

`GT::Analyzers::Process->disconnect()`

Disconnect from database.

`GT::Analyzers::Process->info()`

Prints a small information shown at the start of anashell.

`GT::Analyzers::Process->pg_hist()`

Uncomment this function to plot histograms with pgplot.

`GT::Analyzers::Process->r_hist( $array )`

Plots a histogram using R (www.r-project.org) of the values of $array.

`GT::Analyzers::Process->r_bar( $array )`

Generates a barplot in R by using the values of $array

`GT::Analyzers::Process->r_corr( $arr1, $arr2 )`

Plots the correlation of $arr1 and $arr2 in R.

`GT::Analyzers::Process->report( $file )`

Prints the report of the portfolio using $file as template.

GT::Analyzers::Profitfactor - Calculates the profitfactor

44 DESCRIPTION

Calculate the Profitfactor by dividing the sum of the gains by the sum of the losses.

Parameters

Sum of the Gains

Sum of the Lossse

GT::Analyzers::Profitfactor - Calculates the profitfactor

45 DESCRIPTION

Calculate the Profitfactor by dividing the average of the gains by the average of the losses.

Parameters

Average of the Gains

Average of the Lossse

`GT::Analyzers::Quantity - The quantitiy of shares`

46 DESCRIPTION

The quantitiy of shares

Parameters

none

GT::Analyzers::AvgCosts - Average Costs per trade

47 DESCRIPTION

The mean performance of the portfolio

Parameters

none

48 GT::Analyzers Report

DESCRIPTION

This module is mainly a wrapper to process a report file with HTML::Mason.

This module needs HTML::Mason to display the reports and Cwd and File::Spec to find out the actual path.

GT::Analyzers::RiskReturn - Caluclates the Risk-/Return-Ratio

49 DESCRIPTION

Caluclates the Risk-/Return-Ratio

Parameters

First argument: The portfolio-history.

GT::Analyzers::SellPrice - The price for which the position was sold

50 DESCRIPTION

The price for which the position was sold

Parameters

none

GT::Analyzers::Short - True if it is a short trade

51 DESCRIPTION

True if it is a short trade

Parameters

none

GT::Analyzers::StdTime - Normalizes the value #arg1 per year

52 DESCRIPTION

Normalizes the value #arg1 per year

Parameters

First argument: Value to be normalizes

`GT::Analyzers::Sum - Summarizes the array #arg1`

53 DESCRIPTION

Summarizes the array #arg1

Parameters

First argument: Array reference to be summarized

54 DESCRIPTION

The Sum of the Performance

Parameters

Sum of the Gains

Initial Sum

GT::Analyzers::Type - String: long if long and short if short

55 DESCRIPTION

String: long if long and short if short

Parameters

none

GT::Analyzers::WinRatio - Calcuates the WinRatio

56 DESCRIPTION

Calcuates the WinRatio

Parameters

Number of Gains

Number of Losses

57 GT::ArgsTree

Represent the arguments of calculation objects (indics/signals/systems)

DESCRIPTION

Each calculation object can be parameterized with arguments. But those arguments can themselves be calculation objects. This is represented by a complex syntax that this module can understand and use to create a tree of arguments.

SYNTAX

The argument list is a space separated list of arguments. However when the argument is not a readable value but a computable one, it should be given with a different syntax :

```
{ I::Indicator <indic_arg_list> }
```

AVAILABLE FUNCTIONS

`GT::ArgsTree->new(@args)`

Create an ArgsTree object for the given list of arguments. Instead of a list you can give a string representation of all the arguments.

`$at->add_args(@args)`

Process the list of arguments and adds them to the arguments tree.

`$at->create_objects()`

Creates the required objects to compute the various arguments.

`$at->is_constant($arg_number)`

`$at->is_constant()`

Return true if the corresponding argument is of constant value (ie it doesn't have to be computed each time). If no argument is given, then return true if all arguments are constant.

The first argument is numbered "1" (and not "0").

`$at->get_arg_values($calc, $day)`

`$at->get_arg_values($calc, $day, $n)`

Return the (computed) value of the indicated argument. Returns the list of values of all arguments if no parameter is given.

The first argument is numbered "1" (and not "0").

`$at->get_arg_constant($n)`

Return the constant value of the given argument. Make sure to check that the argument is constant before otherwise it will die.

`$at->get_arg_object($n)`

Return the associated object of the given argument. The object is something able to compute the value of the argument. Make sure the argument is not a constant otherwise it will die.

`$at->get_arg_names()`

`$at->get_arg_names($n)`

Return the name the indicated argument. Returns the list of names of all arguments if no parameter is given.

The first argument is numbered "1" (and not "0").

`$at->get_nb_args()`

Return the number of arguments available.

`my ($full_name, @args) = GT::ArgsTree::parse_args($args)`

Parse the arguments in $args and return the parsed content in the form of two arrays (list of arguments).

`GT::ArgsTree::args_to_ascii(@args)`

Return the ascii representation of all the parameters described in @args.

`$args->prepare($calc, $day)`

Precalculate all possible values for the given day.

`$args->prepare_interval($calc, $first, $last)`

Precalculate all possible values for the given interval.

58 GT::BackTest

Backtest trading systems in different conditions

OPTIONS

Analysis::ReferenceTimeFrame

You can set this configuration item to day, week, month or year. There will be a standardized performance result based on that timeframe. By default the value is "year".

DESCRIPTION

`backtest_single($pf_man, $sys_man, $broker, $calc, $first, $last)`

Backtest the a system using the given portfolio manager (ie set of money management rules) on the data contained by $calc during the period $first to $last (indices used by the calculator).

`GT::BackTest::combinate_system_and_manager(\@systems, \@managers)`

Returns a hash that can be used by backtest_combinations

`GT::BackTest::create_managers_with_filters(\@filters)`

Create all possible managers with all the possible combinations of filters.

59 GT::BackTest::Spool

This modules provides some functions to manage a backtest directory.

$spool = GT::BackTest::Spool->new($data_directory);

Create and initialize a BackTest::Spool object with a specific directory, where backtest data are stored.

$spool->use_cache(0|1)

Tell if data are cached before being written. Default to 1. In that case you have to call $spool->sync from time to time to write data on the disk.

$spool->update_index()

Force an update of the index data. Use that if a long time has elapsed since the read and the index may have been updated.

$spool->add_alias_name($sysname, $alias);

This function will link an alias and a system name.

$spool->get_alias_name($sysname)

Return the alias name of the system if it exists.

$spool->add_results($sysname, $code, $stats, $portfolio, [$set]);

This function will add new data or update old ones in the spooler.

$spool->sync()

Write the cache on disk.

$hash = $spool->list_available_data([$set]);

This function will return a list of systems/codes available. $hash->{$sysname} = [list of codes];

$spool->get_stats($sysname, $code);

This function will return all stats available for a given $sysname and $code.

$spool->get_portfolio($sysname, $code);

This function will return a portfolio for a given $sysname and $code.

60 GT::Brokers

A module for calculating broker's fee & commissions

DESCRIPTION

Brokers rules are used to calculate commissions for each buy/sell order, as well as annual account charge.

`$broker->calculate_order_commission($order)`

Return the amount of money ask by the broker for the given order.

`$broker->calculate_annual_account_charge($portfolio, $year)`

Return the amount of money ask by the broker for the given year according to the given portfolio.

61 GT::Brokers::Cortal

Overview

This module will calculate all commissions and charges according to Cortal rules.

Calculation

For orders "A tout prix" and "prix du marché" :
5 Euros HT / order up to 1500 Euros 10 Euros HT / order up to 3000 Euros 0.30 % / order up to 100 000 Euros + 0.10 % / after
For other orders :
7.5 Euros HT / order up to 1500 Euros 12.5 Euros HT / order up to 3000 Euros 0.50 % / order up to 100 000 Euros + 0.10 % / after
Annual account charge (30/06 and 31/12) :
Up to 150 000 Euros : 0.15 % HT of the portfolio value, after 0 % Minimum : 12 Euros HT.

$broker->calculate_order_commission($order)

Return the amount of money ask by the broker for the given order.

$broker->calculate_annual_account_charge($portfolio, $year)

Return the amount of money ask by the broker for the given year according to the given portfolio.

62 GT::Brokers::Dubus

Overview

This module will calculate all commissions and charges according to Dubus rules.

Calculation

Tarif Normal :

```
4.9 Euros HT / order up o 2000 Euros + 0.30 % HT after (since 26/05/2005 4.9 Ă i

Account charge : 0.37 Ă HT (min 100 Ă)
```

Tarif Forfait :

```
4.9 Euros HT / order up o 2000 Euros + 0.30 % HT after (since 26/05/2005 4.9 Ă i
15 Ă Euros HT / order up to 30000 Euros
25 Ă Euros HT / order up to 75000 Euros
50 Ă Euros HT / more than 75000 Euros

Account charge : 76 Ă
```

Parameters

The first parameter could be initialized to : "Normal" => Tarif Normal "Forfait" => Tarif Forfait

$broker->calculate_order_commission($order)

Return the amount of money ask by the broker for the given order.

$broker->calculate_annual_account_charge($portfolio, $year)

Return the amount of money ask by the broker for the given year according to the given portfolio.

63 GT::Brokers::InteractiveBrokers

Overview

This module will calculate all commissions and charges according to InteractiveBrokers rules.

Calculation

Current calculation for InteractiveBrokers at: http://www.interactivebrokers.com/index.html?html/retailAcco

Germany XETRA/IBIS: 0,1% of stock value, minimum of 4 EUR, maximum of 29 EUR

Switzerland: 0,1% of stock value, minimum of 10 CHF + 0.07% Stamp Tax

UK: 0,1% of stock value, minimum of 5 GBP + 0.5% UK Stamp Tax on purchase

Ireland: same as UK, but 1% Irish Stamp Tax

US: USD 0.01 / share, up to 500 shares USD 0.005 / share, for 501th share and up minimum of 1 USD

Options and futures commissions are not considered.

No annual charge.

Parameters

The first parameter could be initialized to :

'de' => Germany Xetra, 'ch' => Switzerland, 'ie' => Ireland, 'uk' => United Kingdom, 'us' => US Markets

$broker->calculate_order_commission($order)

Return the amount of money ask by the broker for the given order.

$broker->calculate_annual_account_charge($portfolio, $year)

Return the amount of money ask by the broker for the given year according to the given portfolio, which is 0 EUR.

64 GT::Brokers::Logitelnet

Overview

This module will calculate all commissions and charges according to Logitelnet (Societe Generale) rules

Calculation

0.54% / order < 8000 Euros, 8.90 Euros minimum 0.44% / 8000 <= order < 15000 Euros 0.34 % / order up to 15000 Euros

$broker->calculate_order_commission($order)

Return the amount of money ask by the broker for the given order.

$broker->calculate_annual_account_charge($portfolio, $year)

Return the amount of money ask by the broker for the given year according to the given portfolio.

65 GT::Brokers::NoCosts

Overview

This module will calculate no commissions or charges.

$broker->calculate_order_commission($order)

Return the amount of money ask by the broker for the given order.

$broker->calculate_annual_account_charge($portfolio, $year)

Return the amount of money ask by the broker for the given year according to the given portfolio.

66 GT::Brokers::SelfTrade

Overview

This module will calculate all commissions and charges according to SelfTrade rules.

Calculation

Forfait Découverte : 6.5 Euros HT / order up o 3000 Euros + 0.30 % HT after
Forfait Intégral : 14.95 Euros HT / order up to 10000 Euros + 0.15 % HT after
For both options, there's no annual account charge !

Parameters

The first parameter could be initialized to : "Découverte" => Forfait Découverte "Intégral" => Forfait Intégral

$broker->calculate_order_commission($order)

Return the amount of money ask by the broker for the given order.

$broker->calculate_annual_account_charge($portfolio, $year)

Return the amount of money ask by the broker for the given year according to the given portfolio.

67 GT::Brokers::Usaa

Overview

This module will calculate all commissions and charges for the purchase or sale of stock on an exchange according to Usaa brokerage charge schedules.

Calculation

For all orders:

```
US$21.95 + $0.02 * ( quantity of shares - 1000 ) + $3.00
```

where US$3.00 is the exchange fee, and the charge of US$0.02 for shares in excess of 1000. There is no annual account charge.

$broker->calculate_order_commission($order)

Return the calculated broker's commission for the given order.

$broker->calculate_annual_account_charge($portfolio, $year)

Returns the amount of money asked by the broker for the given year according to the given portfolio, which is $0 in the case of USAA Brokerage.

68 GT::Brokers::Zebank

Overview

This module will calculate all commissions and charges according to Zebank rules.

Calculation

0,45 % TTC / order with a minimum of 9 EUR
    Free return if buy & sell during the same day.

$broker->calculate_order_commission($order)

Return the amount of money ask by the broker for the given order.

$broker->calculate_annual_account_charge($portfolio, $year)

Return the amount of money ask by the broker for the given year according to the given portfolio.

69 GT::CacheValues

Cache the computed values (of indic/signals) for a single share

DESCRIPTION

This object is designed to be associated with a GT::Prices object. It may contain the computed value of some indicators corresponding to the GT::Prices object.

`my $cache = GT::CacheValues->new;`

Create a new GT::CacheValues that will contain computed values of some indicators or signals.

`$cache->get($name, $i)`

Return the value of the indicator $name for the day $i.

`$cache->set($name, $i, $value)`

Store the computed value $value of indicator $name for the day $i.

`$cache->is_available($name, $i)`

`$cache->is_available_interval($name, $first, $last)`

Check if the value of indicator $name is available for day $i.

70 GT::Calculator

All data (of a single share) used for calculations

DESCRIPTION

This is a facility object to ease the collaboration between GT::Prices and GT::CacheValues. It contains the prices (GT::Prices), and the result of various indicators and signals within two GT::CacheValues object. This object is manipulated by all the indicators, signals and systems.

A calculator can contain the same serie of prices but indexed on different time frames.

`my $c = GT::Calculator->new($prices [, $code])`

Create a new GT::Calculator object with $prices used for calculations. The calculator is associated to share named $code.

`$c->prices()`

`$c->indicators()`

`$c->signals()`

Return the corresponding object of the current timeframe.

`$c->prices_on_timeframe($timeframe)`

`$c->indicators_on_timeframe($timeframe)`

`$c->signals_on_timeframe($timeframe)`

Return the corresponding object of the indicated timeframe. Learn more about the timeframes in GT::DateTime.

`$c->set_code($code)`

Sets the code of the share which datas are stored in this object.

`$c->code()`

Returns the code of the share associated with this object.

`$calc->set_current_timeframe($timeframe)`

Changes the current timeframe to the indicated one. If the timeframe doesn't exist, it tries to create it. Returns 1 on success and 0 on failure.

`$calc->current_timeframe()`

Returns the current timeframe.

`$calc->create_timeframe($timeframe)`

Create the given timeframe with an other serie of prices available in the calculator. Returns 1 on success and 0 on failure.

`$calc->available_timeframe()`

Returns the sorted list of available timeframes.

`$calc->timeframe_is_available()`

Returns true if the given timeframe is available in the calculator. Otherwise returns false.

71 GT::CloseStrategy

Manages opened positions

DESCRIPTION

A CloseStrategy is more really a position manager. Once a system has opened a position, it's managed by a CloseStrategy. Managing means updating the stop and deciding when to close the position.

```
$cs->get_indicative_stop($calc, $i, $order, $pf_man, $sys_man)
```

```
$cs->get_indicative_long_stop($calc, $i, $order, $pf_man, $sys_man)
```

```
$cs->get_indicative_short_stop($calc, $i, $order, $pf_man, $sys_man)
```

> This function returns an indicative stop level that should be set for the indicated day. It is used before a position is opened to evaluate a stop level that may be used by a MoneyManagement rule.

```
$cs->position_opened($calc, $i, $position, $pf_man, $sys_man)
```

```
$cs->short_position_opened($calc, $i, $position, $pf_man, $sys_man)
```

```
$cs->long_position_opened($calc, $i, $position, $pf_man, $sys_man)
```

> Those functions are callback that are launched when a position has been opened. It can be used to place order on a target that will be valid until they are executed (ie no_discard=1). $cs->position_opened will call the right callback depending on the the position (short or long). It can also be used to set an initial stop level.

```
$cs->manage_position($calc, $i, $position, $pf_man, $sys_man)
```

```
$cs->manage_short_position($calc, $i, $position, $pf_man, $sys_man)
```

```
$cs->manage_long_position($calc, $i, $position, $pf_man, $sys_man)
```

> Manage an open position of the corresponding type. The position may be augmented or reduced by sending new orders modified by $manager->set_order_partial(...). The stop may be updated with $position->set_stop(...).

```
$system->precalculate_all($calc)
```

`$system->precalculate_interval($calc, $first, $last)`

If you run a system on a long period of time you may want to precalculate all the indicators in order to benefit of possible optimizations. This is the role of those 2 functions.

Functions to manage a repository of close strategies

```
GT::CloseStrategy::get_registered_object($name);
GT::CloseStrategy::register_object($name, $object);
GT::CloseStrategy::get_or_register_object($name, $object);
GT::CloseStrategy::manage_object(\@NAMES, $object, $class, $args, $key);
```

72 GT::CloseStrategy::ChannelBreakout

DESCRIPTION

This Channel Breakout exit strategy close a position once the lower level has been triggered for a long position, as well as the upper level for a short position.

73 GT::CloseStrategy::CloseGain

This strategy closes the position once the prices have crossed a limit called target. This target is defined as a percentage from the initial price. By default, it's defined as + 25 %. If you use it together with PartialGain, this strategy will only close the remaining shares to be sold/bought. Take care however to place the CloseGain strategy after the PartialGain strategy.

74 GT::CloseStrategy::OppositeSignal

This strategy closes the position once the opposite signal has been emitted by the system. It will will close a long position on a sell signal and close a short position on a buy signal.

Arguments

The arguments taken by this object are special. The first argument is the name of a TradeFilter to use as the condition. It may be followed by argument to give to the TradeFilter at creation time. After that, there's the name of the real CloseStrategy to apply. This strategy will only be applied if the trade filter accepts a fake "close order".

Examples or arguments :

...->new("TF:AroonTrend", "CS:Stop:SAR"); ...->new("TF:FollowTrend", 15, "CS:Stop:SAR", 0.05, 0.02, 0.02);

The system detects the end of the arguments of the TradeFilter once it detects "CS:" or "CloseStrategy:" at the beginning of the next argument.

75 CloseStrategy::Generic

DESCRIPTION

This is a simple Generic Closestrategy that closes the trade based on one or two signals.

Parameters

First Signal

The first Signal is the signal used to close a long position.

Second Signal

The second signal is used to close short positions.

76 GT::CloseStrategy::LimitPeriodInTheMarket

Only allow the trade to last for X days

DESCRIPTION

This strategy closes the position once the maximum time for the trade has been reached if the second signal is true.

The second/third parameter is a signal which can infirm the closing order of long/short position. In particular you may want to not close a position which looks like to be a great winner...

You confirm the order with a true value and infirm it with a false value.

EXAMPLES

Close a long position if after 3 days, the security hasn't increased. Close a short position if after 3 days, the security hasn't dropped.

```
 CS:LimitPeriodInTheMarket 3
    {S:Generic:Below {I:Prices CLOSE} {I:Generic:PeriodAgo 3 {I:Prices CLOSE}}}
    {S:Generic:Above {I:Prices CLOSE} {I:Generic:PeriodAgo 3 {I:Prices CLOSE}}}
=cut

   sub initialize { my $self = shift; $self->add_arg_dependency(2, 1); }
   sub long_position_opened { my ($self, $calc, $i, $position, $pf_manager,
$sys_manager) = @_;

    return;
}

   sub short_position_opened { my ($self, $calc, $i, $position, $pf_manager,
$sys_manager) = @_;

    return;
}

   sub manage_long_position { my ($self, $calc, $i, $position, $pf_manager,
$sys_manager) = @_; my $initial_period = $calc->prices->date($position-
>{'open_date'}); my $period_in_the_market = $self->{'args'}->get_arg_values($calc,
$i, 1);

    return if (! $self->check_dependencies($calc, $i));

    return if (! $self->{'args'}->get_arg_values($calc, $i, 2));

    if (($i + 1) eq ($initial_period + $period_in_the_market)) {
        my $order = $pf_manager->sell_market_price($calc, $sys_manager->get_name);
        $pf_manager->submit_order_in_position($position, $order, $i, $calc);
    }
```

```
    return;
}
```

sub manage_short_position { my ($self, $calc, $i, $position, $pf_manager, $sys_manager) = @_; my $initial_period = $calc->prices->date($position->{'open_date'}); my $period_in_the_market = $self->{'args'}->get_arg_values($calc, $i, 1);

```
    return if (! $self->check_dependencies($calc, $i));

    return if (! $self->{'args'}->get_arg_values($calc, $i, 3));

    if (($i + 1) eq ($initial_period + $period_in_the_market)) {
        my $order = $pf_manager->buy_market_price($calc, $sys_manager->get_name);
        $pf_manager->submit_order_in_position($position, $order, $i, $calc);
    }

    return;
}
```

77 GT::CloseStrategy::NeverClose

This strategy never close the already opened positions. This is very usefull to design a sort of Multiple Buy & Hold or Multiple Sell & Hold strategies.

78 GT::CloseStrategy::OppositeSignal

This strategy closes the position once the opposite signal has been emitted by the system. It will will close a long position on a sell signal and close a short position on a buy signal.

79 GT::CloseStrategy::PartialGain

This strategy partialy closes the position once the prices have crossed a limit called stop. This stop is defined as a percentage from the initial price. By default, it's defined as + 10 %. The ratio of the position is parameterized. By default it's half the initial position (0.5).

80 GT::CloseStrategy::PartialStop

This strategy partialy closes the position once the prices have crossed a limit called stop. This stop is defined as a percentage from the initial price. By default, it's defined as - 5 %. The ratio of the position is parameterized. By default it's half the initial position (0.5).

81 GT::CloseStrategy::Reinvest::InWinners

This Position Manager will reinvest money in winning trades every time they meet a new target. This strategy is based on the famous "Let your profits run and cut your losses" while thinking about trend following systems where it is very profitable to bet more when we catched a "big one" !

82 GT::CloseStrategy::Reinvest::ShortGain

In a long position the gains are "automatically" reinvested since the initial sum and the gains are on the market. With a short position this is no more true.

This CloseStrategy tries to defeat this by reinvesting the gains each time a certain amount of gain has been made since last time the position was augmented.

Use this CloseStrategy at the end of the "system chain" so that a position is not augmented if it's planned to be closed.

83 GT::CloseStrategy::Stop::BasedOnIndicators

Overview

This strategy end up a position once prices have crossed the trailing stop determined by indicators.

84 GT::CloseStrategy::Stop::Breakeven

Overview

This strategy place a stop order when we reach a profit target, to be sure that if things are going wrong we will never let a winning trade become a losing one ! The stop should be calculated by including commission & slippage.

85 GT::CloseStrategy::Stop::ExtremePrices

This strategy closes the position once the prices have crossed up the highest high in a short position and crossed down the highest low in a logn position.

86 GT::CloseStrategy::Stop::Fixed

This strategy closes the position once the prices have crossed a limit called stop. This stop is defined as a percentage from the initial price. The limit is parameterized. By default it's 4%.

87 GT::CloseStrategy::Stop::KeepRunUp

Overview

This strategy closes the position once the prices have crossed the trailing stop defined as a percentage below the highest high value for a long trade or above the highest low value for a short trade, called the "run up", since the trade is open. The purpose of this strategy is to keep opening profits and avoid to turn profitable trades into loosing ones.

88 GT::CloseStrategy::Stop::SAR

Overview

This strategy end up a position once prices have crossed the trailing stop determined by the Parabolic SAR (Stop And Reversal).

Note

Keep in mind that some source say "the SAR value is today's, not tomorrow's stop level" and other don't ! :)

Using the Parabolic SAR can be very helpful as long as the security is not prone to short term price trend reversals. If price is erratic, reversing quickly in the short trend, the Parabolic SAR will likely produce poor results.

Links

http://www.stockcharts.com/education/Resources/Glossary/parabolicSAR.html
http://www.equis.com/free/taaz/parabolicsar.html http://www.linnsoft.com/tour/techind/sar.htm

89 GT::CloseStrategy::Stop::VAR

Overview

This method uses market volatility and the concept of value at risk (VAR) to help determine meaningful stop-loss prices and position limits for trading securities.

References

"Value At Risk And Technical Analysis" by Luis Ballesca-Loyo Technical Analysis of Stocks and Commodities - August 1999

90 CloseStrategy of Trend Following System (TFS)

91 GT::Conf

Manage configuration

DESCRIPTION

This module provides functions to manage personal GeniusTrader configuration. The configuration information are stored in file ~/.gt/options by default.

The configuration file format is similar to a perl hash, in other words, a key followed by data for that key. keys are delimited from their value by whitespace. key values can contain embedded whitespace.

key value strings can be continued across multiple lines by delimiting the newline with a backslash (\) (watch out for trailing whitespace after the \ and before the newline).

comments introduced with a # as the first character on a line. data lines cannot contain a comment since the # character is used in many data strings.

blank lines and lines with only whitespace are ignored.

EXAMPLES of ~/.gt/options Entries

```
# this is an example of a comment

DB::module genericdbi

DB::bean::dbname beancounter

Graphic::Candle::UpBorderColor "[0,180,80]"

Graphic::Candle::DownBorderColor "[180,0,80]"
```

this example shows how continuing key values across lines can be useful.

```
DB::genericdbi::prices_sql SELECT day_open, day_high, day_low, \
  day_close, volume, date FROM stockprices WHERE symbol = '$code' ORDER \
  BY date DESC
```

comments are permitted on data lines provided they can be distinguished from positional arguments markers (e.g. #1, #2, etc). in order to do this any trailing data line comment marker (#) must be surrounded by whitespace. the code is a bit more forgiving, using this regex (\s+#[\s\D]+.$)

note that the comment must follow the end of the logical data line and terminates at the end of the logical line. logical line means the line after continuation processing has completed.

examples:

```
 Aliases::Global::TFS2[]       SY:TFS #1 #2 | CS:SY:TFS #1 # comment
 graphic::positions::buycolor  "[0,135,0]" # very dark green
 graphic::buysellarrows::buycolor      "[0,135,0,64]" # semitransparent dark green

note: configuration keys are lower cased automatically regardless of how
      they are defined, but their values are as specified when defined
```

FUNCTIONS

`GT::Conf::load([ $file ])`

Load the configuration from the indicated file. If the file is omitted then it looks at ~/.gt/options by default.

`GT::Conf::clear()`

Clear all the configuration.

`GT::Conf::store($file)`

Write all the current configuration in the given file. Note: all prior commentary, if any is lost.

`GT::Conf::get($key,$defaultValue)`

Return the configuration value for the given key. If the key doesn't exist, it returns the optional defaultValue.

If neither the key nor defaultValue exist, it returns undef.

`GT::Conf::set($key, $value)`

Set the given configuration item to the corresponding value. Replaces any previous value.

`GT::Conf::default($key, $value)`

Set a default value to the given item. Must be called by GT itself to give reasonable default values to most of configurations items.

`GT::Conf::get_first($key, ...)`

Return the value of the first item that does have a non-zero value.

`GT::Conf::=_get_home_path()`

Helper function, returns the home directory environment variable HOME on Unix or on windows the environment variables HOMEDRIVE . HOMEPATH

`GT::Conf::conf_dump( [ "regex" ] )`

Helper function, writes the entire configure key=value pairs on stderr. code example: GT::Conf::conf_dump;

pass a perl regex string to filter the output

`my $gt_root_dir = GT::Conf::get_gt_root()`

Helper function, returns the gt root directory which is the directory that contains GT and Scripts directories, along with any others that may be there. if that configuration key-value is unset check for the environment variable GT_ROOT otherwise returns an empty string

92 GT::DB

Database to retrieve (an history of) prices of various shares

DESCRIPTION

No documentation available. Look at the DB::* modules for real exemples.

`get_name ($code)`

Returns the long name of the market (if defined).

See also ~/.gt/sharenames which contains lines of the form <code>\t<long name> mapping a market code to its long name.

93 GT::DB::CSV

Access to a text files by DBI::CSV

DESCRIPTION

This module handels the access to textfiles by using the DBI:File-module.

Configuration

You can put some configuration items in ~/.gt/options to indicate where the database is.

DB::csv::database : the type of the database ("CSV" by default)

DB::csv::dbname : the name of the database ("cours" by default)

DB::csv::dbhost : the host of the database ("" = localhost by default)

DB::csv::dbuser : the user account on the database

DB::csv::dbpasswd : the password of the user account

Functions

`GT::DB::csv->new()`

Creates a new database-object

`$db->disconnect`

Disconnects from the database.

`$db->init_table($code)`

Creates the table of stock $code.

`$db->init_add_info()`

Creates the addinfo-table.

`$db->init_add_info()`

Creates the shares-table.

`$db->get_prices($code)`

Returns a GT::Prices object containing all known prices for the symbol $code.

`$db->get_last_prices($code, $limit)`

Returns a GT::Prices object containing the $limit last known prices for the symbol $code.

`$db->insert($code)`

Creates the table of stock $code.

`$db->get( parameters )`

Get the datasets where all the parameters match

`$db->available( $code, $date )`

Returns 1 if a dataset for the corresponding day is available.

`$db->get( parameters )`

Delete the datasets where all the parameters match

`$db->edit( parameters )`

Edit the dataset where the date and the code matches

`$db->table_exists($code)`

Test if a table for stock $code already exists

`$db->get_db_name($code)`

Returns the name of the stock designated by $code.

`$db->get_db_code($name)`

Returns the code of the stock designated by $name.

`$db->get_add_info($code,$date)`

Returns an additional information about the stock

`$db->get_add_info($code,$date)`

Returns an additional information about the stock

`$db->set_add_info($value, $info, $code, $date)`

Set an additional information about the stock

`$db->update_from_source($code)`

This function is getting the actual information from the web.

`$db->get_all_prices($code)`

Dummy function. Need to define a clear interface for the exchange.

`$db->merge_from_source($source, $code)`

Merges the content of an other database/source into the current db. This needs to be updated with a "ranking" algorithm.

`$db->merge_all_from_source($source)`

Merges the content of all shares in an other database/source into the current db.

`$db->update_all_from_source($source)`

Updates all shares from a source.

94 DB::HTTP

Retrieve prices from a CGI

DESCRIPTION

Overview

This access module enable you to download prices from a remote server using a CGI script (cf web/quotes.pl).

Configuration

Most configuration items have default values, to alter these defaults you must indicate the configuration item and its value in your $HOME/.gt/options file, especially for authentification purpose.

DB::module HTTP

Informs gt you are using the HTTP.pm module. This configuration item is always required in your $HOME/.gt/options file.

DB::HTTP::url : The URL that will be requested to download

DB::HTTP::location : The location of the server (www.geniustrader.org)

DB::HTTP::zone : The server zone (ie: admin)

DB::HTTP::username : The user name (ie : guest)

DB::HTTP::password : The password (ie : anonymous)

DB::HTTP::marker string

Delimits fields in each row of the data file. The marker defaults to the tab character '\t'.

DB::HTTP::header_lines number

The number of header lines in your data file that are to be skipped during processing. Lines with the either the comment symbol '#' or the less than symbol '<' as the first character do not need to be included in this value.. The header_lines default value is 0.

DB::HTTP::format 0|1|2|3 (default is 3) The format of the date/time string. Valid values are: 0 - yyyy-mm-dd hh:nn:ss (the time string is optional) 1 - US Format (month before day, any format understood by Date::Calc) 2 - European Format (day before month, any format understood by Date::Calc) 3 - Any format understood by Date::Manip

DB::HTTP::fields::datetime number

Column index where to find the period datetime field. Indexes are 0 based. For the particular case of datetime, can contain multiple indexes, useful when date and time are separate columns in the data file. The date time format is anything that can be understood by Date::Manip. A typical example would be YYYY-MM-DD HH:NN:SS. The default datetime index is 5.

DB::HTTP::fields::open number

Column index where to find the period open field. Indexes are 0 based. The default open index is 0.

DB::HTTP::fields::low number

Column index where to find the period low field. Indexes are 0 based. The default low index is 2.

DB::HTTP::fields::high number

Column index where to find the period high field. Indexes are 0 based. The default high index is 1.

DB::HTTP::fields::close number

Column index where to find the period close field. Indexes are 0 based. The default close index is 3.

DB::HTTP::fields::volume number

Column index where to find the period volume field. Indexes are 0 based. The default volume index is 4.

You can set the DB::HTTP::directory configuration item to tell where the quotes are cached.

Functions

`new()`

Create a new DB object used to retry quotes from a CGI on a remote server.

`$db->disconnect`

Disconnects from the database.

`$db->set_directory("/new/directory")`

Indicate the directory containing all the cached data.

`$db->get_prices($code, $timeframe)`

Returns a GT::Prices object containing all known prices for the symbol $code.

`$db->get_last_prices($code, $limit, $timeframe)`

NOT SUPPORTED for HTTP db.

Returns a GT::Prices object containing the $limit last known prices for the symbol $code.

95 DB::MetaStock access module

Overview

The MetaStock access module is able to retrieve quotes from almost any type of MetaStock/Computrac database.

Note

This module calls the binary program identified by the $HOME/.gt/options file option "DB::metastock::program" ("/bourse/tools/MetaStockReader" by default) to get quotes from your metastock database.

$HOME/.gt/options file option "DB::metastock::directory" must indicate the directory of the metastock database. (no default).

Please refer to /bourse/tools/MetaStockReader source if you want to learn more about it.

NOTE: this module and the companion binary program has been depreciated in favor of the stand-alone perl module (unfortunately) also named MetaStock-Reader. refer to that modules pod

Configuration

You can indicate the directory which contains the MetaStock database by setting the DB::metastock::directory configuration item. You can also set DB::metastock::program to indicate where the MetaStockReader binary program is located (complete pathname).

new()

Create a new DB object used to retry quotes from a MetaStock database.

$db->disconnect

Disconnects from the database.

$db->set_directory("/new/directory")

Indicate the directory containing all the text files.

$db->get_prices($code, $timeframe)

Returns a GT::Prices object containing all known prices for the symbol $code.

$db->get_last_prices($code, $limit, $timeframe)

NOT SUPPORTED for text db.

Returns a GT::Prices object containing the $limit last known prices for the symbol $code.

96 DB::MetaStockReader access module

Overview

The MetaStockReader access module is able to retrieve quotes from almost any type of MetaStock database.

This module does not require any other code to support its operation and it is not intended to be used with GT::DB::MetaStock and, although named MetaStockReader, it is not the companion program required by GT::DB::MetaStock.

Synopsis

```
   my $db = create_standard_object("DB::" . GT::Conf::get("DB::module"));
   $db->initialize;
   $db->get_prices("FR0000130007");
   $db->disconnect;
or
   my $db = create_standard_object("DB::" . GT::Conf::get("DB::module"));
   $db->get_prices("FR0000130007");
   $db->disconnect;

   $db->initialize is used to initialize the isin code list.
   Function "get_prices" first test if the isin code is initiasize, if not
it call the function "initialize".
```

Note

This module read the MASTER and the XMASTER file of you security directory to get quotes, with a directory and a symbol as main parameters. The MASTER file contain only the 255 first file (*.DAT) security of your directory. The XMASTER file all the others security (*.MWD) of your directory.

Configuration

NOTE: this module supercedes the module GT::DB::MetaStock. do not attempt to use both.

You can indicate the directory which contains the MetaStock database by setting the DB::metastock::directory configuration item.

new()

Create a new DB object used to retry quotes from a MetaStock database.

$db->disconnect

Disconnects from the database.

$db->initialize

Construct the list of isin code.

$db->set_directory("/new/directory")

Indicate the directory containing your equity.

$db->read_master

Read the MASTER file of your directory containing your equity.

$db->read_xmaster

Read the XMASTER file of your directory containing your equity.

$db->find_isin($code)

Return the description for the symbol $code.

$db->get_db_name($code)

Return the name for the symbol $code.

$db->get_prices($code, $timeframe)

Returns a GT::Prices object containing all known prices for the symbol $code.

$db->puissance($value00,$value01,$value02,$value03)

Convert a MSBIN format to a float perl format (4 bytes). It convert first to a IEEE float format (4 bytes).

$db->get_last_prices($code, $limit, $timeframe)

NOT YET SUPPORTED for MetaStockReader module.

Returns a GT::Prices object containing the $limit last known prices for the symbol $code.

97 COPYRIGHT

98 DB::Text access module

Overview

This database access module enable you to work with a full directory of text files.

Configuration

Most configuration items have default values, to alter these defaults you must indicate the configuration item and its value in your $HOME/.gt/options file.

DB::module Text

Informs gt you are using the Text.pm module. This configuration item is always required in your $HOME/.gt/options file.

DB::text::directory path

where files are stored. This configuration item is always required in your $HOME/.gt/options file.

DB::text::marker string

Delimits fields in each row of the data file. The marker defaults to the tab character '\t'.

DB::text::header_lines number

The number of header lines in your data file that are to be skipped during processing. Lines with the either the comment symbol '#' or the less than symbol '<' as the first character do not need to be included in this value.. The header_lines default value is 0.

DB::text::file_extension string

To be appended to the code file name when searching the data file. For instance, if the data file is called EURUSD.csv this variable would have the value '.csv' (without the quotes).

The default file_extension is '.txt'.

if you have data in different timeframes, for instance, EURUSD_hour.csv and EURUSD_day.csv, use the following value for this directive:

DB::text::file_extension _$timeframe.csv

DB::text::format 0|1|2|3 (default is 3) The format of the date/time string. Valid values are: 0 - yyyy-mm-dd hh:nn:ss (the time string is optional) 1 - US Format (month before day, any format understood by Date::Calc) 2 - European Format (day before month, any format understood by Date::Calc) 3 - Any format understood by Date::Manip

DB::text::fields::datetime number

Column index where to find the period datetime field. Indexes are 0 based. For the particular case of datetime, can contain multiple indexes, useful when date and time are separate columns in the data file. The date time format is anything that can be understood by Date::Manip. A typical example would be YYYY-MM-DD HH:NN:SS. The default datetime index is 5.

DB::text::fields::open number

Column index where to find the period open field. Indexes are 0 based. The default open index is 0.

DB::text::fields::low number

Column index where to find the period low field. Indexes are 0 based. The default low index is 2.

DB::text::fields::high number

Column index where to find the period high field. Indexes are 0 based. The default high index is 1.

DB::text::fields::close number

Column index where to find the period close field. Indexes are 0 based. The default close index is 3.

DB::text::fields::volume number

Column index where to find the period volume field. Indexes are 0 based. The default volume index is 4.

new()

Create a new DB object used to retrieve quotes from a directory full of text files containing prices.

$db->disconnect

Disconnects from the database.

$db->set_directory("/new/directory")

Indicate the directory containing all the text files.

$db->get_prices($code, $timeframe)

Returns a GT::Prices object containing all known prices for the symbol $code.

$db->get_last_prices($code, $limit, $timeframe)

Returns a GT::Prices object containing the $limit last known prices for the symbol $code.

99 GT::DB::bean

Access to beancounter database of quotes

DESCRIPTION

This module is used to retrieve quotes from a MySQL/PostgreSQL database as setup by beancounter. By default, the database is supposed to be running on localhost and the only authentication done is the standard Unix one.

Configuration

You can put some configuration items in ~/.gt/options to indicate where the database is.

DB::bean::dbname : the name of the database ("beancounter" by default)

DB::bean::dbhost : the host of the database ("" = localhost by default)

DB::bean::dbport : the port where the server is running ("" = default port number)

DB::bean::dbuser : the user account on the database (current user by default)

DB::bean::dbpasswd : the password of the user account

DB::bean::db : the database being used (mysql|Pg) ("mysql" by default)

Functions

`GT::DB::mysql->new()`

`$db->disconnect`

Disconnects from the database.

`$db->get_prices($code, $timeframe)`

Returns a GT::Prices object containing all known prices for the symbol $code.

`$db->get_last_prices($code, $limit, $timeframe)`

Returns a GT::Prices object containing the $limit last known prices for the symbol $code.

Notice that beancounter only supports daily data, therefore it will throw an error if you try to retrieve data in timeframes smaller than daily.

`$db->get_db_name($code)`

Returns the name of the stock designated by $code.

100 GT::DB::genericdbi

Access to any database of quotes, as long as a dbi driver is available

DESCRIPTION

This module is used to retrieve quotes from your existing database

Configuration

You can put some configuration items in ~/.gt/options to indicate where the database is.

DB::genericdbi::dbname : the name of the database

DB::genericdbi::dbhost : the host of the database

DB::genericdbi::dbport : the port where the server is running

DB::genericdbi::dbuser : the user account on the database

DB::genericdbi::dbpasswd : the password of the user account

DB::genericdbi::db : the database being used (mysql|Pg|...) ("mysql" by default)

DB::genericdbi::prices_sql : The query used to retrieve price data.

Make sure to retrieve the data in the following order: open, high, low, close, volume, date/time

Also, make sure to retrieve the data ordered by date/time descending Example:

```
SELECT period_open, period_high, period_low, period_close, volume, Conca
```

DB::genericdbi::name_sql : The query used to retrieve a symbol's description.

Example:

```
SELECT name FROM stockinfo WHERE symbol = '$code';
```

Functions

`GT::DB::genericdbi->new()`

`$db->disconnect`

Disconnects from the database.

`$db->get_prices($code, $timeframe)`

Returns a GT::Prices object containing all known prices for the symbol $code.

`$db->get_last_prices($code, $limit, $timeframe)`

Returns a GT::Prices object containing the $limit last known prices for the symbol $code in the given $timeframe.

`$db->get_db_name($code)`

Returns the name of the stock designated by $code.

101 GT::DB::pg

Access to PostgreSQL database of quotes

DESCRIPTION

This module is used to retry quotes from a Postgresql database. By default, the database is supposed to be running on localhost and the only authentication done is the standard Unix one.

Configuration

You can put some configuration items in ~/.gt/options to indicate where the database is.

DB::pg::dbname : the name of the database ("cours" by default)

DB::pg::dbhost : the host of the database ("" = localhost by default)

DB::pg::dbuser : the user account on the database

DB::pg::dbpasswd : the password of the user account

Functions

`GT::DB::pg->new()`

`$db->disconnect`

Disconnects from the database.

`$db->get_prices($code, $timeframe)`

Returns a GT::Prices object containing all known prices for the symbol $code.

`$db->get_last_prices($code, $limit, $timeframe)`

Returns a GT::Prices object containing the $limit last known prices for the symbol $code.

`$db->get_db_name($code)`

Returns the name of the stock designated by $code.

102 GT::DateTime

Manage TimeFrames and provides date/time helper functions

DESCRIPTION

This module exports all the variable describing the available "periods" commonly used for trading : $PERIOD_TICK $PERIOD_1MIN, $PERIOD_5MIN, $PERIOD_10MIN, $PERIOD_15MIN, $PERIOD_30MIN, $HOUR, $PERIOD_2HOUR, $PERIOD_3HOUR, $PERIOD_4HOUR, $DAY, $WEEK, $MONTH, $YEAR.

The timeframes are represented by those variables which are only numbers. You can compare those numbers to know which timeframe is smaller or which one is bigger.

It also provides several functions to manipulate dates and periods. Those functions use modules GT::DateTime::* to do the actual work depending on the selected timeframe.

Functions provided by submodules

map_date_to_time($date) is a function returning a time (ie a number of seconds since 1970) representing that date in the history. It is usually corresponding to the first second of the given period.

map_time_to_date($time) is the complementary function. It will return a date describing the period that includes the given time.

Functions

`GT::DateTime::map_date_to_time($timeframe, $date)`

`GT::DateTime::map_time_to_date($timeframe, $time)`

Those are the generic functions used to convert a date into a time and vice versa.

`GT::DateTime::convert_date($date, $orig_timeframe, $dest_timeframe)`

This function does convert the given date from the $orig_timeframe in a date of the $dest_timeframe. Take care that the destination timeframe must be bigger than the original timeframe.

`GT::DateTime::list_of_timeframe()`

Returns the list of timeframes that are managed by the DateTime framework.

`GT::DateTime::name_of_timeframe($tf)`

Return the official name of the corresponding timeframe.

`GT::DateTime::name_to_timeframe($name)`

Returns the timeframe associated to the given name.

`GT::DateTime::timeframe_ratio($first, $second)`

Returns how many times the second timeframe fits in the first one.

103 GT::DateTime::10Min

This module treat dates describing a 10 minute period. They have the following format : YYYY-MM-DD HH:N0:00

104 GT::DateTime::15Min

This module treat dates describing a quarter-hour. They have the following format : YYYY-MM-DD HH:NN:00

105 GT::DateTime::1Min

This module treat dates describing a 1 minute period. They have the following format : YYYY-MM-DD HH:NN:00

106 GT::DateTime::2Hour

This module treat dates describing the 2Hour timeframe. They have the following format : YYYY-MM-DD HH:00:00

107 GT::DateTime::30Min

This module treat dates describing an half-hour. They have the following format : YYYY-MM-DD HH:N0:00

108 GT::DateTime::3Hour

This module treat dates describing the 3Hour timeframe. They have the following format : YYYY-MM-DD HH:00:00

109 GT::DateTime::4Hour

This module treat dates describing the 2Hour timeframe. They have the following format : YYYY-MM-DD HH:00:00

110 GT::DateTime::5Min

This module treat dates describing a 5 minute period. They have the following format : YYYY-MM-DD HH:NN:00

111 GT::DateTime::Day

This module treat dates describing a day. They have the following format : YYYY-MM-DD

112 GT::DateTime::Hour

This module treat dates describing an Hour. They have the following format : YYYY-MM-DD HH:00:00

113 GT::DateTime::Month

This module treat dates describing a month. They have the following format : YYYY-MM

114 GT::DateTime::Tick

This module treat dates describing ticks. They have the following format : YYYY-MM-DD HH:NN:SS

115 GT::DateTime::Week

This module treat dates describing a week. They have the following format : YYYY-WW

WW being the week number.

116 GT::DateTime::Year

This module treat dates describing a year. They have the following format : YYYY

117 GT::Dependency

A dependency system for indicators/signals/systems.

DESCRIPTION

This module is inheritated by any object which needs a Dependency mechanism. That's why it's listed in @ISA of GT::Indicators, GT::Signals, GT:Systems and several other modules ...

`$object->add_indicator_dependency($indic, $nbdays)`

`$object->add_signal_dependency($signal, $nbdays)`

`$object->add_prices_dependency($nbdays)`

`$object->add_arg_dependency($argnum, $nbdays)`

`$object->add_volatile_indicator_dependency($indic, $nbdays)`

`$object->add_volatile_signal_dependency($signal, $nbdays)`

`$object->add_volatile_prices_dependency($nbdays)`

`$object->add_volatile_arg_dependency($argnum, $nbdays)`

Add a dependency on a precise indicator or signal. The current object needs nbdays of history (current day included) on the given indicator or signal to be able to produce a result. A prices dependency indicates the number of days of history prices

A volatile dependency will be removed by $object->remove_volatile_dependencies(). It is used for adding last-minute dependencies whose values are known late because they are computed based on other data.

`$object->get_prices_dependency()`

`$object->get_signal_dependencies()`

`$object->get_indicator_dependencies()`

Return the dependency or list of dependencies.

`$object->remove_volatile_dependencies()`

Removes all volatile dependencies.

`$object->days_required`

Returns the number of days required so that the object can produce a result.

`($first, $last) = $object->update_interval($calc, $first, $last)`

Check the limits of the interval. Return new limits. The interval is contained in the first interval but all days will produce a result. The new interval may be equal to the given interval.

`$object->check_dependencies($calc, $i)`

`$object->check_dependencies_interval($calc, $first, $last)`

Check that there is enough data available. If there isn't return false. Otherwise make sure the required data are computed and return true.

`$object->dependencies_are_available($calc, $i)`

`$object->dependencies_are_available_interval($calc, $first, $last)`

Check if all dependencies have been computed.

`$object->compute_dependencies($calc, $i)`

`$object->compute_dependencies_interval($calc, $first, $last)`

Calculate all dependent indicators and detect all dependent signals.

118 GT::Indicators::Example

The example indicator

DESCRIPTION

The Example indicator trys to express X, Y, Z.

Calculation

The Example indicator is calculated by ...

Parameters

The number of days of the average ...

119 miscellaneous file format information

this document may contain slightly misleading information, due to changes in the gt code base, the authors operational version, which served as the basis for this data or the errors might just be errors, either explicit or error by omission.

document improvements are welcome. please post to the gt development list.

files used by genius trader (gt)

```
$HOME/.gt -- this is your private gt configuration directory

data kept here will be used only when the gt process is running
with your user id.

/usr/share/geniustrader is a shared geniustrader directory. files
kept there can be shared by anyone. gt will search for specific files
in this directory for alias files (see below) and rgb (color) data.
this path is the default path, it can be changed using an envvar
and gt configuration key-values. the envvar is "GT_ROOT". the default
root can also be set using the gt configuration key "GT::ROOT".
config keys for Path::Aliases (8 of them) and SearchPath::X11_rgb for
the color data file.

$HOME/.gt/options -- this is your primary gt configuration file

scan.pl and backtest_many.pl
  market_file -- list of security codes
  system_file -- system-signals to scan

aliases files
  global_alias -- from dir $HOME/.gt/
  object_alias -- from Path::Aliases keys for each specific type

colors -- x11 rgb file from SearchPath::X11_rgb search path
  used if a colour name is used but not found in the hardcoded list
  no harm if file not found (named colour will fail to "black".

xml files (GT/MetaInfo.pm)
  your gt portfolio
    -- named on command line manage_portfolio.pl and other
       Script apps (backtest*.pl etc)
    -- named in graphic config PortfolioPositions directives
```

```
security (price data) databases (flat files)
  GT/Prices.pm -- prices files
  GT/DB.pm -- $HOME/.gt/sharenames (used when the prices database
              scheme doesn't include company name)
  GT/DB/MetaStockReader.pm
  GT/DB/MetaStock.pm (depreciated in favor of MetaStockReader.pm)
  GT/DB/HTTP.pm
fixme

chart image files -- most likely written to stdout by default
  GT/Graphics/Driver/GD.pm
  GT/Graphics/Driver/SVG.pm
  GT/Graphics/Driver/ImageMagick.pm
  GT/Graphics/Driver/Postscript.pm
fixme

analysis report templates --
  fix me

prior analyses (GT/Analyzers/Process.pm)
  fixme
```

120 market_file – this is just a general text file

a list of market codes or security symbols one code per line. the code can appear anywhere on the line. a '#' indicates the beginning of a comment; characters up to the end of the line are ignored. a comment may also appear on a line containing a code.

blank lines, and lines containing only whitespace are ignored.

codes are character strings from the this set of chars [a-zA-X0-9_^.]

scan.pl runs its analysis on each code in turn

121 system_file | signal_file – this is just a general text file

a list of one or more gt signal or system descriptions or specifications (sig-sys-desc), one per logical line. long sig-sys-desc can be broken into shorter more readable lines by placing the line continuation character '\' just prior to the line terminator character (watch out for trailing whitespace and be

sure there is a blank line or normally terminated line as the last line of a logical line.

a '#' indicates the beginning of a comment; characters up to the end of the line are ignored. comments can appear on sig-sys-desc that are on one line, but comments cannot be embedded into logical line groupings. you can effectively comment out an entire logical line by putting a comment on first line of the logical line group.

blank lines, and lines containing only whitespace are ignored.

scan.pl runs an analysis based on each sig-sys-desc found in the file.

sig-sys-desc examples:

```
SY:SMA 2 3 4 | TF:AcceptAll

S:Generic:And \
 { S:Generic:Below { I:Prices CLOSE } { I:G:PeriodAgo } } \
 { S:Generic:Below { I:Prices CLOSE } { I:SMA 50 {I:Prices CLOSE} } } \
 NAME below 50 day sell \
```

the following logical line group is effectively commented out

```
#SY:SMA 10 20 200 \
| TF:AcceptAll

 System:ADX 30 | TradeFilters:LongOnly | MoneyManagement:FixedSum
 same as above using abbreviations
 SY:ADX 30 | TF:LongOnly | MM:FixedSum
```

122 global_alias – $HOME/.gt/options file

a <system_alias> is a short hand for one or more system components. system components is another reference for sig-sys-desc. a <system_alias> name should be restricted to the characters in this character set [a-zA-Z0-9_] (alphanumeric and underscore). using any of these characters is bound to be troublesome: ':@#[]'.

the format of a system_alias entry:

```
  "Aliases::Global::<system_alias>" <whitespace> <sig-sys-desc>
```

where "Aliases::Global::<system_alias>" is case insensitive, and will always be processed in lower case. <sig-sys-desc> on the other hand is case sensative. see sig-sys-desc for details on how to craft these elements.

123 object_alias – system component (objects) aliases

object aliases are aliases of individual system components but not complete or partial systems (e.g. they cannot contain a "|" symbol.

these aliases are stored in top level dir /usr/share/geniustrader/aliases by default, but are also searched for in $HOME/.gt/aliases/.

object aliases are grouped by the kind of system component ("signals", "indicators", "systems", "closestrategy", "moneymanagement", "tradefilters", "orderfactory", "analyzers") <kind> they represent and are searched for in directories bearing that name. this allows a common name to be used in a system context.

the format of a object_alias entry:

```
"Path::Aliases::<kind>::<object_alias>" <whitespace> <sig-sys-desc>
```

or in your $HOME/.gt/options file:

```
"Aliases::<kind>::<object_alias>" <whitespace> <sig-sys-desc>
```

where "(Path::)*Aliases::<kind>::<object_alias>" is case insensitive – is it?

can <kind> be abbreviated using the standards?

object_alias examples:

```
Aliases::Indicators::MyMean  { I:Generic:Eval ( #1 + #2 ) / 2 }
```

other than that they are very much like a system alias, but in use they are referenced in <sig-sys-desc> by prefixing the '@' symbol to the <object_alias> identifier. for example { @I:MyMean 50 {I:RSI} } denotes that the indicator I:MyMean is an object alias.

124 $HOME/.gt/options

file containing perl hash key-values that set the value of the given key. for the most part every key will (should) have a hardcoded default value, use this file to override the default should you want a different value.

gt isn't (but probably should be) shipped with a default options file. there are a couple required items that must be set there as they do not have hardcoded values:

```
securities prices sources
truetype font paths
```

securities prices sources setup:

to use the sample_data files add these lines – watch for trailing whitespace

```
    DB::module Text
    DB::text::directory /path/to/sample_data/directory
    # no trailing / on directory

  to use a database engine
    refer to your db engine type interface in GT/DB/
    fix me
```

paths to your computers truetype fonts – be sure to edit the path as well as the actual file name for your computer absolute minimum is Path::Font::Arial key and a value

```
    Path::Font::Arial /path/to/truetype/font/arial.ttf
    Path::Font::Courier /path/to/truetype/font/couri.ttf
    Path::Font::Times /path/to/truetype/font/times.ttf
    Path::Font::Fixed /path/to/truetype/font/VeraMono.ttf
```

a broker module (not required, but recommended) Brokers::module Self-Trade

background color override (default is harsh white) Graphic::BackgroundColor "LINEN"

there isn't but maybe should be a key for the graphics driver (GD (recommended), SVG, PostScript, ImageMagick). this is set in the gt Script app (Scripts/backtest.pl, Scripts/graphic.pl)

125 xml files (GT/MetaInfo.pm)

fix me

126 $HOME/.gt/sharenames

lines mapping a market code to its long name. each line has this format:

```
  'the securities code' a literal tab character 'the securities long name'
```

do not quote the code. if the long name is quoted the quotes will be part of the name string.

127 security (price data) databases

```
GT/Indicators/EVWMA.pm /bourse/metainfo/       # EVWMA.pm appears broken
GT/List.pm             /bourse/listes/
```

fix me

128 analysis report templates

```
Scripts/manage_portfolio.pl    Scripts/Templates/
Scripts/backtest.pl            Scripts/Templates/
Scripts/backtest_multi.pl      /bourse/perl
```

fix me

129 prior analyses (GT/Analyzers/Process.pm)

fix me

130 chart image files

fix me

131 DESCRIPTION

fix me

132 BUGS

fix me

133 SEE ALSO

scan.pl, backtest.pl, backtes_many.pl, gt_sig-sys-desc.pod, Tools.pm, $HOME/.gt/options, $HOME/.gt/aliases/, $HOME/.gt/portfolio/,

134 gt system description specifications (sys-sig-indic-desc)

used for defining indicators, signals, systems, and trading system components. these sys-sig-indic-desc are found throughout gt configuration and control files and on command lines.

sys-sig-indic-desc are text strings used to specify or describe the parameters used by gt to perform a particular technical analysis (ta) on a security.

in general sys-sig-indic-desc

a)

are delimited with curly brackets '{...}'

for example in { I:SMA } the whitespace within the curly brackets is ignored, thus {I:SMA} is equivalent, but harder for a human to scan.

b)

may include other sys-sig-indic-desc

for example { I:Prices CLOSE } is an embedded sys-sig-indic-desc in this { I:SMA 50 { I:Prices CLOSE } } sys-sig-indic-desc.

c)

are written with case sensitive tokens unless otherwise noted

sig-sig-indic names are always case sensitive, but other arguments might be case insensitive. refer to the pod for the sig-sig-indic in question, or to the documentation regarding the argument (for example GT::Tools, GT::Graphics::Tools). in the examples a) and b) above only 'CLOSE' is case insensitive:

```
{ I:SMA 50 { I:Prices close } }
```

d)

use whitespace to delimit individual elements within a sys-sig-indic-desc. for example example b) above can be written like this:

```
{I:SMA 50{I:Prices CLOSE}}
```

note the required whitespace to delimit the time period (50) from the indicator name (I:SMA) and the price value id token (CLOSE) from the indicator name (I:Prices).

e)

may be written using sig-sig-indic name abbreviations (refer to GT::Tools for details)

without sig-sig-indic name abbreviations example a) would be written like this

```
{ Indicators::SMA }
```

note Indicators is plural (and case sensitive), as are all the sig-sig-indic names. but trading system components are a mixed bag, some plural, others singular (see GT/Docs/how_to_spec+debug_a_system.pod), the singular and any misspelling or altered case will fail, usually with a somewhat unhelpful error message. so use the abbreviations.

f)

may include system or object aliases

{ @S:3EMAlong #1 #2 #3 } is an example of a signal object alias (@S:3EMAlong) with 3 arguments (refer to GT/Docs/object_aliases.pod and GT::Tools for details on system and object aliases).

g)

trading system components are delimited with vertical bar '|'.

trading systems include a system and possibly other components that, collect together the named components and listed parameters into a trading system rule set. (see GT/Docs/how_to_spec+debug_a_system.pod) for example

```
SY:ADX | TF:LongOnly | MM:FixedSum
```

ADX system with a longonly trade filter and a fixed sum money manageer. however, missing from this trading system are a broker, which will add (subtract) trading and account costs from trading performance analyses. order filters, which aid in preventing trades that have flaws that the systems signals do not recognize. lastly, and most importantly a closingstrategy, which determines when to close an existing open position. a closingstrategy is the sell counterpart to the systems buy, but note both (systems and a closingstrategy) define two signals. the first signal applies to 'long' positions, the second to 'short' positions.

135 positional value substitution – sys-sig-indic arguments

a sys-sig-indic-desc may have parameters that specify values to be used. within a sys-sig-indic-desc whitespace is used to delimit these arguments from each other and from the sys-sig-indic name.

in addition the alias provisions allow for numbered parameter substitution. for details on system and object aliases see GT/Docs/object_aliases.pod and GT::Tools.

sys-sig-indic, for the most part, have predefined default arguments that will be used in the absence of user provided values.

there isn't an easy way to get the number of and defaults values of arguments supported by a particular sys-sig-indic short of reading the perl

code for the sys-sig-indic (look for @DEFAULT_ARGS). however, the appropriate evaluation script display_*.pl can be used with advantage to see the default args as well as how any user passed args are applied. the standard output from those scripts will show the values used.

when an indicator uses a prices value token (one of OPEN, HIGH, LOW, CLOSE, or VOLUME) the usual default is CLOSE (or VOLUME) as appropriate for the indicator. to alter that indicators prices value token the entire prices sys-sig-indic description must be embedded within the outer sys-sig-indic. in addition, in most cases the prices argument follows other arguments. it isn't possible (at least i've yet to determine how) to use earlier defaults and change a subsequent argument value without entering all the earlier arguments explicitly. take simple moving average { I:SMA } for example:

```
{ I:SMA } defaults to { I:SMA 50 { I:Prices CLOSE } }
```

in order to change the default prices value token from CLOSE to OPEN you must write the indic-desc in the expanded form:

```
{ I:SMA 50 { I:Prices OPEN } }
```

```
yep, that's the way it works.
```

136 indicator description (indic-desc) – define an indicator

an indicator is a mathematical calculation based on a securities price and/or volume, or possibly some other numeric value associated with a security or company, shares outstanding for example.

examples of indic-desc:

```
{ I:Prices LOW }
{ I:Prices VOLUME }
{ I:SMA }
{ I:EMA }

{ I:BOL 40 1.9 }
{ I:BOL 40 1.9 { I:Prices HIGH } }
```

in gt, an indicator is a series of data values for each bar (time period) in the window of time being analyzed. the indicator will have at least one value per bar, but may have many more. by default an indicators principle value is the first value returned. except for display_indicator.pl,

which will output all values, this first value will be the value used when the indicator is listed in a sys-sig-indic-desc without a numeric value indicator. say what!?

ok, indicators that have multiple values can be identified by appending a slash '/' and a number corresponding to the indicator value desired to the indicator name as a means to designate the value of interest. the numbers start with 1 the default value and increase by one for each subsequent value. by convention the code /99 is used to indicate all values are to be returned.

'indicator_name/number' syntax examples:

```
{ I:AROON/3 }
{ I:BOL/3 }
{ I:ADX/2 }
{ I:STO/4 }
{ I:HilbertPeriod/10 }
{ I:Chandelier/1 }    # identical to { I:Chandelier }
```

{ I:Prices } is completely different. it will only return one value at a time, by default it is CLOSE or LAST. furthermore { I:Prices } does not use the indic/number encoding, instead using a 'name' argument. the names arguments are case insensitive:

```
OPEN | FIRST
HIGH
LOW
CLOSE | LAST
VOLUME
DATE
```

incidentally, { I:Prices DATE } is useful if you want the timeframe adjusted date string that corresponds to the time period index.

to reiterate – { I:indic } will return one value per bar, and it is, by default, the first value defined by that indicator. in other words { I:indic/1 }. by convention the first indicator value should be the value logically associated with the indicator name, but there isn't any way to enforce that convention. in many cases gt indicators have multiple values that include oscillator values along with other values that relate to that indicator. to be sure what values are available from a given indicator read the pod, then the code.

the sys-sig-indic devel/evaluation apps display_indicator.pl, display_signal.pl, and display_system.pl are useful in getting valid run-time default values and the other argument for the specified sys-sig-indic desc without having to read perl code.

```
% display_indicator.pl I:AROON T | head
display_indicator.pl: interval: 2457 .. 2657
Indicator I:AROON has 3 values ... all values selected
        I:AROON/1  <=> AroonUp[25, { I:Prices HIGH}, { I:Prices LOW}]
        I:AROON/2  <=> AroonDown[25, { I:Prices HIGH}, { I:Prices LOW}]
        I:AROON/3  <=> AroonOsc[25, { I:Prices HIGH}, { I:Prices LOW}]

        timeframe day, time periods 2457 .. 2657
Calculating all 3 indicators ...
AroonUp[25, { I:Prices HIGH}, { I:Prices LOW}][2008-10-03] = 8.0000
AroonDown[25, { I:Prices HIGH}, { I:Prices LOW}][2008-10-03] = 92.0000
AroonOsc[25, { I:Prices HIGH}, { I:Prices LOW}][2008-10-03] = -84
```

137 signal description (sig-desc) – define a signal

signals are binary values (e.g. true/false, yes/no) that are used to trigger trading system actions (orders). a signal is a data series with a discrete value for each bar in the time period over the time being analyzed.

signal descriptions (sig-desc) are similar to indicator descriptions with arguments (positional value substitution). note a signal may also have multiple signal values. this is unusual, but something to be aware of (refer to S:Swing:Trend, S:Swing:TrendEnding, S:Trend:HilbertChannelBreakout and others).

sig-desc look very much like indic-desc, here are some examples

```
{ Signals::Generic::CrossOverUp {I:BBO} {I:Generic::Eval 1.0} }
{ S:Generic:Above {I:Prices CLOSE} {I:BOL/2 20 2.0} }
{ S:Generic:Increase {I:ADX} }
```

using the available boolean logic signals one can construct some fairly complex signal descriptions.

```
{ S:G:And \
  {S:G:CrossOverUp {I:SMA 3} {I:SMA 23}} \
  {S:G:Increase {I:ADX}} \
}

{S:G:And \
  {S:G:Or \
    {S:G:And \
      {S:G:Above {I:G:PeriodAgo 4  {I:STO/3 3 2 2 2}} 80} \
      {S:G:Above {I:G:PeriodAgo 4} {I:G:PeriodAgo 4 {I:SMA 50}}} \
```

```
      } \
      {S:G:And \
        {S:G:Below {I:G:PeriodAgo 4  {I:STO/3 3 2 2 2}} 20} \
        {S:G:Below {I:G:PeriodAgo 4} {I:G:PeriodAgo 4 {I:SMA 50}}} \
      } \
    } \
    {S:G:Above {I:Prices HIGH} {I:G:PeriodAgo 4 {I:Prices HIGH}}} \
  }
```

use script app display_signal.pl to aid you in the development, debugging and proofing of complex signal descriptions.

138 systems description (sys-desc) – define a systems

a gt sys-desc defines two signals, the first generates an order to (potentially) open a 'long' position, the second generates an order to (potentially) open a 'short' position. these are expressly not buys and sells, but orders to (potentially) enter into a new position either on the 'long' or the 'short' side of a trade. depending on other trading system components (tradefilters, orderfactory and moneymanagers) these orders may not be fulfilled. positions once opened are managed by the PortfolioManager via one or more CloseStrategy trading system component. the second signal, if omitted will default to always false

```
--statement validation required--
```

the first signal must always be present. you can disabled either signal using this sig-desc:

```
{ S:G:False }
```

note that is one of the two required sig-desc within a sys-desc.

gt has a couple predefined (hardcoded) systems:

```
{ SY:TTS }             # initial attempt to imitate turtle trade
{ SY:TFS }             # trend following system
{ SY:Stochastic }      # stochastic system
see GT/Systems/ for the rest
```

the usual arguments and embedded sys-sig-indic-desc arguments apply to sig-desc.

```
{ SY:Stochastic 12 3 4 5 }
{ SY:TFS 20 7 }
```

using relatively complex sys-sig-indic-desc you can formulate much more interesting sig-desc:

```
{ SY:Generic \
  { S:Generic:Above \
    {I:Prices HIGH } {I:Generic:MaxInPeriod #1 {I:Prices CLOSE}} \
  } \
  { S:Generic:Below \
    {I:Prices LOW  } {I:Generic:MinInPeriod #1 {I:Prices CLOSE}} \
  } \
}

{ SY:Generic \
  {S:G:And \
    {S:G:Or \
      {S:G:And \
        {S:G:Above {I:G:PeriodAgo 4  {I:STO/3 3 2 2 2}} 80} \
        {S:G:Above {I:G:PeriodAgo 4} {I:G:PeriodAgo 4 {I:SMA 50}}} \
      } \
      {S:G:And \
        {S:G:Below {I:G:PeriodAgo 4  {I:STO/3 3 2 2 2}} 20} \
        {S:G:Below {I:G:PeriodAgo 4} {I:G:PeriodAgo 4 {I:SMA 50}}} \
      } \
    } \
    {S:G:Above {I:Prices HIGH} {I:G:PeriodAgo 4 {I:Prices HIGH}}}} \
  {S:G:And \
    {S:G:Or \
      {S:G:And \
        {S:G:Above {I:G:PeriodAgo 4  {I:STO/3 3 2 2 2}} 80} \
        {S:G:Above {I:G:PeriodAgo 4} {I:G:PeriodAgo 4 {I:SMA 50}}} \
      } \
      {S:G:And \
        {S:G:Below {I:G:PeriodAgo 4  {I:STO/3 3 2 2 2}} 20} \
        {S:G:Below {I:G:PeriodAgo 4} {I:G:PeriodAgo 4 {I:SMA 50}}} \
      } \
    } \
    {S:G:Below {I:Prices LOW} {I:G:PeriodAgo 4 {I:Prices LOW}}}} \
}
```

139 trading sys description – define a set of trading system components

sig-sys-desc are used to specify a set of properties and parameters to be used by gt to perform the technical analysis desired. it may encompass one or more components of an analysis system. they are composed of the items described below. see sig-sys-desc examples above.

supported abbreviations (not listed elsewhere): Generic = G: Signals = S: Indicators = I:

140 trading system components – parts of a system

```
GT/<directory>          abbreviation
   Systems                  SY:
   OrderFactory             OF:
   MoneyManagement +        MM:
   tradeFilters  +          TF:
   CloseStrategy  +         CS:
   Brokers                 <none>
```

see directories for details on the elements of that component

components of a systems description are separated by vertical bars ("|"). components marked + above allow multiple elements of that component.

yet more sys-desc examples:

```
System:ADX 30 | TradeFilters:LongOnly | MoneyManagement:FixedSum
```

same as above using abbreviations

```
SY:ADX 30 | TF:LongOnly | MM:FixedSum
```

sig-sys-desc examples:

```
SY:SMA 2 3 4 | TF:AcceptAll
```

the following logical line group is effectively commented out

```
#SY:SMA 10 20 200 \
 | TF:AcceptAll
```

141 SEE ALSO display_indicator.pl, display_signal.pl, display_system.pl, GT/Docs/how_to_spec+debug_a_system GT/Docs/object_aliases.pod scan.pl, backtest.pl, gt_file.pod, GT::Tools, GT::Graphics::Tools, and the pod for individual system, signal and indicators

142 how to create and debug a gt system

most of this material is available in various places in the gt code and documentation base. pulling it together here is an attempt to aid first time and more experienced users in the finer points of gt use.

objective – help GT users with trading systems

this file is targeted to both budding as well as seasoned (e.g. graybeards) GT users. the intent is to gather sage wisdom and experience in the use of gt primarily in the area of system description development, evaluation, troubleshooting, proofing and ultimately use.

contributions, corrections, enhancements welcome

scope – how to create and debug a gt system

you are fooling yourself trying to get something out of gt if you don't spend some time with your favorite pod reading tool going hand-over-hand with the gt application scripts as well as some of the informational documentation. then you will need to review the components that can be included as part of a gt trading system.

here is a reading list to start with

```
gt_sig-sys-desc.pod
gt_files.pod
backtest.pl
GT/SystemManager.pm
GT/CloseStrategy.pm
GT/PortfolioManager.pm
GT/TradeFilters.pm
GT/OrderFactory.pm
GT/MoneyManagement.pm
```

if you are intending to write new perl code (or modify existing code) you will need to study Writing_an_Indicator_Cookbook.pdf in GT/Docs, along with some of the existing modules to see how things are done.

document abbreviations

```
fixme         looking for user input
gt            genius trader
-tbs-         to be supplied
(verify)      statement needs to be verified
```

143 table of contents

144 1.0 gt trading systems

gt trading systems are described in terms of trading system components, signals and indicators.

gt trading systems are formed using orders and positions. orders may not result in a position being entered into. once a position is opened (entered into) the portfolio manager takes control of it.

the portfolio manager includes trading system components tradefilter (verify), orderfactory (verify), closestrategy, moneymanager,

trading systems are formed by combining gt trading system component descriptions together separating each with the vertical bar (pipe) (|) symbol

1.1 gt trading system examples

-tbs-

1.2 trading system components

gt defines these trading system components

1. 2.1 Systems: – abbreviation SY
   each gt trading system requires a systems (yes it is plural) that defines two signals that, when triggered, yield an order to open a position. gt defines two types of positions, long and short, and manages each type separately, hence the need for two separate signals. the first signal is for opening a new long position, the second signal is for opening a new short position.
   whether or not an order results in a position being opened depends on a number of other trade system components defined below.
2. 2.2 OrderFactory: – abbreviation OF:
   -tbs-
   multiple orderfactory are supported per trading system (verify)
3. 2.3 TradeFilter: – abbreviation TF:
   restricts the types of trades allowed.
   multiple tradefilters are supported per trading system

4. 2.4 MoneyManager: – abbreviation MM:

   defines how gt manages investments. affects share quantities allocated to a given order

   multiple moneymanagers are supported trading system, each will be evaluated in the sequence defined. the resulting order share quantity will be the quantity that remains after all moneymanagers have been polled.

5. 2.5 CloseStrategy: – abbreviation CS:

   because each type of position (long or short) is managed separately a closestrategy must also specify two signals that trigger management activity of each position type.

   the first signal will generate orders that relate to an existing long position, the second signal will generate orders for an existing short position.

   closestrategy, really a misnomer for position management, includes complete and partial position closing as well as reinvesting (adding to) existing positions.

   multiple closestrategies are supported per trading system. each is evaluated in the sequence defined, the resulting order, if any will be applied to the appropriate existing position.

   ```
   verify this verify this verify this
   ```

   orders that issue from closestrategies are not subject to any of the trading filters (e.g. orderfactory, tradefilter or moneymanager) defined in the trading system but are applied immediately. (i'm thinking that orderfactories and moneymanagers do in fact affect position manager orders)

   ```
   verify this verify this verify this
   ```

6. 2.6 Broker: – abbreviation none

   a broker is used to apply trading costs to each trade

   a broker isn't a full fledged gt trading system component since it cannot be defined as a component in a system description.

   specify a broker using the command line option

   ```
   --broker broker_name argument(s)
   ```

   or by setting the gt configuration key "Brokers::module" to the desired broker_name and any arguments.

145 2.0 gt sig-indic descriptions

-tbs-

2.1 Indicators: – abbreviation I:

refer to pod file GT/Docs/gt_sig-sys-desc.pod

2.2 Signals: – abbreviation S:

refer to pod file GT/Docs/gt_sig-sys-desc.pod

here is something i just (re)discovered, it might (or might not) be in the referenced pod, but it is very very significant; if it is ignored you will probably get hard to interpret results and troubleshooting will be painful at best.

signals, just like indicators can have more than a signal output. it's uncommon, but there are a couple. the significance is when you use one of these multi-name signals as an element in a larger sys-sig-indic description to be sure to specify the signal value selector (e.g. the /#) that you want. by default gt will use the first one unless otherwise specified (verify). except, possibly, when the signal is being used as a generic systems? (verify)

as of this writing these are the multi-value gt signals and the value names in order. numbered parameter arguments are not shown:

```
Signals:Swing:Trend                    TrendUp, TrendDown
Signals:Swing:TrendEnding              TrendUpEnding, TrendDownEnding
Signals:Systems:MacdDiff               MacdDiffHigh, MacdDiffLow
Signals:Trend:HilbertChannelBreakout   HCBkUp, HCBkDown
Signals:Trend:TTT                      TTTUp, TTTDown
```

146 3.0 aliases

two forms of aliases are supported by gt. both are implemented in package GT::Tools. refer to that packages' pod for the most current implementation details.

3.2 global

refer to pod file GT/Docs/object_aliases.pod

3.2 object

refer to pod file GT/Docs/object_aliases.pod

147 4.0 gt application scripts

gt provides perl application scripts in the Scripts directory. these scripts combine the methods and functions found in the gt packages (files in the GT directory hierarchy) in ways to solve, implement or ??? a particular aspect of marketable securities technical analysis, manage a portfolio of stock holdings, and to evaluate gt signals, indicators and systems.

in general terms there are two categories of tools (the scripts): 1) those specifically intended for technical analysis of a security and 2) those that are most useful in terms of gt system development and validation tools. this distinction isn't hard and fast, but more of a general observation, guidance about what to use the scripts for.

4.1 gt securities analysis tools

-tbs-

1. 1.1 graphic.pl

   see the script pod for all the gory details.

   used primarily as a traditional technical analysis charting tool. the application takes directives consisting of signals, including a system component signal set, as well as indicators and creates a traditional technical analysis chart for the security specified.

   can be used to visually depict operation of signals, indicators and even the system component that can aid in their troubleshooting.

   in order to use graphic.pl you will need to rtfm;

   perldoc -t graphic.pl

2. 1.2 scan.pl

   tool to scan a number of securities with (or against) a number of system component signal sets (can be complete trading systems, but only the system component signal set is evaluated) to see which securities trigger which signals

   perldoc -t scan.pl

   also review perldoc -t GT/Docs/gt_files.pod for an all in one place description of the file formats used by scan.pl

3. 1.3 backtest.pl, backtest_many.pl, backtest_multi.pl

   three different variations on a traditional technical analysis tool that evaluates the performance of a specified trading system over the prior history of a specified securities trading data.

   (i thought one of these provided for user specified programmatic trading system parametric variation in an effort to optimize the system for the given security and its trading characteristics. but is don't see its use documented in my fast look-see)

4. 1.3.1 backtest.pl

   a traditional technical analysis backtesting tool that uses a specified trading system over the prior history of a specified securities trading data and analyzes the trading systems financial performance against a simple buy-and-hold strategy over the same time period.

   output includes optional html formatted text or simple ascii text plus optionally a very simple graphic depiction of the trading system compared to the buy-and-hold strategy.

   the html output option includes provision to embed the graphic image within the html page.

5. 1.3.2 backtest_many.pl

   somebody fixme

6. 1.3.3 backtest_multi.pl

   somebody fixme

7. 1.4 manage_portfolio.pl

   creates, maintains and analyzes your real (or imaginary) securities investment portfolio. a portfolio is a history of all transactions (orders and positions) that have taken place in the portfolio.

   takes input in the various forms including command line options, simple text files, beancounter database portfolio table

   stored as a gzipped xml file.

8. real portfolio management with manage_portfolio.pl

   by creating and maintaining a gt portfolio your positions can be marked on the graphic.pl charts using this graphic directive:

   ```
   --add=PortfolioPositions(./bc_pf,show)
   ```

   to update your bc_pf portfolio (i use bc_pf to indicate it is my beancounter porfolio, you can use any filename)

   ```
   %  manage_portfolio.pl bc_pf file ./trades_‘dex‘.scott2gt
   ```

   where the input file ./trades_‘dex‘.scott2gt has the appropriate format (refer to manage_portfolio.pl pod for details)

   the command dex is simply /usr/bin/date '+%d%b%y' | tr '[A-Z]' '[a-z]'

   trades are pasted into ./trades_‘dex‘ by manually scraping them off the brokers web page. that file is post-processed with a fairly trivial, but customized perl script that extracts the salient facts and arranges them to suit manage_portfolio.pl.

   to get a report on your portfolios' positions:

   ```
   %  manage_portfolio.pl bc_pf report positions
   ```

   to get a report on your portfolios' history:

```
% manage_portfolio.pl bc_pf report history
```

the report options performance and analysis need work if you can help please let the gt devel list know.

9. 1.5 analyze_backtest.pl
   somebody fixme
10. 1.6 anashell.pl
    somebody fixme
11. 1.7 select_combination.pl
    somebody fixme

4.2 gt development and test tools

-tbs-

1. 2.1 display_indicator.pl
   used primarily to develop and debug complex indicator descriptions.
   also useful to validate new or modified indicator packages
   perldoc -t display_indicator.pl
2. 2.2 display_signal.pl
   used primarily to develop and debug complex signal descriptions.
   also useful to validate new or modified signal packages
   perldoc -t display_signal.pl
3. 2.3 display_system.pl (just mine?)
   used primarily to develop and debug complex the two signals generated by the system component description.
   perldoc -t display_system.pl

148 5.0 developing gt systems

evaluating. proving, troubleshooting gt systems and other complex gt sig-indic descriptions

5.1 indicators

manually evaluate indicator results with display_indicator.pl.

this application will often show conditions like uninitialized values that can be masked when encountered in graphic.pl or backtest.pl.

compare a modified version of an indicator with an unmodified version

```
gdiff -u `! display_indicator.pl --sta 1jan09 --end 1may09 I:SnR   t` \
         `! display_indicator.pl --sta 1jan09 --end 1may09 I:SnR_h t`
```

the command ! is from 'unix power tools'. it turns it's input into a file and puts that filename in on the command line in its place.

and visually with graphic.pl

notice the same curve is plotted using the modified (I:SnR_h) indicator and the unmodified version with different colors to highlight any anomalous behavior.

```
graphic.pl --start 2008-10-01 --end 1may09 -timeframe day \
 --add=curve'( I:SnR/2, orange)' --add=curve'( I:SnR/1, darkgreen )' \
 --add=curve'( I:SnR_h/2, blue)' --add=curve'( I:SnR_h/1, red)' \
 T >/tmp/T8mon.png
```

it is also important to evaluate the result of passing different arguments to a newly developed indicator. in this instance we are checking that the indicator can handle the four arguments (e.g. 10 7 {I:Prices LOW} {I:Prices HIGH})

```
graphic.pl --start 2008-10-01 --end 1may09 -timeframe day \
 --add=curve'( I:SnR_h/2, blue)' \
 --add=curve'( I:SnR_h/2 10 7 {I:Prices LOW} {I:Prices HIGH}, green )' \
 --add=curve'( I:SnR_h/1, red)' \
 --add=curve'( I:SnR_h/1 10 7 {I:Prices LOW} {I:Prices HIGH}, black)' \
 T >/tmp/T8mon.png
```

when developing a new indicator one typically has an expected result in mind, furthermore, the default arguments are normally set to standard technical analysis values. this can often lead to 'code fixes' that yield the expectations when there are more fundamental aspects of the problem being overlooked. to avoid getting trapped in this condition it's important, and often very helpful to expose subtle coding errors and omissions by running test cases with a range of each arguments values. at the very least this will help ensure the code correctly handles arguments other than the hardcoded defaults and has reasonable protection against out-of-range argument values.

5.2 signals

-tbs-

manually evaluate signal generation with display_signal.pl and visually with graphic.pl

-tbs-

5.3 systems

mark system long/short triggers with graphic.pl

-tbs-

5.4 backtest.pl

when debugging or trying to evaluate a system try limiting and isolating individual trading system components using the techniques and the trading component specs listed below. the results can be useful to illuminate problems with the system being evaluated and to identify the trading component(s) that need to be adjusted.

a) isolate trades

```
-tra LongOnly
-tra ShortOnly
```

b) limit trades

```
-tra OneTrade
```

c) limit position entry/exit signals

```
define only one well understood position close signal
  --closestrategy='NeverClose'
  --closestrategy='LimitPeriodInTheMarket 10'
```

d) graph the results

```
  -display-trades -graph graphs/bt_code.png
  set the file name (argument to -graph) to whatever
  makes sense to you
```

5.4.1 backtest.pl examples

instead of

```
./backtest.pl -tim day -init 100000 -disp -gr graphs/a_bt_jpm.png \
 --start 2002-07-01 --end 2004-01-01 \
 -sy 'Generic
        {S:G:And
          { S:G:Below {I:Prices LOW} {I:G:Eval { I:SnR_h/1 } + 0.06 } }
          { S:G:Above {I:Prices LOW} { I:SnR_h/1 } }
        }
        { S:G:False }' \
 -clo 'LimitPeriodInTheMarket 5' \
 JPM
```

use

```
./backtest.pl -tim day -init 100000 -disp -gr graphs/a_bt_jpm.png \
 --start 2002-07-01 --end 2004-01-01 \
 -sy 'Generic
        {S:G:And
          { S:G:Below {I:Prices LOW} {I:G:Eval { I:SnR_h/1 } + 0.06 } }
          { S:G:Above {I:Prices LOW} { I:SnR_h/1 } }
        }
        { S:G:False }' \
 -mo 'Basic' \
 -tra 'LongOnly' \
 -tra 'OneTrade' \
 --closestrategy='NeverClose' \
 JPM
```

examples of other backtests.pl commands

```
./backtest.pl --full \
--system="Generic {S:G:CrossOverUp {I:SMA 20} {I:SMA 60}} \
 {S:G::CrossOverDown {I:SMA 20} {I:SMA 60}}" \
--closestrategy="Stop::KeepRunUp" \
--moneymanagement="FixedShares" \
--tradefilter="LongOnly" \
--orderfactory="ClosedToClose" \
--broker="InteractiveBrokers" \
CCRT

./backtest.pl --start 2008-01-02 --end 2009-04-30 \
--system="Generic {S:G:CrossOverUp {I:SMA 20} {I:SMA 60}} \
 {S:G:CrossOverDown {I:SMA 20} {I:SMA 60}}" \
--closestrategy="Stop::KeepRunUp" \
--moneymanagement="FixedShares" \
--tradefilter="LongOnly" \
--orderfactory="ClosedToClose" \
--broker="InteractiveBrokers" \
CCRT

backtest.pl --timeframe week --start 1apr08 --end 1may09 \
"'`cat buysell-signal | perl -ne 'print $_ unless  m/^\s*#.*$|^\s*$/'`'" \
MIDD

backtest.pl --timeframe week --start 1apr08 --end 1may09 \
"`cat buysell-signal`" \
MIDD
```

```
backtest.pl -tim day -init 100000 -disp -gr graphs/bt_aapl.png \
-sy 'TFS 15 7' -mo 'Basic' \
-mo 'ShareMultiples 100 2' \
-clo 'Reinvest:InWinners 2' \
-clo 'ChannelBreakout { I:SMA 80 } { I:SMA 80 }' \
-tra LongOnly \
-tra OneTrade \
AAPL
```

5.4.2 you've got the code, use it

make a copy of the package file 'before' you start hacking on it. then hack away on the copy.

```
hack in
  'use Data::Dumper;'
and then sprinkle
 'print STDERR Dumper data_object;'
statements around and see what comes out.
you might want to send this output to a file since it can get
copious quickly.
```

Data::Dumper is recommended because you don't have to be concerned with the data type being printed. it handles scalars (simple variables), arrays, and complex gt data objects with ease.

5.4.3 self hacking guidance

with package file GT/Indicator/SnR_h.pm under active development add these perl statements towards the top of the file:

```
our $debug = 0;
    $debug = 1; # enable debug outputs
```

now you can add print statements like these to monitor internal variable without using a debugging perl

```
if ( $debug > 1 ) {
  my $ioff = $self->{'args'}->get_arg_names(2) + 1;
  print STDERR "\n";
  print STDERR "min1[ $i - $ioff ] $min1\n";
  print STDERR "min2[ $i ] $min2\n";
  print STDERR "\n";
  print STDERR "max1[ $i - $ioff ] max1\n";
```

```
  print STDERR "max2[ $i ] $max2\n";
  print STDERR "\n";
}
```

in addition, you can easily disable the debugging output by commenting out the $debug = 1; # enable debug outputs line but leave in most of your debug bits to share/discuss or retain for later.

also note that you can easily arrange for various levels of debug output using various the comparison conditional values

```
if ( $debug ) { ...

if ( $debug >= 2 ) { ...
```

the more adventurous might do things more like this, but probably a lot smarter and better than this

```
my $BIT1 = 0x01; my $B1MSK = ~0x01;
my $BIT2 = 0x02; my $B2MSK = ~0x02;
my $BIT3 = 0x04; my $B3MSK = ~0x04;
my $BIT4 = 0x08; my $B4MSK = ~0x08;

my $debug = $BIT2 + $BIT4;

if ( $debug & $B4MSK ) { ...
```

finally use a bash command like

```
$ display_indicator.pl -short 35 --start 1jan09 --end 1may09 \
I:SnR_h T > /tmp/snr.out 2>&1
```

or for csh

```
% display_indicator.pl -short 35 --start 1jan09 --end 1may09 \
I:SnR_h T >& /tmp/snr.out
```

with instrumented code you need to keep separate stdout and stderr for scripts that write image files to stdout like graphic.pl.

for this use a bash command like

```
$ graphic.pl -file example.gconf ENS 1> ENS.png 2> example.out
```

or for csh

```
% ( graphic.pl -file example.gconf ENS > ENS.png ) >& example.out
```

6.0 new indicator and signal code development

refer to Writing_an_Indicator_Cookbook.pdf (GT/Docs/Writing_an_Indicator_Cookbook.pdf), the seminal work on how to develop new gt indicators and by extension signals.

while it may still seem unnecessary when initially developing a brand new indicator or signal, the statement

```
return if (! $self->check_dependencies($calc, $i));
```

or the preferred form

```
return unless $self->check_dependencies($calc, $i);
```

is how an indicator or signal gets 'set up'. (if anyone can improve on the terminology here that would be great) without this statement you can expect the indicator or signal will return nothing. what 'set up' means is everything is solved if it isn't already. if the necessary dependencies cannot be solved the indicator or signal will probably fail (without raising an exception, either methinks, and that is unfortunate). if you're interested in how this all works read the code, it's in GT/Dependency.pm, and if you can figure a way to raise an exception in appropriate circumstances please do so and contribute the code.

149 object aliases

object aliases are user defined aliases, shorter names for commonly used gt sys-sig-indic descriptions. object aliases can be used in any of the gt sys-sig-indic descriptions, as well as in gt trading system component specifications like CloseStrategy, TradeFilters and OrderFactory. it isn't clear whether an object alias may be used to define multiple components of a trading system (e.g. contain a connecting '|' symbol), but the code commentary, and pod implies that is not their intended purpose. for that it's probably just wiser to use a system alias (e.g. global alias) (refer to -tbd-).

the intention of some of the ras version changes in various object aliases handling methods is to not permit object aliases to define multiple system components (e.g. contain a '|' symbol). however, sufficient code may not be in place to guarantee that attempts to do that will always be detected and flagged with a useful error message.

so don't do it. use a system alias (global alias) for that purpose.

NB: many of the object alias file locations and some object alias file processing attributes described here may only be implemented in specific ras hack versions of the gt toolkit. see the section 'some important notes' for some more details. contact ras directly (or start a devel list message thread) if any of these features are needed in your particular gt use model.

object aliases – @I::my_object_alias

to use an object alias in a gt sys-sig-indic description you prepend to the object aliases name a string that starts with the '@' symbol, then add the type designator ($kind) (e.g. Indicator, Signal, etc) followed by one or two colons ':' and then finally insert the name of the object alias itself. it's recommended that you use the standard gt sys-sig-indic abbreviations in place of the type designators (e.g. $kind) (e.g. Indicators, Signals, Systems, etc) (see GT::Tools pod). for example, if you have an object alias named my_mean and it is an indicator type ($kind is indicator) it would appear in a gt sys-sig-indic description as:

```
@I:my_mean
or possibly @I::my_mean
or @Indicators::my_mean (not recommended)
```

more examples of object aliases definition as well as usage will be presented as the topic is developed.

150 object alias files

object alias files are simple text files that specify an object alias and it's value. in a perl language context these files are configuration files in that they have a key-value pair record structure.

there are two different formats that object aliases come in. which form is used to define the object aliases depends on the particular file location (pathname) the object aliases is defined in. regardless of the object alias format type the actual value of the object alias remains the same, but the key string varies depending on the file location.

the files (locations) that can define an object aliases are

```
$HOME/.gt/options
```

and

```
files in the directory $HOME/.gt/aliases named in accordance
with the $kind string (see below)

and/or the files identified by the value of gt config keys
of the form "Path::Aliases::$kind" (case insensitive),
defined in the users $HOME/.gt/options file.
```

in the absence of any explicit directive, additional (global) object alias files will be searched for in the gt install directory – generally defined as the directory just above the current working directory. because most gt users install gt and then cd to Scripts to use the gt toolkit to perform ta analyses, the typical cwd is the gt Scripts dir (this may not apply when using the cpan branch, or some other installation model). in order to support all (most) other installation models there are a number of explicit ways to specify alternate paths for these global or shared object alias file locations as described in the paragraph 'object alias file locations and file names' below.

the variable $kind assumes each of the following strings during object alias processing and file loading:

```
signals
indicators
systems
closestrategy
moneymanagement
tradefilters
orderfactory
analyzers
```

these $kind strings have no uniqueness abbreviations nor any mis-spelling tolerance, thus 'indicator', 'signal' and 'system' will all fail.

the value of $kind, when used as a key (left hand side of the key-value pair), it is not case sensitive but when the $kind string is used in a value context (right hand side of the key-value pair) it is case sensitive and by default will be lower case string as listed above.

151 object alias formats

there are two different forms of object alias formats depending on the file in which the object alias is defined.

object alias format a

this form of object aliases can only be used in the $HOME/.gt/options file.

```
 "Aliases::$kind::"<object_alias> <whitespace> { <object_alias specification> }
```

specific format details

```
the string "Aliases::$kind::" must be prepended (without the quotes)
to the object_alias name (<object_alias>).

the object_alias name is any proper perl string but it should
avoid these problematic characters '@#{}[]+-*/'.

the resulting string (e.g. Aliases::$kind::object_alias_name)
is used as a gt config hash key: as a consequence the key string
will be stored lower case, but is case insensitive.
the value of the key will be unaltered and will be case sensitive.

the key string (e.g. Aliases::$kind::object_alias_name) is
separated by whitespace from the object alias <object_alias specification>.

the object alias definition <object_alias specification>
is enclosed in curly braces '{...}'.
( statement should be validated and verified )

the object alias definition <object_alias specification> value
will consist of the remaining characters on the current logical
line (if continuation and comment line support is present in the
gt toolkit version for the $HOME/.gt/options file the logical line
ends when a line is encountered that doesn't end with a '\' character
immediately before the newline character).
```

object alias format a examples

here are working examples – valid only in your $HOME/.gt/options file

```
Aliases::Indicators::MyMean_opt       { I:Generic:Eval ( #1 + #2 ) / 2 }
Aliases::Indicators::PVOL_opt         {I:Prices VOLUME}
# example use: display_indicator.pl @I:MyMean_opt GOOG '50 {I:RSI}'
#              display_indicator.pl @I:pvol_opt GOOG
```

object alias format b

the second form of an object alias format can only be used in object alias files where the alias type information (i.e. $kind) is derived from the files pathname. this object alias format omitd the string "Aliases::$kind::" from the object_alias name (<object_alias>). the object alias processing and storage remains the same as previously described so the preceeding formatting details apply with this minor variation:

```
the object alias definition <object_alias specification> value
will consist of the remaining characters on the current logical
line (if continuation and comment line support is present in the
gt toolkit version for object alias files the logical line
ends when a line is encountered that doesn't end with a '\' character
immediately before the newline character).
```

the type b object alias format:

```
<object_alias> <whitespace> { <object_alias specification> }
```

format b applies to object aliases files (private files) named based on the $kind strings in the directory $HOME/.gt/aliases or in other, possibly shared, directories as will be described later.

object alias format b examples

the examples below are of $kind indicator, so they would appear in a file named indicator in an aliases directory at $HOME/.gt/ or at some other directory path in the file system or in an absolute pathname identified via the key "Path::Aliases::Indicator".

```
MyMean_gbl    { I:Generic:Eval ( #1 + #2 ) / 2 }
PHI_gbl       {I:Prices VOLUME}
PO    {I:Prices OPEN}
PH    {I:Prices HIGH}
```

```
Pl    {I:Prices LOW}
PC    {I:Prices CLOSE}
PV    {I:Prices VOLUME}
vol   {I:Prices VOLUME}
```

the following examples are of $kind signals, thus they might appear in a file named $HOME/.gt/aliases/signals.

```
3Xoverlong    { S:G:Or { S:G:And {S:G:Above #1 #3} {S:G:CrossOverUp #2 #3} } { S:G

3Xovershort   { S:G:Or \
  { S:G:And {S:G:Below #1 #3} {S:G:CrossOverDown #2 #3} } \
  { S:G:And {S:G:Below #2 #3} {S:G:CrossOverDown #1 #3} } \
}

3EMAlong      { @S:3Xoverlong {I:EMA #1} {I:EMA #2} {I:EMA #3} }

3EMAshort     { @S:3Xovershort {I:EMA #1} {I:EMA #2} {I:EMA #3} }
```

152 object alias file locations and file names

the gt system will look in multiple directories for object alias files. if the paths (directories) do not exist or there are no appropriately named files found in the directories gt will silently ignore the situation. negative conditions will neither cause a warning nor an error, this can be problematic or not depending ...

object alias files are sought in the file system in two places:

1. $HOME/.gt/aliases. files therein must be named after $kind. this is a fixed pathname and it cannot be altered. however the absense of the directory $HOME/.gt/aliases will inhibit the reading and processing of any user private object alias files.

   note: alias files must be name after $kind (e.g. $kind takes on each of these strings 'signals', 'indicators', 'systems', 'closestrategy', 'money-management', 'tradefilters', 'orderfactory', 'analyzers') and must be lower case (if your os/fs is case sensitive).

   pathname examples: $HOME/.gt/aliases/signals $HOME/.gt/aliases/indicators $HOME/.gt/aliases/systems $HOME/.gt/aliases/closestrategy

2. default global (possibly for shared) object aliases files. the location and naming of this collection of object aliases files can be controlled by each GT user via gt configuration key-values and/or an environment variable.

   by default the directory for global (shared) object aliases is sought in one of up to five locations, in priority order:

```
i)   the value of multiple gt config keys "Path::Aliases::$kind".
     this feature will override the pathname for each $kind global
     object alias file, so a user may/must have a gt config key
     for each $kind of object alias pathname set in this manner.

ii)  the users value in gt config key "Path::Aliases::Shared".

iii) the users value in gt config key "GT::Root".

iv)  the users environment variable "${GT_ROOT}".

 v)  one directory above the current working directory.
```

item i) overrides items ii) .. v) for $kind object aliases only.

items ii) .. v) are mutually exclusive – the value assigned by the key sets the directory path where global (shared) object alias files of $kind are sought. if no files are found or the directory does not exist the condition is silently ignored.

items ii) .. v) apply to all shared object alias $kind files not overridden by item i).

(the preceeding statement needs to be validated and verified)

each object alias file found via key-value pairs per items ii) .. v) must be named one of the $kind strings (see above).

NB: the default $kind filenames are lower case, but if you elect to change the default name via item i) whatever value is specified will be used as the filename unchanged.

example assuming there is no explicit gt root setting (e.g. item v) above) (pathname shown relative to current working directory)

```
../aliases/indicators
../aliases/systems
../aliases/signals
../aliases/closestrategy
```

same example with the gt root directory explicity set using a gt config "GT::Root" key-value pair:

```
GT::Root       /usr/share/geniustrader
```

files with these pathnames will be sought:

```
/usr/share/geniustrader/aliases/indicators
/usr/share/geniustrader/aliases/systems
/usr/share/geniustrader/aliases/signals
/usr/share/geniustrader/aliases/closestrategy
```

similarly you can use gt config key "Path::Aliases::Shared" to set the directory path in the case when "GT::Root" isn't a suitable directory for object alias files. "Path::Aliases::Shared" overrides "GT::Root". gt config key "GT::Root" overrides envar "${GT_ROOT}". default global location '..' relative to cwd.

complete per user object aliases directory controls

if the default paths and file names (e.g. $kind) are unacceptable for some reason, do not despair or attempt to change the code. you can easily specify alternate paths and file names for the global object alias files using the existing naming features.

this might be done to facilitate two or more users sharing object aliases files. or it might be needed to support multiple sets of object aliases files or whatever.

alternate paths are under full user control using the gt %conf option keys "Path::Aliases::$kind" to establish new paths as well as the filenames used for object alias files.

```
Path::Aliases::$kind  /this/users/preferred/path/alias_file
```

will cause gt to search /this/users/preferred/path/ for the file named 'alias_file' hopefully containing type b $kind object aliases. if the file or path does not exist it will be silently ignored. this is both a blessing and a curse, depending ...

it is required that any key specified be in the set of $kind name strings (see above) but the value for the key (e.g. the value of these keys is the absolute pathname of the file containing $kind object aliases) is entirely up to the user. the value will be case sensitive depending on your os/fs. it is expected that the object aliases found at the path associated with the $kind key be of type $kind. in other words, the file named by the value of key Path::Aliases::Indicators should contain indicator aliases, and key Path::Aliases::systems should contain system aliases.

by way of NEGATIVE example, while the following are operationally correct they are NOT recommended practice.

```
Path::Aliases::signals         /home/shared/some_aliases
Path::Aliases::indicators      /home/shared/objects
Path::Aliases::systems         /etc/gtsys
Path::Aliases::closestrategy   /cs
```

as shown, the user is free to define different directories as well as alternate filenames for the global (shared) object alias files. the effect of putting say signal object aliases in files expected to be indicators hasn't been evaluated, but the expected result is (or should be) that the signal alias would simply not be found.

use suggestion: if you must have different directories object alias files, for the sanity of your maintainers as well as your users, use $kind or filenames that somehow relate to $kind where ever else you set the path.

in other words:

```
Path::Aliases::signals                /home/shared/GT/signals
Path::Aliases::indicators     /home/shared/devGT/indicators
Path::Aliases::systems                /etc/systems
Path::Aliases::closestrategy  /var/shared_GT_oa/closestrategy
```

pathname upper/lower case observations

the $kind filenames in dir tree $HOME/.gt/aliases will be lower case.

the default global $kind filenames will be lower case.

if you change the default $kind pathname using any of the config options ("Path::Aliases::$kind", "Path::Aliases::Shared", "GT::Root", or envar "${GT_ROOT} whatever pathnames you specify will be used without change.

gt config option keys are always case insensitive and are stored lower case.

completely valid object alias files stored in an appropriate directory but having a case conflict will be silently ignored. suggestion: avoid using upper case pathnames.

153 other features

object aliases can include other object aliases

object aliases support recursion. this makes the combination of system and object aliases a powerful and useful feature.

object aliases support numbered parameteric substitution

if an object alias specification includes arguments encoded with a hash '#' followed by an integer (generally delimited with whitespace) the actual argument value used for that parameter will be the value of the argument that corresponds with the integer.

say what!?

as a fairly useless example say we define an indicator object alias

```
P     {I:Prices #1}
```

when used in a sys-sig-indic the parameter that is represented by '#1' must be provided as the first argument to the sys-sig-indic. in the following examples argument #1 is either LOW or VOLUME.

```
display_indicator.pl @I:P GOOG LOW
display_indicator.pl @I:P GOOG VOLUME
```

a more useful example is one defined earlier:

```
 MyMean_gbl    { I:Generic:Eval ( #1 + #2 ) / 2 }
```

which can be used as follows:

```
display_indicator.pl @I:MyMean_gbl GOOG '{I:Prices HIGH} {I:Prices LOW}'
```

in this case #1 has the value "{I:Prices HIGH}" and #2 "{I:Prices LOW}"
as an example of recursion (i think), object aliases

```
 PH    {I:Prices HIGH}
 Pl    {I:Prices LOW}
```

can be used as follows:

```
display_indicator.pl @I:MyMean_gbl GOOG @I:PH @I:Pl
```

examples

For example, define these signal object aliases in your $HOME/.gt/aliases/signals file

```
# 3Xover... is a generic crossover signal involving three lines
3Xoverlong { S:G:Or \
 { S:G:And {S:G:Above #1 #3} {S:G:CrossOverUp #2 #3} } \
 { S:G:And {S:G:Above #2 #3} {S:G:CrossOverUp #1 #3} } }

3Xovershort { S:G:Or \
 { S:G:And {S:G:Below #1 #3} {S:G:CrossOverDown #2 #3} } \
 { S:G:And {S:G:Below #2 #3} {S:G:CrossOverDown #1 #3} } }

# 3EMA... passes its arguments to and invokes 3Xover...
3EMAlong  { @S:3Xoverlong {I:EMA #1} {I:EMA #2} {I:EMA #3} }

3EMAshort { @S:3Xovershort {I:EMA #1} {I:EMA #2} {I:EMA #3} }
```

with a (global) system alias (see -tbd-) named 3EMA defined in your $HOME/.gt/options file:

```
Aliases::Global::3EMA          SY:Generic \
 { @S:3EMAlong 60 90 120 } { @S:3EMAshort 60 90 120 }
```

you can evaluate initial positions using the aliased signals defined. note however, the arguments are fixed.

with a similar system alias named 3EMA[] in your $HOME/.gt/options file:

```
Aliases::Global::3EMA[]        SY:Generic \
 { @S:3EMAlong #1 #2 #3 } { @S:3EMAshort #1 #2 #3 }
```

you can pass different arguments to the aliased signals using the numbered parameter substitution feature. use this alias in a sys-sig-indic desc like

```
SY:3EMA[ 20 50 150 ]
```

include the square brackets '[' ']' as shown and delimit the parameters with whitespace.

and if you don't want the parameter symmetry (e.g. using same args for the long and short position triggers) you can redefine everything with parameters numbered #1 to #6

```
Aliases::Global::3EMA[]        SY:Generic \
 { @S:3EMAlong #1 #2 #3 } { @S:3EMAshort #4 #5 #6 }
```

and use a sys-sig-indic desc like SY:3EMA[20 50 150 15 40 120]

some important notes:

1. it's critical that the leading '{' is present in each object alias otherwise the arguments don't get fully substituted
2. continuation and comment lines for object alias files is supported only with the ras versions of methods GT:Tools::resolve_object_alias and GT:Conf::load. the current trunk version of GT::Conf::load performs the initial object alias file reads internally but doesn't handle continuation or comment lines for them. also, the trunk version of method GT:Conf::load is now technically broken (flawed): if called multiple times, say to read an additional gt configuration file, object alias processing will be repeated unnecessarily. finally, the current ras versions of the modules GT:Tools and GT:Conf also diverge from the trunk versions in other ways as well, so it isn't a simple matter of updating just these two modules.
3. blank lines and lines starting with # are ignored (possibly may apply to ras versions only)

4. lines may contain a trailing comment provided there is whitespace following the object alias definition and the comment character '#'. (possibly may apply to ras versions only)

154 GT::Eval

Create unknown standard objects at run-time

DESCRIPTION

This modules provides several functions to manipulate objects based on their type name.

`$object = create_standard_object($object_type, ...)`

: This will create an object of type $object_type. The following parameters will be passed to the object at creation time.

`create_db_object()`

: Return a GT::DB object created based on GT::Conf data. Thus GT::Conf::load() should be called before this function. If DB::module doesn't exist in the config, it tries to load the user configuration (supposing it has never been done before).

`get_standard_name($object, $shorten, $number)`

: Return the standard name of an object that can be later used to create it again.

`get_long_name ($code)`

: Returns the long name of the market (if defined).

  See also ~/.gt/sharenames which contains lines of the form <code>\t<long name> mapping a market code to its long name.

155 GT::Graphics::Axis

An axis can be displayed on a side of a Zone. It's associated to a scale. It precises how much space there's between ticks.

GT::Graphics::Axis->new($scale)

Create a new axis and use the associated scale.

$a->set_{left,right,top,bottom}_side()

Indicate that the axis is on the corresponding side of the graphic.

$a->set_zone($zone)

Indicate the zone attached to the axis.

$a->set_rectangle($x1, $y1, $x2, $y2)

Indicate the rectangle in which the axis should be displayed. The rectangle is defined by the lower left corner (x1,y1) and the upper right corner (x2,y2).

$a->set_space_for(_big)_ticks

Define the space between (big) ticks.

$a->set_grid_level({0|1|2})

Indicate if a grid should be displayed : - 0 => no grid - 1 => grid on big ticks - 2 => grid on all ticks

$a->set_grid_color($color)

Use the indicated color for the grid.

$a->label_display({0|1})

Tell if labels should be displayed or not.

$a->set_custom_ticks([[$x,$label], ...], $below_zone)

Use the given custom ticks. If $below_zone is set to one, the labels will be displayed below the zone starting at the given coordinate, otherwise it will be displayed right below the given coordinate (ie below the tick).

$a->set_custom_big_ticks([[$x,$label], ...], $below_zone)

Use the given custom big ticks. If $below_zone is set to one, the labels will be displayed below the zone starting at the given coordinate, otherwise it will be displayed right below the given coordinate (ie below the tick).

$a->display($driver, $picture)

Display the axis on the picture.

156 GT::Graphics::DataSource

A datasource if a source of data for a graphical object. The datas are always indexed by an integer.

157 FUNCTION TO IMPLEMENT

Each real datasource has to implement a few functions :

Constructor : $ds->new(...)

A constructor for the datasource has the right to have parameters. When constructed it should update the available range and set the selected range to the available range.

$ds->get($index)

Return the data associated to the corresponding index.

$ds->is_available($index)

Tell if the data is available for the corresponding index.

$ds->update_value_range()

Update the minimum value and the maximum value.

158 GENERIC FUNCTIONS AVAILABLE

($start, $end) = $ds->get_selected_range()

Return the range of selected data.

$ds->set_selected_range($start, $end)

Set the range of selected data.

($start, $end) = $ds->get_available_range()

Return the range of available data.

$ds->set_available_range($start, $end)

Set the range of available data.

($min, $max) = $ds->get_value_range()

Return the minimum and the maximum of the values available within the selected range.

$ds->set_min_value($min)

Set the minimum value.

$ds->set_max_value($max)

Set the maximum value.

159 GT::Graphics::DataSource::Close

This datasource provides close price information. It uses a GT::Prices object as a basis.

GT::Prices::DataSource::Close->new($prices)

Create a new close prices data source.

160 GT::Graphics::DataSource::GenericIndicatorResults

This datasource is a generic module to handle any information provided by an indicator.

Details

This module will return either a serie of single data or a serie of array.

If the input arguments contains a string like "Indicators::BOL/99", we will assume that the user wants to have all data available form the Bollinger indicator and we will return an array with all the data.

If the input arguments contains a string like "Indicators::BOL" (It's the same than "Indicators::BOL/0"), "Indicators::BOL/1" (or / any number), we will assume that the user only wants a serie of single data, which is in our example the third serie (keep in mind that the first serie start at zero).

We will either use only a single data serie or a all data available for the calculation of the value range.

GT::Graphics::DataSource::GenericIndicatorResults->new($calc, $indicator_desc)

Create a new indicator data source.

161 GT::Graphics::DataSource::PortfolioEvaluation

This datasource provides the evaluation of a portfolio.

GT::Graphics::DataSource::PortfolioEvaluation->new($calc, $portfolio)

To create a new portfolio evalution datasource object, you need to give a calculator and a portfolio as parameters.

162 GT::Graphics::DataSource::Prices

This datasource provides prices information. It uses a GT::Prices object as a basis.

GT::Prices::DataSource::Prices->new($prices)

Create a new prices data source.

163 GT::Graphics::DataSource::PricesColor

This datasource provides a color depending on the prices movement. Green when up, red when down, black when equal.

It uses a GT::Prices object as a basis.

GT::Prices::DataSource::PricesColor->new($prices)

Create a new prices color data source.

$pc->set_{up,down,unchanged}_color($color)

Change the color returned for the up/down/unchanged days.

164 GT::Graphics::DataSource::SingleIndicator

This datasource is a generic module to handle any information provided by an indicator.

We will return a serie of data based on a single generic indicator name, like "Indicators::BOL/2".

165 GT::Graphics::DataSource::Systems

This datasource is a generic module to handle any information provided by a system.

166 GT::Graphics::DataSource::Volume

This datasource provides volume information. It uses a GT::Prices object as a basis.

GT::Prices::DataSource::Volume->new($prices)

Create a new volume data source.

167 GT::Graphics::Driver

A graphic driver is a well defined interface that let you actually generate a picture by using drawing primitives. Those primitives are used by "drawable" objects that implements the display($driver, $picture) method.

168 Drawing API

$picture = $driver->create_picture($rootzone)

This does create the empty picture on which you'll draw various things. The picture has the size corresponding to the given "zone".

$driver->line($picture, $x1, $y1, $x2, $y2, $color)

$driver->dashed_line($picture, $x1, $y1, $x2, $y2, $color)

$driver->antialiased_line($picture, $x1, $y1, $x2, $y2, $color)

$driver->rectangle($picture, $x1, $y1, $x2, $y2, $color)

$driver->filled_rectangle($picture, $x1, $y1, $x2, $y2, $color)

$driver->polygon($picture, $color, @points)

$driver->filled_polygon($picture, $color, @points)

$driver->circle($picture, $x, $y, $width, $height, $color)

The last 7 methods are simple drawing methods. The coordinates are absolute (ie as expressed in the $rootzone). Take care to convert them if needed.

For the rectangles, you give the lower left corner and the upper right corner.

The (0,0) coordinate is the lower left corner.

my $oldwidth = $driver->line_width($picture, $width)

This method changes the default width of displayed lines. It returns the previous width so that you can restore it to its previous value once you're finished with the operation needing a special line width.

If $width isn't given, it only returns the actual width.

$driver->string($p, $name, $size, $color, $x, $y, $text, $halign, $valign, $orientation)

This method is used to draw texts on the picture.

Horizontal/vertical align : $ALIGN_LEFT, $ALIGN_CENTER, $ALIGN_RIGHT

Orientation : $ORIENTATION_UP, $ORIENTATION_DOWN, $ORIENTATION_RIGHT, $ORIENTATION_LEFT

The text is displayed near (x,y) by follwing the required alignments.

169 Output API

$driver->save_to($p, $filename)

Save the picture in the given filename.

$driver->dump($p)

Dump the picture to the standard output.

170 Generic functions

Those functions don't need to be reimplemented, they are implemented with the other primitives.

$driver->cross($picture, $x1, $y1, $x2, $y2, $color)

Draw a cross.

171 DATA STRUCTURE

Colors

Colors are simple RGB triplet associated to an alpha channel : [R, G, B, A] They are only array references.

Some variables are available for the most common colors :

$COLOR_WHITE
$COLOR_BLACK
$COLOR_RED
$COLOR_GREEN
$COLOR_BLUE

Fonts

Font names are simple strings (true type font names). Font size are numbers.

Some variables are available for the most common values :

$FONT_SIZE_TINY
$FONT_SIZE_SMALL
$FONT_SIZE_MEDIUM
$FONT_SIZE_LARGE
$FONT_SIZE_GIANT

$FONT_ARIAL
$FONT_TIMES
$FONT_HELVETICA
$FONT_SERIF
$FONT_SANS_SERIF
$FONT_FIXED
$FONT_PROPORTIONNAL

172 GT::Graphics::Driver::GD

The GD driver implements the drawing primitives using the GD module that lets you create PNG images.

173 GT::Graphics::Driver::ImageMagick

This driver implements the drawing primitives using the ImageMagick Perl extension.

174 GT::Graphics::Driver::Postscript

This driver implements the drawing primitives using the PostScripts::Simple-module.

175 GT::Graphics::Driver::SVG

Overview

This driver implements the drawing primitives using the SVG module available in CPAN to create Scalable Vector Graphics (SVG) files, which is an exciting new XML-based language for Web graphics from the World Wide Web Consortium (W3C).

Links

http://www.adobe.com/svg/main.html http://www.w3.org/TR/SVG/ http://www.roasp.com/

176 GT::Graphics::Graphic

A graphic is composed of a layout of zones. Objects are affected to the various zones. Those objects may be displayed. The display engine may use an associated default scale to obtain coordinates of points to draw.

GT::Graphics::Graphic->new($zone)

Create a new graphic using the specified zone layout.

$graphic->set_zone($zone)

Define the layout of the display zones. You shouldn't call this once you added graphical objects because objects may reference zones that are no more part of the new layout.

$graphic->set_background_color($color)

Set the background color of the graphic.

$graphic->add_object($object)

Add a graphical object to the graphic.

$graphic->display($driver, $picture)

Display the graphic in the picture. It will display the zones and the graphical objects.

177 GT::Graphics::Object

A graphical object is a part of a graphic. It can display itself on a picture.

178 FUNCTIONS TO IMPLEMENT

Each graphic object will have to implement these two functions (methods).

$o->init(...)

This function is called with the trailing arguments (e,g, $args[2] and up) given to the generic new constructor defined here.

$o->display($driver, $picture)

Display the graphic object on the picture using the given driver. It may use $o->{'source'} and $o->{'zone'} (e.g. the graphic objects argument hash $self->{'source'} and $self->{'zone'} to get the data to display and display itself in the correct zone.

179 GENERIC FUNCTIONS

my $obj = GT::Graphics::Object::<Something>->new($datasource, $zone, ...)

The generic new constructor declared and defined here takes the first 2 arguments (e.g. $datasource and $zone) and assigns them to the objects argument hash $self->{'source'} and $self->{'zone'}, respectively, and passes the remaining arguments, if any, to the init method which must be provided by the GT::Graphics::Object::<Something> package.

in addition, the new constructor sets the objects argument hash keys 'bg_color' and 'fg_color' (e.g. $self->{'bg_color'} and $self->{'fg_color'}) to the gt config key-values corresponding with Graphic::BackgroundColor and Graphic::ForegroundColor respectively.

my $o_z_level = $o->get_z_level()

$o->set_z_level($z)

Those two functions are used to manage the order in which the orders are displayed. An object with a low Z level is drawn first.

my $o_datasource = $o->get_source()

$o->set_source($source)

set/get the datasource associated to this object.

$o->set_zone($zone)

Set the zone in which the object will be displayed.

$o->set_special_scale($scale)

Use a special scale to draw this object.

my $o_scale = $o->get_scale()

Return the associated scale. If it exists, it uses the special scale, otherwise returns the default scale associated to the zone.

$o->set_background_color($color)

Use this color as background color.

$o->set_foreground_color($color)

Use this color as foreground color.

my $o_scale = $o->get_background_color()

return the objects background color.

my $o_fgc = $o->get_foreground_color()

return the objects foreground color.

180 GT::Graphics::Object::BarChart

This graphical object display a serie of bars.

181 GT::Graphics::Object::BuySellArrows

This graphical object display buy and sell arrows.

Description and Usage

the object doesn't accept arguments when created, however it reads gt configure file for and sets these values (default values are indicated):

```
Graphic::BuySellArrows::BuyColor      "green"
Graphic::BuySellArrows::SellColor     "red"
Graphic::BuySellArrows::Distance      8
Graphic::Candle::Height               3
Graphic::BuySellArrows::SizeFactor    1
```

note: the default SizeFactor of 1 should make this modified version work identically to the prior version

personally, i find a Distance of about 24 and a SizeFactor of 3 to 6 makes the arrow plot better. in addition i prefer to darken the colors and make the partly transparent

```
Graphic::BuySellArrows::BuyColor      "[0,135,0,64]"  # very dark green
Graphic::BuySellArrows::SellColor     "[150,0,0,64]"  # dark red
```

used in graphic.pl graphic configuration file as shown below:

plotting on the primary price plot

```
--add=BuySellArrows(Systems::Generic \
 { S::Generic::CrossOverUp   {I::MACD/1 26 52 20} {I::MACD/2 26 52 20} } \
 { S::Generic::CrossOverDown {I::MACD/1 26 52 20} {I::MACD/2 26 52 20} } \
 )
```

plotting arrows on a secondary zone

–add=New-Zone(5)

–add=New-Zone(100) –add=Curve(I:MACD/1 26 52 20, [120, 40, 0]) –add=Curve(I:MACD/2 26 52 20, red) –add=Text("macd: 26 52 20", 2, 95, left, center, small, [120, 40, 0], arial) –add=Set-Scale(auto)

–add=BuySellArrows(Systems::Generic \ { S::Generic::CrossOverUp {I::MACD/1 26 52 20} {I::MACD/2 26 52 20} } \ { S::Generic::CrossOverDown {I::MACD/1 26 52 20} {I::MACD/2 26 52 20} } \ ) –add=Set-Special-Scale(auto,"log")

182 GT::Graphics::Object::Candle

This graphical object display a series of candlesticks.

183 GT::Graphics::Object::CandleVolume

This graphical object display a series of candlesticks. The width of each candle is determined by the volume.

184 GT::Graphics::Object::CandleVolumePlace

This graphical object display a series of candlesticks. The width of each candle is determined by the volume.

185 GT::Graphics::Object::Curve

This graphical object display a curve.

186 GT::Graphics::Object::Histogram

This graphic object displays a histogram.

used in graphic.pl it takes two arguments, an indicator and a color. it plots vertical bars (histogram) from zone zero to data value.

the histogram default color is "yellow", but can be changed via configuration option "Graphic::Histogram::Color" and by the color parameter on the graphic option statement. in addition the color can be controlled by an indicator as well (see examples).

bars will be clipped at upper and lower zone boundries. clipped bars will display a small arrow at the clipped end. there are two hardcoded variations that set the color of this marker. the old way requires you to edit the file and set the hash variable $self->{'inverse'} in the sub init method to any value.

```
    $self->{'inverse'} = 'enable';
```

in this mode the clip marker color will be the inverse of the current color. benefits include good contrast with respect to the histogram especially when an indicator is controlling the color.

currently the clip marker color will be either the default value "blue" or the value set by your configuration option parameter "Graphic::Histogram::ClipColor"

187 examples

```
typical in a graphic config file
  --add=Histogram(I:MACD/3 26 52 20, brown)
  --add=Histogram(I:MACD)
  --add=Histogram(I:VOSC 21, [127,127,127])
  --add=Histogram(I:ADL, "dark blue")

using indicator to set historgram color: (in options file only)
  Graphic::Histogram::Color Indicators::Generic::If \
   {Signals:Generic:Below {I:Prices OPEN} {I:Prices CLOSE}} green red
```

$hist->set_color_datasource($ds)

Use the indicated datasource to retrieve the color of the bar.

188 GT::Graphics::Object::Marks

This graphical object display a serie of '+' marks.

It needs a data source with a single value per coordinate pair (in other words typical datasource for prices will not work) my $graph_ds = GT::Graphics::DataSource::Prices->new($q); # wrong my $marks_ds = GT::Graphics::DataSource::Close->new($q); # ok

189 GT::Graphics::Object::Mountain

190 GT::Graphics::Object::MountainBand

191 GT::Graphics::Object::Text

This graphical object displays a block of text.

192 GT::Graphics::Object::Positions

This graphical object displays all positions in a portfolio on a graph if the order date coincides with the graph time span.

the default buy orders color is Graphic::Positions::BuyColor which defaults to 62 % green intensity if not set in .gt/options

the default sell orders color is Graphic::Positions::SellColor which defaults to 62 % red intensity if not set in .gt/options

the marker color is always adjusted to be a transparent version of the color specified (and will likely alter the transparency attribute if it is already set)

if a 6th argument is supplied via Positions->new method and it is "true" (in perl not zero or something other than "") then a horizontal line will be drawn from the opening of the position (or start date) to the end date. price lines are not drawn for closed positions since the price points will likely differ. the line color will be based on the type of position, green for long, red for short.

193 SYNOPSIS

my $pf = GT::Portfolio->create_from_file("./my_portfolio);

$all_trades = GT::Graphics::Object::Positions->new($calc, $zone, \ $pf, $first, $last, "enable priceline");

NB: $calc, $zone, required by GT::Graphics::Object::<object>->new() which Positions.pm inherits. see Object.pm.

$graphic->add_object($all_trades);

```
  where $calc will yield the security symbol ($code) being processed
        $portfolio contains the portfolio data
        $first, $last are the dates of interest
```

194 EXAMPLES: script code

```
   (from ras hack of Samal Chandrans' portgraph.pl)

   my $pf = GT::Portfolio->create_from_file($pfname);
   my $all_trades = GT::Graphics::Object::Positions->new(
    $calc, $zone, $pf, $first, $last, "plotline");
   $graphic->add_object($all_trades);

   (from ras hack of backtest.pl)
```

```
my $positions = GT::Graphics::Object::Positions->new(
 $calc, $zone, $analysis->{'portfolio'}, $first, $last, "show priceline");
$positions->set_special_scale($scale_p);
$graphic->add_object($positions);
```

195 BUGS, NOTES, LIMITATIONS

no testing with widths other than the default width

i selected width of 24 because it was large enough for my old tired eyes to see on an otherwise cluttered graph but not so large as to obliterate the adjacent candle sticks

no testing with non-closed short positions. manage_portfolio doesn't seem to support them (or am i missing something) so i've been unable to easily mechanize an test/evaluation portfolio with them.

196 GT::Graphics::Object::Text

This graphical object displays a block of text.

$o->set_horizontal_align($value)

$o->set_vertical_align($value)

$o->set_x_position($value_in_pc)

$o->set_y_position($value_in_pc)

$o->set_font_size($fontsize)

$o->set_font_face($font_face)

197 GT::Graphics::Object::Trades

This graphical object displays trades as markers on a plot.
has opaque colors for lines and arrows

198 GT::Graphics::Object::VotingLine

This graphical object display buy and sell arrows.

The Voting Line contains the results of a System Manager, which is an entity interacting between a Portfolio Manager, a Trading System, an Order Factory, Trade Filters and Close Strategies.

199 GT::Graphics::Scale

A scale converts local data (numbers) into coordinate ready to be displayed.

Can use linear scale or logarithmic ones.

linear : X = a * x + b logarithmic : X = ln(x - b + 1) * a

$s->set_vertical_linear_mapping($y1, $y2, $ty1, $ty2)

Parameterize the scale to elaborate a mapping of [$y1,$y2] => [$ty1,$ty2] on the vertical axis.

$s->set_horizontal_linear_mapping($x1, $x2, $tx1, $tx2)

Parameterize the scale to elaborate a mapping of [$x1,$x2] => [$tx1,$tx2] on the horizontal axis.

$s->set_vertical_logarithmic_mapping($y1, $y2, $ty1, $ty2)

Parameterize the scale to elaborate a logarithmic mapping of [$y1,$y2] => [$ty1,$ty2] on the vertical axis.

$s->set_horizontal_logarithmic_mapping($x1, $x2, $tx1, $tx2)

Parameterize the scale to elaborate a logarithmic mapping of [$x1,$x2] => [$tx1,$tx2] on the horizontal axis.

($nx, $ny) = $s->convert_to_coordinate($x, $y)

Returns the coordinate of the ($x, $y) point with the scale modification applied.

$nx = $s->convert_to_x_coordinate($x)

Returns the X coordinate of the $X value with the scale modification applied.

$ny = $s->convert_to_y_coordinate($y)

Returns the Y coordinate of the $y value with the scale modification applied.

($x, $y) = $s->get_value_from_coordinate($nx, $ny)

Returns the value corresponding to the given coordinate.

$s->copy_horizontal_scale($other)

$s->copy_vertical_scale($other)

Copy the horizontal/vertical scale defined in the $other scale object.

200 GT::Graphics::Tools

This modules provides several helper functions that can be used in all modules.

It provides functions for managing labels on axes that can be imported with use GT::Graphics::Tools qw(:axis) :

build_axis_for_timeframe($prices, $timeframe, $put_label, $period)

Create the ticks for a time axis for the given $prices using the indicated $timeframe. If $put_label then labels we'll be put for each tick. If $period the label will be the name of the period, otherwise it will be the date of the first subperiod (usually a day).

build_axis_for_interval($min, $max, $many, $label)

Create the ticks between $min and $max for a numeric axis. If $many then many ticks (~20) will be created otherwise only a few (~5) will be created. If $label then the ticks will be labelled.

($min, $max) = union_range($min1, $max1, $min2, $max2)

Return the range resulting of the union of the two given ranges.

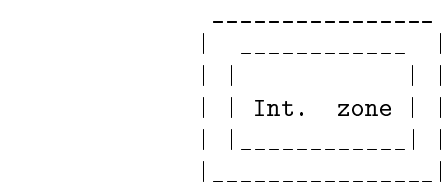
It provides functions to manage colors. They can be imported with use GT::Graphics::Tools qw(:color) :

get_color({$colorname|$color_code})

Return a color. You can ask it by its name ("blue", "light blue", ..) or by its RGB code "[125,164,198]".

201 GT::Graphics::Zone

A zone is a part of the graphic. It has an external size and an internal size. The internal size may be split in several sub zones.



A zone can also be considered like a ń drawable ż object and as such it implements the display(...) method. It will call the display methods for the axis, draw the border if needed and draw the title.

GT::Graphics::Zone->new($width, $height, $left, $right, $top, $bottom);

Creates a new zone with the given internal size. $left, $right, $top, $bottom is the free space to keep around the internal zone.

$z->add_subzone($zx, $zy, $zone)

Add $zone as a subzone of the current zone and place it at position ($zx,$zy). $zx and $zy are positive integers.

($ax, $ay) = $z->absolute_coordinate($x, $y)

Returns the absolute coordinate of the given point. ($x, $y) is the coordinate in the zone $z. ($ax, $ay) are the coordinate of the same point but in the root zone.

($x, $y) = $z->zone_coordinate($zx, $zy)

Returns the bottom left coordinate of the zone identified by ($zx, $zy). This function will only give good results once all zones have been created and linked together.

$z->update_size()

Update the size of the zone according to the size of the childs.

$z->get_subzone($zx, $zy)

Return the subzone indicated by the coordinate.

$z->set_axis_{left,right,top,bottom}($axis)

Put an axis on the indicated side.

$z->get_axis_{left,right,top,bottom}()

Get the axis of the indicated side.

$z->set_title_{left,right,top,bottom}($title)

Put a title on the indicated side.

$z->get_title_{left,right,top,bottom}()

Get the title of the indicated side.

$z->set_title_font_name($name)

$z->set_title_font_size($size)

$z->set_title_font_color($size)

Those 3 methods are used to change the font for the title.

$z->set_default_scale($scale)

Use $scale as the default scale for this zone.

$z->set_border_width($width)

Add a border of the given width around the zone. It will not sharp the zone but extend/decrease the space around it.

$z->set_border_color([$R,$G,$B])

Change the color of the border.

$z->includes_point($x, $y, [$extented])

Returns true if the point is within the zone, false otherwise. If $extended is true, the border of the zone is considered as part of the zone.

$z->display($driver, $graphic)

Display the zone with its axis, its borders and its titles.

202 SIMPLE FUNCTIONS

$z->width() && $z->height()

$z->external_width() && $z->external_height()

($x, $y) = $z->get_position()

$z->get_parent()

203 INTERNAL FUNCTIONS

$z->set_parent($parent)

$z->set_position($zx, $zy)

204 GT::Indicators

Provides some functions that will be used by all indicators.

DESCRIPTION

GENERIC EXPORTED FUNCTIONS

`build_object_name($encoded, [ @args ], $key)`

Generate the name of an indicator based on its "encoded" name.

FUNCTIONS TO RETRIEVE USUAL PRICES

$GET_OPEN, $GET_FIRST, $GET_HIGH, $GET_LOW, $GET_LAST, $GET_CLOSE and $GET_VOLUME are functions references that can be passed as arguments to some indicators where input functions are expected. They are automatically exported when doing "use GT::Indicators;".

For example, the indicator AMA (Arithmetic Moving Average) can be used to calculate the average of anything (other indicators or prices). You can provide those functions if you want to calculate an average of some prices (or quotations or volumes).

MANAGE A REPOSITORY OF INDICATORS

```
GT::Indicators::get_registered_object($name);
GT::Indicators::register_object($name, $object);
GT::Indicators::get_or_register_object($name, $object);
GT::Indicators::manage_object(\@NAMES, $object, $class, $args, $key);
```

DEFAULT FUNCTIONS FOR INDICATORS

`GT::Indicators::<indicator>->new($args, $key, $func) depreciated`

$func has been depreciated and is no longer supported. if $func is passed as an argument it will result in a runtime error.

`GT::Indicators::<indicator>->new($args, $key)`

Create a new <indicator> with the given arguments. $key is optional, useful for indicators which can use non-usual input streams.

`$indic->calculate_all($calc)`

calculate_all will calculate all the values of the indicators for all possibles days.

`$indic->calculate_interval($calc, $first, $last)`

Provide a default non-optimized version of calculate_interval that calls calculate once for each day.

Real indicators are encouraged to override this function to provide an optimized version of the calculation algorithm by possibly reusing the result of previous days.

`$indic->initialize()`

Default method that does nothing. note this method is called by GT::Registry::manage_object if the object is new and needs initialization.

`$indic->get_name`

`$indic->get_name($n)`

Get the name of the indicator. If the indicator returns several values, you can get the name corresponding to any value, you just have to precise in the parameters the index of the value that you're interested in.

`$indic->get_nb_values`

Return the number of different values produced by this indicator that are available for use.

205 GT::Indicators::ADL

Accumulation/Distribution line

DESCRIPTION

Overview

The Accumulation/Distribution Line was developed by Marc Chaikin to assess the cumulative flow of money into and out of a security.

Calculation

The ADL is the cumulative sum of (((Close - Low) - (High - Close)) / (High - Low)) * Volume

Links

http://www.stockcharts.com/education/What/IndicatorAnalysis/indic_AccumDistLine.html
http://www.equis.com/free/taaz/accumdistr.html

206 GT::Indicators::ADX

ADX

DESCRIPTION

volatility index

Overview

Calculation

Examples

GT::Indicators::ADX->new() GT::Indicators::ADX->new([20])

Validation

This indicators is validated by the values from comdirect.de. The stock used was the DAX (data from yahoo) at the 04.06.2003:

```
ADX[14]             [2003-06-04] = 19.9961 (comdirect=20.00)
+DMI[14]            [2003-06-04] = 28.9251 (comdirect=28.93)
-DMI[14]            [2003-06-04] = 21.1723 (comdirect=21.17)
DMI[14]             [2003-06-04] = 15.4754 (comdirect=15.48)
```

Links

207 GT::Indicators::ADXR

Overview

Calculation

ADXR = (Today's ADX + ADX #1 days ago) / 2

Examples

GT::Indicators::ADXR->new() GT::Indicators::ADXR->new([20])

Links

GT::Indicators::ADXR::calculate($calc, $day)

GT::Indicators::AROON

Overview

Developed by Tushar Chande in 1995, the Aroon is an indicator system that can be used to determine whether a stock is trending or not and how strong the trend is. "Aroon" means "Dawn's Early Light" in Sanskrit and Chande choose that name for this indicator since it is designed to reveal the beginning of a new trend.

The Aroon indicator consists of two lines, Aroon(up) and Aroon(down). The Aroon Oscillator is a single line that is defined as the difference between Aroon(up) and Aroon(down). All three take a single parameter which is the number of time periods to use in the calculation. Since Aroon(up) and Aroon(down) both oscillate between 0 and +100, the Aroon Oscillator ranges from -100 to +100 with zero serving as the crossover line.

Calculation

Aroon(up) for a given time period is calculated by determining how much time (on a percentage basis) elapsed between the start of the time period and the point at which the highest closing price during that time period occurred. When the stock is setting new highs for the time period, Aroon(up) will be 100. If the stock has moved lower every day during the time period, Aroon(up) will be zero. Aroon(down) is calculated in just the opposite manner, looking for new lows instead of new highs.

Examples

GT::Indicators::AROON->new() GT::Indicators::AROON->new([20])

Validation

This indicators is validated by the values from comdirect.de. The stock used was the DAX (data from yahoo) at the 04.06.2003:

AroonUp[25] [2003-06-04] = 100.0000 (comdirect: 100.0) AroonDown[25] [2003-06-04] = 76.0000 (comdirect: 76.0) AroonOsc[25] [2003-06-04] = 24.0000 (comdirect: 24.0)

Links

http://stockcharts.com/education/resources/glossary/aroon.html http://www.paritech.com/educatio
http://www.geocities.com/WallStreet/Floor/1035/aroon.htm

GT::Indicators::AROON::calculate($calc, $day)

208 GT::Indicators::AT3

Overview

AT3 is an excellent data-fitting technique by Tim Tillson (cf. "Smoothing Techniques For More Accurate Signals" in Technical Analysis of Stocks and Commodities - January 1998) that works with RSquare.

Calculation

N is the Exponential Moving Average Period a is the amplification percentage of the filter's response to price movement (0 < a < 1)

a = RSquare()

e1 = N-days EMA of Close Prices e2 = N-days EMA of e1 e3 = N-days EMA of e2 e4 = N-days EMA of e3 e5 = N-days EMA of e4 e6 = N-days EMA of e5

c1 = (-a)^3 c2 = 3 * a^2 c3 = - 6 * a^2 - 3 * a - 3 * a^3 c4 = 1 + 3 * a + a^3 + 3 * a^2

T3 = c1 * e6 + c2 * e5 + c3 * e4 + c4 * e3

GT::Indicators::T3::calculate($calc, $day, $args, $key, $data)

209 GT::Indicators::ATR

Overview

The Average True Index (ATR) is a measure of volatility. High ATR values often occur at market bottom following a 'panic' sell-off. Low ATR values are often found during extended sideways periods, such as those found at tops and after consolidation periods.

Calculation

The Average True Index is a moving average of the True Ranges.

Parameters

The standard ATR works with a fourteen-day parameter : n = 14

Example

GT::Indicators::ATR->new() GT::Indicators::ATR->new([12])

Note

The Average True Range can be interpreted using the same techniques that are used with other volatility indicators.

Validation

This indicators is validated by the values from comdirect.de. The stock used was the DAX (data from yahoo) at the 04.06.2003:

ATR[14] [2003-06-04] = 79.2343 (comdirect=79.23)

Links

http://www.equis.com/free/taaz/avertrurang.html http://stockcharts.com/education/What/Indicat
http://www.finance-net.com/apprendre/techniques/atr.phtml

GT::Indicators::ATR::calculate($calc, $day)

210 GT::Indicators::BBO

%B - the Bollinger Band Oscillator

DESCRIPTION

Bollinger Calculated this oscillator. The %B falls below 0 if the Price crosses the lower Band. It is set to > 1 if the price raises above the upper band.

Parameters

The parameters are identical with those of the BOL-Indicator.

GT::Indicators::BOL

Bollinger Bands are similar to moving average envelopes. The difference between Bollinger Bands and envelopes is envelopes are plotted at a fixed percentage above and below a moving average, whereas Bollinger Bands are plotted at standard deviation levels above and below a moving average.

The standard Bolling Bands (BOL 20-2) can be called like that : GT::Indicators::BOL->new()

If you need a non standard BOL : GT::Indicators::BOL->new([25, 2.5])

GT::Indicators::BOL::calculate($calc, $day)

GT::Indicators::BPCorrelation (Bravais-Pearson Correlation Coefficient)

This function will calculate the Bravais-Pearson Correlation Coefficient. Correlation analysis measures the relationship between two items and shows if changes in one item will result in changes in the other item.

this indicator requires three arguments, and provides no default values for any of them.

the first argument is the number of intervals in the period, it can be a constant or a data series. the period is used as the number of data values used in each computation

the second and third arguments must be functions (e.g. data series or data objects?)

the indicator will validate that the arguments are provided and are of the correct type.

examples (display_indicator)

```
%   display_indicator.pl I:BPCorrelation 13000 \
'20 {I:Prices OPEN} {I:Prices CLOSE}'

%   display_indicator.pl I:BPCorrelation 13000 \
'14 {I:G:Cum 1} {I:Prices CLOSE}'
```

GT::Indicators::Correlation::calculate($calc, $day)

211 GT::Indicators::CCI

Overview

Developed by Donald Lambert, the Commodity Channel Index (CCI) was designed to identify cyclical turns in commodities. The assumption behind the indicator is that commodities (or stocks or bonds) move in cycles, with highs and lows coming at periodic intervals.

Calculation

There are 4 steps involved in the calculation of the CCI : 1. Calculate the last period's Typical Price (TP) = (H+L+C)/3 where H = high, L = low, and C = close. 2. Calculate the 20-period Simple Moving Average of the Typical Price (SMATP). 3. Calculate the Mean Deviation. First, calculate the absolute value of the difference between the last period's SMATP and the typical price for each of the past 20 periods. Add all of these absolute values together and divide by 20 to find the Mean Deviation. 4. The final step is to apply the Typical Price (TP), the Simple Moving Average of the Typical Price (SMATP), the Mean Deviation and a Constant (.015) to the following formula :

CCI = ((Typical Price - Simple Moving Average of the Typical Price) / (0.015 * Mean Deviation))

Parameters

Lambert recommended using 1/3 of a complete cycle (low to low or high to high) as a time frame for the CCI. Note that the determination of the cycle's length is independent of the CCI. If the cycle runs 60 days (a low about every 60 days), then a 20-day CCI would be recommended.

Example

GT::Indicators::CCI->new() GT::Indicators::CCI->new([25])

Note

Traders and investors use the CCI to help identify price reversals, price extremes and trend strength. As with most indicators, the CCI should be used in conjunction with other aspects of technical analysis. CCI fits into the momentum category of oscillators.

Validation

This Indicator was validated by the data available from comdirect.de: The DAX at 04.06.2003 (data from yahoo.com) had a CCI of 158.71 This is consistent with this indicator: 158.7057

Links

http://www.equis.com/free/taaz/cci.html http://www.stockcharts.com/education/What/IndicatorA
http://www.finance-net.com/apprendre/techniques/cci.phtml

GT::Indicators::CCI::calculate($calc, $day)

GT::Indicators::CHAIKIN

Overview

The Chaikin Oscillator is a moving average oscillator based on the Accumulation/Distribution indicator (ADL.pm).

Calculation

The formula is the difference between the 3-day exponential moving average and the 10-day exponential moving average of the Accumulation/Distribution Line.

Examples

GT::Indicators::CHAIKIN->new() GT::Indicators::CHAIKIN->new([3, 10])

Links

http://www.stockcharts.com/education/What/IndicatorAnalysis/indic_ChaikinOscillator.html
http://www.equis.com/free/taaz/chaikinosc.html

GT::Indicators::CHAIKIN::calculate($calc, $day)

212 GT::Indicators::CMO

Chande Moment Oscillator

DESCRIPTION

The CMO indicator was developed by Trushar Chande and presented 1994 in the book "The New Technical Trader". It can be used as an oscillator (CMO > 50 => overbought, CMO < -50 => oversold) or as a trend indicator (the higher/lower the CMO, the stronger the trend)

```
CMO = 100 * (SumUp-SumDown) / (SumUp+SumDown)
```

Parameters

Period (default 10)

This argument is used to calculate the SumUp and SumDown.

Creation

```
GT::Indicators::CMO->new()
```

Links

213 GT::Indicators::CNDL

Overview

The CandelCode (CNDL) indicator is based on the article "Coding Candelsticks" published in Technical Analysis of Stocks and Commodities (November 1999) by Viktor Likhovidov.

Calculation

Parameters

Two parameters are used to initialize the Bollinger Bands necessary to calculate all required thresholds; the standard deviation number is set to 0.5 with a period of 55.

Examples

GT::Indicators::CNDL->new() GT::Indicators::CNDL->new([55, 0.5])

Appendix

If you need to find quickly the candel code of a specific pattern, here is a conversion table : /GT/Docs/CandelsticksCodes

Links

http://www.traders.com/Documentation/FEEDbk_docs/Archive/012000/TradersTips/TradersTips

214 GT::Indicators::ChaikinsVola

Chaikins Volatility

DESCRIPTION

This is calculated as the Rate of change of an Moving Average of the difference between High and Low.

Parameters

Period 1

The Period used for the ROC.

Period 2

The Period used for the MA.

High

Low

GT::Indicators::Chandelier

The Chandelier Exit is described in Dr. Alexander Elder's Book "Come into my Trading Room" and provides stops for closing long or short positions. It was originally conceived by Chuck LeBeau.

It accepts the number of bars to use for the calculation and a coefficient as parameters with 22 and 3 being the defaults that are also used in the examples in the book.

ChanUp should be used for long positions, ChanDn for short positions.

GT::Indicators::Chandelier::calculate($calc, $day)

215 GT::Indicators::ForwardKPercent

Probability to make a profitable trade

DESCRIPTION

This indicator calculates the K%-value:

```
               CLOSE - MIN(3, Low)
K% (3) = ------------------------------
           MAX(3, HIGH) - MIN(3,Low)
```

Be aware that this indicator "knows" the future so don”t use it for your trading strategies :)

PARAMETERS

Number of days

The number of days the indicator looks in the future

216 GT::Indicators::MeanPossiblePerformance

DESCRIPTION

PARAMETERS

Number of days

The number of days the indicator looks in the future

Data

This is the data to use as input. If you don't specify anything, the close price will be used by default.

217 GT::Indicators::DMI -

DESCRIPTION

EXAMPLES

218 GT::Indicators::DSS

Double Smoothed Stochastic (William Blau).

DESCRIPTION

From http://www.wealth-lab.com/cgi-bin/WealthLab.DLL/getdoc?id=128:

DSS applies 2 smoothing EMAs of different lengths to a Stochastic Oscillator. DSS ranges from 0 to 100, like the standard Stochastic Oscillator. The same rules of interpretation that you use for Stochastics can be applied to DSS, although DSS offers a much smoother curve than Stochastics.

From http://www.tradesignalonline.com/Lexicon/Default.aspx?name=DSS%3a+Double+Smoothed

Calculation of the DSS indicator is similar to stochastics. The numerator: first the difference between the current close and the period low is formed, and this is then exponentially smoothed twice. The denominator is formed in the same way, but here the difference is calculated from the period high minus the period low. Numerator and denominator yield the quotient, and this value is multiplied by 100.

Calculation

As can be seen from above, there is some disagreement on the calculation process. We follow the latter and calculate DSS-BLAU as follows:

DSS-BLAU[p1,p2,p3] = EMA[p3, EMA[p2, Close - LowestLow[p1]] 100 * ———————————————————— - EMA[p3, EMA[p2, HighestHigh[p1]-LowestLow[p1]]

The handling and the calculations of signals is similar to the Stochastic-Indictor.

Parameters

Period 1 (default 5)

The period over which to consider highest highs and lowest lows.

Period 2 (default 7)

The period of the first smoothing

Period 3 (default 3)

The period of the second smoothing

High, Low, and Close of Source

The source from which the indicators is calculated.

219 GT::Indicators::EMA

Exponential Moving Average

DESCRIPTION

An exponential moving average gives more importance to recent prices ...

Parameters

Period (default 20)

The first argument is the period used to calculate the average.

Other data input

The second argument is optional. It can be used to specify an other stream of input data for the average instead of the close prices. This is usually an indicator (detailed via {I:MyIndic <param>}).

Start data input

The third argument is optional. It can be used to specify the stream of input data to compute the starting point of the moving average. The default is computed by the SMA of the given period.

If a very long period is required, it may be advisable to set this to {I:PRICES CLOSE} (or whatever data stream is used as the input for the EMA) to avoid excessive history data being required just to compute the starting value. Using the first value of the input series does not result in a large error and requires no dependencies.

Calculation

alpha = 2 / (N + 1) EMA[n] = EMA[n-1] + alpha * (INPUT - EMA[n-1])

In TA, the first value is often constructed as SMA(N).

Note: One criticism could be that the EMA is calculated starting from the designated period. But actually the EMA goes all the way back to the beginning of the available data. But in all tools I checked they start computation from the loaded data on.

Creation

```
GT::Indicators::EMA->new()
GT::Indicators::EMA->new([20])
```

If you need a 30 days EMA of the opening prices you can write one of those lines :

```
GT::Indicators::EMA->new([30, "{I:Prices OPEN}"])
```

A 10 days EMA of the RSI could be created with :

```
GT::Indicators::EMA->new([10, "{I:RSI}"])
```

GT::Indicators::ENV

An envelope is composed of two moving averages. One moving average is shifted upward and the second moving average is shifted downward. Envelopes define the upper and the lower boundaries of a security's normal trading range.

The standard envelope (ENV 25-6) can be called like that : GT::Indicators::ENV->new()

If you need a non standard ENV : GT::Indicators::ENV->new([21, 5])

GT::Indicators::ENV::calculate($calc, $day)

GT::Indicators::EPMA

Overview

The Endpoint Moving Average (EPMA) is focus on divergences between the original time series and the transposed time series. They may be used in forecasting applications or as additional inputs for neural analyses.

Calculation

EPMA(n) = [2 / (n * (n + 1))] * Sum of (((3 * i) - n - 1) * Close(i)) from i = 1 to i = n

Examples

GT::Indicators::EPMA->new() GT::Indicators::EPMA->new([50]) GT::Indicators::EPMA->new([30], "OPEN", $GET_OPEN)

Links

http://www.ivorix.com/en/products/tech/smooth/epma.html

220 GT::Indicators::EVWMA (Elastic Volume Weighted Moving Average)

Overview

The Elastic Volume Weighted Moving Average (eVWMA) differs from usual average in that :

- It does not refer to any underlying averaging time period (for example, 20 days, 50 days, 200 days). Instead, eVWMA uses share volume to define the period of the averaging.

- It incorporates information about volume (and possibly time) in a natural and logical way

- It can be derived from, and seen as an approximation to, a statistical measure and thus has a solid mathematical justification.

=====>>>>> SIGNIFICANT USAGE ISSUE <<<<<======

to use I:EVWMA one must have a database containing the number of shares floating for each security being analyzed. this shares_float database is only searched for in an xml file at /bourse/metainfo/"$code".xml

currently the beancounter database doesn't store this security attribute, nor does beancounter fetch this value in the course of a normal daily update.

yahoo does provide the attribute via 'f6', but how one might create the required xml file based database is not described.

Calculation

eVWMA(0) = Today's Close eVWMA(i) = ((Number of shares floating - Today's Volume) \ * eVWMA(i-1) + Today's Volume * Today's Close) \ / Number of shares floating

Example

GT::Indicators::EVWMA->new()

Links

http://www.christian-fries.de/evwma/ http://www.linnsoft.com/tour/techind/evwma.htm

GT::Indicators::EVWMA::calculate($calc, $day)

221 GT::Indicators::ElderRay

GT::Indicators::ElderRay->new([13])

INFORMATION

It has been invented by Alexander Elder, and it is explained in his book "Trading for a living" ("Vivre du trading" in french).

222 GT::Indicators::FISH

Overview Infos cited from chart manual at http://www.geocities.com/user42_kevin/chart/index.html

The fisher transform indicator by John Ehlers is a range oscillator showing where today's price is within the past N-days highest and lowest, with some smoothing is used plus what's known in mathematics as a fisher transform. This is similar to Stochastics and Williams %R but the transformation stretches values near the high and low, helping to highlight extremes. =head2 Calculation

The calculation is as follows. The prices used are the midpoint between the day's high and low (as in most of Ehlers' indicators). Today's price is located within the highest and lowest of those prices from the past N days, scaled to -1 for the low and 1 for the high.

```
price = (high + low) / 2

            price - Ndaylow
raw = 2 * ------------------ - 1
          Ndayhigh - Ndaylow
```

This raw position is smoothed by a 5-day EMA and a log form which is the mathematical fisher transform, before a final further 3-day EMA smoothing.

```
smoothed = EMA[5] of raw

                                1 + smoothed
fisher = EMA[3] of 0.5 * log ------------
                                1 - smoothed
```

Parameters

The standard Fisher-Transform works with a 10-days parameter : n = 10

Links http://mesasoftware.com/technicalpapers.htm http://www.geocities.

Creation

```
GT::Indicators::FISH->new()
GT::Indicators::FISH->new([20])
```

If you need a 30 days Fisher Transform of the opening prices you can write one of those lines :

```
GT::Indicators::FISH->new([30, "{I:Prices OPEN}"])
```

A 10 days Fisher Transform of the RSI could be created with :

```
GT::Indicators::FISH->new([10, "{I:RSI}"])
```

GT::Indicators::SMI::calculate($calc, $day)

223 GT::Indicators::FRAMA

FRactal Adaprive Moving Average

DESCRIPTION

An exponential moving average gives more importance to recent prices ... similarly a Frama uses a variable (adaptive) alpha

Parameters

Period (default 20)

The first argument is the period used to calculed the average.

Other data input

The second argument is optional. It can be used to specify an other stream of input data for the average instead of the close prices. This is usually an indicator (detailed via {I:MyIndic <param>}).

Calculation The alpha is calculated following http://www.mesasoftware.com/technicalpa see the self explaining code below :D when knowing the alpha FRAMA is FRAMA[n] = FRAMA[n-1] + alpha * (INPUT - FRAMA[n-1])

In TA, the first value is often constructed as SMA(N).

Note: One criticism could be that the EMA is calculated starting from the designated period. But actually the EMA goes all the way back to the beginning of the available data. But in all tools I checked they start computation from the loaded data on.

Creation

```
GT::Indicators::FRAMA->new()
GT::Indicators::FRAMA->new([20])
```

If you need a 30 days FRAMA of the opening prices you can write one of those lines :

```
GT::Indicators::FRAMA->new([30, "{I:Prices OPEN}"])
```

A 10 days EMA of the RSI could be created with :

```
GT::Indicators::FRAMA->new([10, "{I:RSI}"])

note!!! The number of days has to be EVEN!!
```

224 GT::Indicators::ForceIndex

GT::Indicators::ForceIndex->new()

INFORMATION

The force index itself varies too much. To be of any use, you'd better use a 2 days exponential moving average of it.

It has been invented by Alexander Elder, and it is explained in his book "Trading for a living" ("Vivre du trading" in french).

225 GT::Indicators::FromTimeframe

Get data from an other timeframe

DESCRIPTION

If you need data from an other timeframe (e.g. to determine the trend on weekly basis), you can use this indicator.

PARAMETERS

Data

A normal indicators-/data-object.

Timeframe

The timeframe you want to get

Days

The number of periods you want to go back in the requested timeframe.

226 GT::Indicators::GAPO

Overview

The Gopalakarishnan Range Index (GAPO) characterizes the price behavior of markets. Although GAPO doesn't generate buy or sell signals, it does help identify the random behavior of price activity. A higher value indicates a more erratic market; a lower value indicates consistent price movement.

Calculation

GAPO = (Log(Highest High (n) - Lowest Low (n))) / Log (n)

Parameters

The standard GAPO index works with a five-day parameter : n = 5

Example

GT::Indicators::GAPO->new() GT::Indicators::GAPO->new([6])

Advice/Idea

I think that the best way to use the results of this indicator is to look after the average or the moving average of the results, in order to have smooth data.

GT::Indicators::GAPO::calculate($calc, $day)

227 GT::Indicators::GMEAN

Overview

The geometric mean indicator calculates the geometic mean of each days high and low. While the arithmetic mean has an equal absolute distance to high and low, the geometric mean has an equal relative distance to high and low.

arithmetic mean: high - mean = mean - low geometric mean: high / gmean = gmean / low

Calculation

gmean = (high * low)^(1/2)

Links

228 GT::Indicators::Generic::Abs

Return the absolute value of its 1st parameter

DESCRIPTION

NAME

GT::Indicators::Generic::ByName - Alias to another indicator

DESCRIPTION

Sometimes, during the computation of an indicator, one needs to reference the current value of a series that is being computed by this indicator. If the current indicator is used explicity, an infinite recursion arises due to the dependency mechanism.

This indicator resolves the recursion. This indicator is nothing more than an alias of a series calculated by an indicator. Just give as first parameter the name of the value to use.

This indicator is used when, during the calculation of an indicator, an intermediate series wants to leverage another intermediate series or an output value.

For example,

```
$self->{'sma1'} = GT::Indicators::SMA->new([
                  $self->{'args'}->get_arg_names(1),
                  $self->{'args'}->get_arg_names(2) ]);
$self->{'sma2'} = GT::Indicators::SMA->new([
                  $self->{'args'}->get_arg_names(1),
                  "{I:Generic:ByName "
                . $self->{'sma1'}->get_name . "}" ]);
```

The first statement defines an intermediate series which smoothes the second parameter. The second statement takes that series and applies smoothing again. Similarly, the following applies smoothing to the first output value.

```
$self->{'sma3'} = GT::Indicators::SMA->new([
                  $self->{'args'}->get_arg_names(1),
                  "{I:Generic:ByName " . $self->get_name(0) . "}" ]);
```

Care has to be taken that I:Generic:ByName is in fact given the name of an indicator, lest that series will not be found. Note that if the series is an indicator, the name of the series is the name of the selected return value. The get_name method will always retrieve the name of a series.

Remember that the parseable syntax does not yield a name.

229 GT::Indicators::Generic::Container

Fake indicator which does nothing but acts as a data container

DESCRIPTION

This indicator can be used to store arbitrary series of data in particular temporary values used during calculations of complicated indicators. If you need to calculate the SMA of an expression, you can store the result of that expression in that indicator.

All arguments passed serves only one purpose : differentiate the various series of data stored. Care should be taken to ensure the uniqueness of the indicator name, if there is a chance that several instances of this indicator are active at the same time (e.g., when used as the long and short signals of a system).

230 GT::Indicators::Generic::Cum

Overview

This function keeps a running total of its input. Each period is calculated, it adds the current value of the input to the previous total. For example, {I:Generic:Cum 1} will keep adding 1 for each period of time loaded. In effect, this counts how many records are currently loaded.

GT::Indicators::Generic::Cum::calculate($calc, $day)

231 GT::Indicators::Generic::Diff

Difference between two days

DESCRIPTION

Calculates the difference between the actual value and the value n days ago.

{I:RSI} 14

232 GT::Indicators::Generic::Divide

Calculates Param1 / Param2

DESCRIPTION

This Indicator is calculation an division of several parameters.

Overview

Calculation

Examples

Links

233 GT::Indicators::Generic::Eval

Evaluate the given expression

DESCRIPTION

This indicator evaluates the expression given via its argument. Any indicator is replaced by its current value.

Example of accepted argument list :

int({I:RSI})

1+1

{I:Generic:SignalLength {Signals:Prices:Advance 5}}+1

100 - {I:RSI 10} * 2

The argument list is treated via perl's eval function so any standard perl code may be accepted ... but it's only meant for simple single expression.

234 GT::Indicators::Generic::If

Return a value or another depending on a signal

DESCRIPTION

This indicator takes three parameters. First a signal followed by two indicators. if the signal is true it returns the value of the first indicator, otherwise it returns the value of the second indicator.

{S:Prices:Advance} {I:Generic:MaxInPeriod 5} {I:Generic:MinInPeriod 5}

{S:Generic:CrossOverUp {I:RSI} 80} {I:SAR} {I:Generic:MaxInPeriod 10}

235 GT::Indicators::Generic::Max

Return the max of all parameters

DESCRIPTION

This indicator returns the biggest value of all its parameters.

236 GT::Indicators::Generic::MaxInPeriod

Calculate a maximum

DESCRIPTION

This indicator calculates the maximum of any serie of data in the last XX days or since a given date.

PARAMETERS

Number of days / date

You can specify either a number or a date. In the first case the maximum will be calculated with the last <number> days. In the second case it will be the maximum since the given date.

Data

This is the data to use as input. If you don't specify anything, the closing price will be used by default.

Example of accepted argument list :

1. {I:RSI}
2. -01-03
3.
4. -04-05 {I:Prices HIGH}
5. "2005-04-05 14:30:00" {I:Prices HIGH}

237 GT::Indicators::Generic::Min

Return the minimum of all parameters

DESCRIPTION

This indicator returns the smallest value of all its parameters.

238 GT::Indicators::Generic::MinInPeriod

Calculate a mimimum

DESCRIPTION

This indicator calculates the minimum of any serie of data in the last XX days or since a given date.

PARAMETERS

Number of days / date

> You can specify either a number or a date. In the first case the minimum will be calculated with the last <number> days. In the second case it will be the minimum since the given date.

Data

> This is the data to use as input. If you don't specify anything, the closing price will be used by default.

Example of accepted argument list :

1. {I:RSI}
2. -01-03
3.
4. -04-05 {I:Prices LOW}
5. "2005-04-05 14:30:00" {I:Prices LOW}

239 GT::Indicators::Generic::PeriodAgo

Return data from some periods ago

DESCRIPTION

This function returns N-Period Ago value of the indicator given on arguments. Without indicator, the close price is used.

EXAMPLES

The high of 3 days ago :

```
I:Generic:PeriodAgo 3 {I:Prices HIGH}
```

240 GT::Indicators::Generic::SignalLength

Length of any signal

DESCRIPTION

This indicator returns the number of consecutive periods where the signal in parameter has been detected.

241 GT::Indicators::Generic::Speed

Speed of a indicator

DESCRIPTION

This function returns today's value minus yesterday's value of the indicator given on arguments. Without indicator, the close price is used.

242 GT::Indicators::Generic::Sum

Calculation of the Sum of the last n days

DESCRIPTION

Calculates the Sum of the last n days.

Overview

Calculation

Examples

GT::Indicators::Generic::SumUp->new()

Links

243 GT::Indicators::Generic::SumDownDiffs

Calculation of the Sum of the last n days when the price goes down

DESCRIPTION

Calculates the Sum of the difference of the last n days when the price goes down.

Overview

Calculation

Examples

GT::Indicators::Generic::SumDownDiffs->new()

Links

244 GT::Indicators::Generic::SumIf

Return a sum depending on a signal

DESCRIPTION

This indicator takes three parameters. First a signal followed by a period and an indicator. It returns the sum of the days where the Signal is true.

{S:Generic:CrossOverUp {I:RSI} 80} 14 {I:SAR}

245 GT::Indicators::Generic::SumUpDiffs

Calculation of the Sum of the differences of the last n days when the price goes up

DESCRIPTION

Calculation of the Sum of the differences of the last n days when the price goes up.

Overview

Calculation

Examples

GT::Indicators::Generic::SumUpDiffs->new()

Links

246 GT::Indicators::HilbertPeriod

Overview

Calculation

Examples

Links

TASC November 2000 - page 108

GT::Indicators::HilbertPeriod::calculate($calc, $day)

247 GT::Indicators::HilbertSine

Overview

Calculation

Examples

Links

TASC May 2000 - page 27

248 GT::Indicators::IFISH

Overview Remember The Fisher Transform

```
                        1 + x
    fisher = 0.5 log ------------
                        1 - x
Its inverse is

               exp(2*fisher)-1
    ifisher = ----------------
               exp(2*fisher)+1
```

The input values should lie in the Interval [-5,5] so they have to be adjusted to this interval. So an Oscillator is moved,scaled,smoothed and then inverted. Ehlers ist using a WMA for the smoothing I will use an EMA.

Paramters

The User has to input valid scaling parameters. for the RSI they are 0.1 and 50 so 0.1(RSI-50) varies between -5 and 5. 1. smoothing period 2. scaling value 3. midpoint adjustment

Links http://mesasoftware.com/technicalpapers.htm

Creation

```
GT::Indicators::IFISH->new()
```

GT::Indicators::SMI::calculate($calc, $day)

249 GT::Indicators::InstantTrendLine

Overview

Calculation

Examples

Links

TASC May 2000 - page 22

GT::Indicators::InstantTrendLine::calculate($calc, $day)

250 GT::Indicators::Interquartil

Interquartil-Distance

DESCRIPTION

The Interquartil-distance; which is the position at which you can divide the data by x% on the left side and (100-x)% on the right side.

Parameters

Percentage
: Percentage of the IQD (median = 50%)

Period (default 50)
: The first argument is the period used to calculed the average.

Other data input
: The Data for the calculation.

Creation

To create a kind of dynamic borders for the RSI try:

Indicators::Interquartil(90,50,{I:RSI}) Indicators::Interquartil(10,50,{I:RSI})

251 GT::Indicators::KAMA

Perry Kaufmanns Adaptive Moving Average

DESCRIPTION

This Indicator was developed by Perry Kaufmann and presented in the book "Trading Systems and Methods, 3rd Ed." in 1998. The KAMA is automatically adapted to the volatility of the market.

The interpretation is similar to the classic SMA.

Period (default 21)

The first argument is the period used to calculed the average.

Fastest period (default 30)

The fastest period to be considered.

Slowest period (default 2)

The slowest period to be considered.

Creation

```
GT::Indicators::KAMA->new()
GT::Indicators::KAMA->new([10])
```

252 GT::Indicators::Keltner

Keltner Channel

DESCRIPTION

Parameters

Period 1 (default 9)

Period on which the indicator has to be calculated

Constant (default 2)

A constant factor with wich the Average True range is multiplied.

Creation

Link

253 GT::Indicators::KirshenbaumBands

Overview

Kirshenbaum Bands are similar to Bollinger Bands, in that they measure market volatility. However, rather than use Standard Deviation of a moving average for band with, they use Standard Error of linear regression lines of the Close. This has the effect of measuring volatility around the current trend, instead of measuring volatility for changes in trend.

Author

Paul Kirshenbaum, a money manager and mathematician with PhD in economics from NYU, submitted this rather unique trading band which is "de-trended".

GT::Indicators::KirshenbaumBands::calculate($calc, $day)

254 GT::Indicators::LinearRegression

This function will calculate an L-Period linear regression line. Note that the term "linear regression" is the same as a "least squares" or "best fit" line.

The linear regression value is "a * i + b". i is the day number. a and b are also provided by the indicator if you want to calculate the value of the linear regression for other days.

Parameters

Takes 2 or 3 parameters. The first is the period over which the regression is calculated. The following parameters indicate the series that are being compared. If there is only a second parameter, this parameter forms the dependent variables, while the numerical sequence is the independent parameter. If there are both a second and a third parameter, the former is the independent and the latter the dependent parameter.

GT::Indicators::MACD

The standard Moving Average Convergence Divergence (MACD 12-26-9) can be called like that : GT::Indicators::MACD->new()

If you need a non standard MACD : GT::Indicators::MACD->new([20, 50, 15])

GT::Indicators::MACD::calculate($calc, $day)

GT::Indicators::MACD::calculate_interval($calc, $first, $last)

255 GT::Indicators::MAMA

Mesa Adaptive Moving Average

DESCRIPTION

please see Ehlers work

Parameters

Period (default 20)

Creation

```
GT::Indicators::MAMA->new()
```

256 GT::Indicators::MASS

Overview

The Mass Index was designed to identify trend reversals by measuring the narrowing and widening of the range between the high and low prices. As this range widens, the Mass Index increases; as the range narrows the Mass Index decreases.

The Mass Index was developed by Donald Dorsey.

Calculation

Mass Index = A-day sum of the ratio between the B-day EMA of (High - Low) and the B-day EMA of the B-day EMA of (High - Low)

Parameters

The standard Mass Index is calculated with : A = 25 and B = 9

Examples

GT::Indicators::MASS->new() GT::Indicators::MASS->new([30, 14])

Links

http://www.equis.com/free/taaz/massindex.html http://www.charthelp.com/reports/c21.htm

GT::Indicators::MASS::calculate($calc, $day)

257 GT::Indicators::MEAN

Overview

The mean indicator is simply an average of each days high and low.

Calculation

mean =(high+low)/2

Links

258 GT::Indicators::MFI

Overview

The Money Flow Index (MFI) is a momentum indicator that measures the strength of money flowing in and out of a security. It is related to the Relative Strength Index (RSI), but where the RSI only incorporates prices, the Money Flow Index accounts for volume.

Interpretation

Look for divergence between the indicator and the price action. If the price trends higher and the MFI trends lower (or vice versa), a reversal may be imminent.

Look for market tops to occur when the MFI is above 80. Look for market bottoms to occur when the MFI is below 20.

Calculation

The Money Flow Index requires a series of calculations.

Money Flow = Typical Price * Volume

If today's Typical Price is greater than yesterday's Typical Price, it is considered as a Positive Money Flow and if today's price is less, it is considered as a Negative Money Flow.

Positive Money Flow is the sum of the Positive Money over the specified number of periods. Negative Money Flow is the sum of the Negative Money over the specified number of periods. The Money Ratio is then calculated by dividing the Positive Money Flow by the Negative Money Flow.

Money Ratio = Positive Money Flow / Negative Money Flow

Money Flow Index = 100 - (100 / (1 + Money Ratio))

Parameters

The standard MFI works with a fourteen-day parameter : n = 14

Example

GT::Indicators::MFI->new() GT::Indicators::MFI->new([8])

Links

http://www.equis.com/free/taaz/moneyflow.html http://www.linnsoft.com/tour/techind/mfi.htm

GT::Indicators::TP::calculate($calc, $day)

GT::Indicators::MOM

The standard Momentum is the Momentum 12 days : GT::Indicators::MOM->new() If you need a non standard Momentum use for example : GT::Indicators::MOM->new([9]) or GT::Indicators::MOM->new([25])

GT::Indicators::MOM::calculate($calc, $day)

259 GT::Indicators::MaxDrawDown

Overview

Calculate the MaxDrawDown, which is the worst percentage loss after reaching a maximum.

260 GT::Indicators::MaxPossibleGain

Shows the maximal Gain for a long-strategy

DESCRIPTION

This indicator calculates the maximum gain that is possible in a certain period of time. Be aware that this indicator "knows" the future so don't use it for your trading strategies :)

PARAMETERS

Number of days

The number of days the indicator looks in the future

Data

This is the data to use as input. If you don't specify anything, the high price will be used by default.

261 GT::Indicators::MaxPossibleLoss

Shows the maximal Loss for a long-strategy

DESCRIPTION

This indicator calculates the maximum loss that is possible in a certain period of time. Be aware that this indicator "knows" the future so don't use it for your trading strategies :)

PARAMETERS

Number of days

The number of days the indicator looks in the future

Data

This is the data to use as input. If you don't specify anything, the high price will be used by default.

262 GT::Indicators::OBV

Overview

On Balance Volume (OBV) is a momentum indicator that relates volume price change.

On Balance Volume was developed by Joe Granville and originally presented in his book New Strategy of Daily Stock Market Timing for Maximum Profits.

Calculation

On Balance Volume is calculated by adding the day's volume to a cumulative total when the security's price closes up, and subtracting the day's volume when the security's price closes down.

If today's close is greater than yesterday's close then : OBV = Yesterday's OBV + Today's Volume

If today's close is less than yesterday's close then : OBV = Yesterday's OBV - Today's Volume

If today's close is equal to yesterday's close then : OBV = Yesterday's OBV

Example

GT::Indicators::OBV->new()

Links

GT::Indicators::OBV::calculate($calc, $day)

263 GT::Indicators::PERF

The performance indicator display a security's price performance from a reference day as a percentage. If the market is not available for the reference day, use nearest preceding day.

Note: The day must be given in GT internal format and must match the timeframe.

Example : GT::Indicators::PERF->new(["2001-09-22"]); GT::Indicators::PERF->new(["2001-09-22", "{I:Prices VOLUME}"]);

GT::Indicators::PERF::calculate($calc, $day)

264 GT::Indicators::PFE

Polarized Fractal Efficiency

DESCRIPTION

Parameters

Period (default 10)

The first argument is the period used to calculed the average.

Period 2 (default 5)

Period in which the EMA is calculated.

Exponent (default 2)

Correction-Factor (default 1)

Set this factor to 200 for values > 1000

Datasource

Creation

Link

265 GT::Indicators::PFE

Polarized Fractal Efficiency

DESCRIPTION

Parameters

Period (default 10)

The first argument is the period used to calculate the average.

Exponent (default 2)

Correction-Factor (default 1)

Set this factor to 200 for values > 1000

Datasource

Creation

Link

266 GT::Indicators::PGO

Overview

The Pretty Good Oscillator (PGO) ...

Calculation

PGO = (Close - N-Day SMA of Close) / N-Day EMA of True Range)

Parameters

N = 89

267 GT::Indicators::PP

Overview

Pivot Points and Daily Support and Resistance.

Calculation

The calculation for the new day are calculated from the High (H), low (L) and close (C) of the previous day. Pivot point = P = (H + L + C)/3 First area of resistance = R1 = 2P - L First area of support = S1 = 2P - H Second area of resistance = R2 = (P -S1) + R1 Second area of support = S2 = P - (R1 - S1)

Links

http://www.sixer.com/y/s/education/tutorial/edpage.cfm?f=pivots.cfm&OB=indicators
http://www.tradertalk.com/tutorial/Pivpt.html

GT::Indicators::PP::calculate($calc, $day)

268 GT::Indicators::PercentagePosition

Relative Position in a certain period

DESCRIPTION

This indicators calculates the realtive position in a period. Zero means that a new low is reached and 100 corresponds to a new high.

PARAMETERS

Period 1

The number of days in which the indicator looks for a high/low.

Indicator

The source

269 GT::Indicators::Prices

Return the prices/volume/date of any share

DESCRIPTION

As you often need the prices while using Generic indicators, this modules makes it easy for you to include prices through an indicator: {I:Prices OPEN} or {I:Prices LOW 13330}

PARAMETERS

Data

You have to tell in which data you're interested. You have to choose between OPEN, HIGH, LOW, CLOSE, VOLUME, DATE.

Share

If you don't specify a second argument, you will use the data of the share that you're working on. But sometimes you may want to use the prices of a second share (for comparison, etc). In that case you can specify its code.

270 GT::Indicators::QSTICK

Overview

The QStick indicator was designed by Tushar Chandle to quantify candelsticks.

The distance between opening and closing prices, as known as the size of the candelstick's body, is the heart of candelsticks studies. The QStick indicator is a simple moving average of theses distances.

Interpretation

QStick values below zero show that there is a majority of black candelsticks, so that the stock is under pressure. QStick values upper zero show that there is a majority of white candelsticks, so that the stock is going up.

Note

I don't really like the terms 'majority of black candelsticks' and 'majority of white candelsticks', i prefer to talk about 'negative volatility' and 'positive volatility'. Moreover, it might be more usefull to calculate the QStick with the percentage deviation corrected by the standard deviation instead of the absolute deviation, in order to compares qstick values or to think about levels crossover.

Calculation

QStick Indicator = A-day simple moving average (SMA) of (Close - Open)

Examples

GT::Indicators::QSTICK->new() GT::Indicators::QSTICK->new([20])

Links

http://www.metastock.fr/QSTICK.htm

GT::Indicators::QSTICK::calculate($calc, $day)

271 GT::Indicators::RAVI

RAVI Trendindicator

DESCRIPTION

The RAVI is a simple yet efficient trendindicator. It is calculated as follows:

RAVI = ABS (100 * (SMA(Short) - SMA(Long)) / SMA(Long))

The long Period divided by the short should always be 10. A Trend is indicated if the RAVI crosses the 3%-level.

Parameters

Short Period (default 7)

The first argument is the period used to calculed the short average.

Long Period (default 65)

The second argument is the period used to calculed the long average.

272 GT::Indicators::REMA

Regularized Exponential Moving Average

DESCRIPTION

Regularized EMA This modification of the classical EMA is described in Stock&Commodities (July 2003). It is an adaption that includes the momentum / second derivation of the value into the MA.

It should be used for the calculation of the MACD. The classical EMA is calculated as:

F(n+1)=F(n)+A*[G(n+1)-F(n)] — A(alpha): A=2/(Period+1)

The REMA is calculated as followed:

```
F(n+1)={F(n)*(1+2*L)+A*[G(n+1)-F(n)]-L*[F(n-1)]}/(1+L)

with L(Lambda) as Regularization Factor.
Lambda should be > 0.5
```

Parameters

Period (default 20)

The first argument is the period used to calculed the average.

Lambda (default 0.5)

See above

Other data input

The second argument is optional. It can be used to specify an other stream of input data for the average instead of the close prices. This is usually an indicator (detailed via {I:MyIndică<param>}).

273 GT::Indicators::RMI

Relative Momentum Index

DESCRIPTION

Parameters

Period (default 5)

The first argument is the period used to calculed the average.

Moment distance (default 10)

Creation

```
GT::Indicators::RMI->new()
GT::Indicators::RMI->new([10,20])
```

Links

http://www.geocities.com/burzum_3/rmi.html

GT::Indicators::ROC

The Rate of Change (ROC) is similar to the Momentum. The standard Rate of Change is the ROC 12 days : GT::Indicators::MOM->new() If you need a non standard Momentum use for example : GT::Indicators::MOM->new([9]) or GT::Indicators::MOM->new([25])

Validation

This Indicator was validated by the data available from comdirect.de: The DAX at 04.06.2003 (data from yahoo.com) had a ROC of 8.05. This is consistent with this indicator: 8.0451

GT::Indicators::ROC::calculate($calc, $day)

274 GT::Indicators::RSI

Relative Strength Index

DESCRIPTION

The standard RSI is the RSI 14 days : GT::Indicators::RSI->new() If you need a non standard RSI use for example : GT::Indicators::RSI->new([25])

Validation

This indicators is validated by the values from comdirect.de. The stock used was the DAX (data from yahoo) at the 04.06.2003:

RSI[14,{I:Prices CLOSE}][2003-06-04] = 57.5433 (comdirect=57.54)

275 GT::Indicators::RSquare

Overview

This function calculates the R-Squared coefficient.

Calculation

Pwr(Corr(Cum(1),C,14,0),2)

GT::Indicators::RSquare::calculate($calc, $day)

276 GT::Indicators::Range

The range is nothing more than the difference between the high and the low of the day.

GT::Indicators::Range::calculate($calc, $day)

277 GT::Indicators::SAR

Overview

The Parabolic SAR, developed by Welles Wilder, is used to set trailing price stops. SAR refers to "Stop-And-Reversal". It is designed to create exit points for both long and short positions in such a way that it allows for reactions or fluctuations at the beginning of the position, but accelerates upward (for long positions) or downward (for short positions) as the movement tops out.

Calculation

If Long : SAR(i) = SAR(i-1) + Acceleration Factor * (Extreme Point of the current position - SAR(i-1))

Wilder's acceleration factor (AF) is 0.02 for the initial calculation. Thereafter the AF is increased 0.02 every period there is a New High made. If a new high is not made then the AF is not increased from the last SAR. This continues until the AF reaches 0.2. Once the AF reaches 0.2 it stays at that value for all future SAR calculations until the trade is stopped out.

If Short : SAR(i) = SAR(i-1) - Acceleration Factor * (Extreme Point of the current position - SAR(i-1))

The AF is initially 0.02 and changes by 0.02 intervals until it is 0.2 but the change in the AF is made only after each New Low of a period is made. The AF is never increased above 0.2.

Parameters

Most softwar packages only allow the user to vary the acceleration factor increment and the acceleration factor maximum, fixing the starting acceleration factor at 0.02. This restriction hampers the trend-following abilities of the parabolic, so don't be surprised if GeniusTrader is going a little step further and let you set up your own initial acceleration factor.

Links

http://www.stockcharts.com/education/Resources/Glossary/parabolicSAR.html
http://www.equis.com/free/taaz/parabolicsar.html http://www.linnsoft.com/tour/techind/sar.htm

GT::Indicators::SAR::calculate($calc, $day)

278 GT::Indicators::SMA

Simple Moving Average

DESCRIPTION

A simple arithmetic moving average.

Parameters

Period (default 50)

The first argument is the period used to calculed the average.

Other data input

The second argument is optional. It can be used to specify an other stream of input data for the average instead of the close prices. This is usually an indicator (detailed via {I:MyIndic| <param>}) but it can also be "{I:Prices OPEN}", "{I:Prices HIGH}", "{I:Prices LOW}", "{I:Prices CLOSE}", "{I:Prices FIRST}" and "{I:Prices LAST}" and in which cases the corresponding prices serie will be used.

Creation

```
GT::Indicators::SMA->new()
GT::Indicators::SMA->new([20])
```

If you need a 30 days SMA of the opening prices you can write the following line:

```
GT::Indicators::SMA->new([30, "{I:Prices OPEN}"])
```

A 10 days SMA of the RSI could be created with :

```
GT::Indicators::SMA->new([10, "{I:RSI}"])
```

279 GT::Indicators::SMI

Overview

The Stochastic Momentum Index (SMI) is based on the Stochastic Oscillator. The difference is that the Stochastic Oscillator calculates where the close is relative to the high/low range, while the SMI calculates where the close is relative to the midpoint of the high/low range. The values of the SMI range from +100 to -100. When the close is greater than the midpoint, the SMI is above zero, when the close is less than than the midpoint, the SMI is below zero.

The SMI is interpreted the same way as the Stochastic Oscillator. Extreme high/low SMI values indicate overbought/oversold conditions. A buy signal is generated when the SMI rises above -50, or when it crosses above the signal line. A sell signal is generated when the SMI falls below +50, or when it crosses below the signal line. Also look for divergence with the price to signal the end of a trend or indicate a false trend.

The Stochastic Momentum Index was developed by William Blau and was introduced in his article in the January, 1993 issue of Technical Analysis of Stocks & Commodities magazine.

Calculation

CM = Close - (Highest high(n) + Lowest low(n)) / 2 CM' = EMA(EMA(CM, A), B) HL = Highest high(n) - Lowest low(n) HL' = EMA(EMA(HL, A), B)

%K = 100 * CM' / (HL' / 2) %D = SMA(%K)

Restrictions

This indicator requires that the first four parameters are constant values and will abort otherwise.

Examples

GT::Indicators::SMI->new() GT::Indicators::SMI->new([14, 3, 3, 3])

Links

http://www.fmlabs.com/reference/default.htm?url=SMI.htm http://trader.online.pl/MSZ/ew-Stochastic_Momentum_Indicator.html (note that the former incorrectly uses "-" in CM).

GT::Indicators::SMI::calculate($calc, $day)

280 GT::Indicators::STO

Overview

Developed by George C. Lane in the late 1950s, the Stochastic Oscillator is a momentum indicator that shows the location of the current close relative to the high/low range over a set number of periods. Closing levels that are consistently near the top of the range indicate accumulation (buying pressure) and those near the bottom of the range indicate distribution (selling pressure).

Calculation

%K Fast = 100 * ((Last - Lowest Low(n)) / (Highest High(n) - Lowest Low(n))) %D Fast = M-periods SMA of %K Fast

```
%K Slow = A-periods SMA of %K Fast
%D Slow = B-periods SMA of %K Slow
```

possibly helpful information:

```
%K Fast corresponds to STO/1, %D Fast to STO/2, %K Slow to STO/3
and %D Slow to STO/4

%K Slow may also be known as the stochastic oscillator
%D Slow is also known as the signal line
```

arguments and defaults for STO

STO accepts 7 arguments, the defaults are, in order: 5, 3, 3, 3, {I:Prices HIGH}, {I:Prices LOW}, {I:Prices CLOSE}

```
The first argument is the Period n used in the formula above and for
each of the subsequent SMA periods.

The second argument is M-periods, the third is A-periods, fourth is B-periods.
```

By default the data used is price, which can be changed by specifying different indicators. the order is High(n), Low(n), Last in the %K Fast formula above.

Note that Metastock calculates the slowing via a sum of the last M-periods, rather than a SMA. Metastock displays %K Slow and %D Slow.

Examples

GT::Indicators::STO->new() GT::Indicators::STO->new([14, 3, 3, 3])

Links

http://www.stockcharts.com/education/What/IndicatorAnalysis/indic_stochasticOscillator.html
http://www.equis.com/free/taaz/stochasticosc.html

GT::Indicators::STO::calculate($calc, $day)

281 GT::Indicators::SWMA

The Sine-Weighted Moving Average (SWMA) is a moving average using a sine factor to take into account both time and price movements. Very good at catching tops and bottoms, while filtering out unnecessary noise.

Calculation

SWMA = (Sum of (sin(n*180/6*PI/180) * Close(i)) for i = 1 to i = period) / (Sum of (sin(n*180/6*PI/180)) for i = 1 to i = period)

Examples

GT::Indicators::SWMA->>new() GT::Indicators::SWMA->new([30, {I:Prices OPEN}])

Links

http://www.ivorix.com/en/products/tech/smooth/swma.html

GT::Indicators::SWMA::calculate($calc, $day)

GT::Indicators::SafeZone

The SafeZone stop is described in Dr. Alexander Elder's Book "Come into my Trading Room" and provides stops for closing long or short positions.

It accepts the number of bars to use for the calculation and a coefficient as parameters with 20 and 2 being the defaults that are also used in the examples in the book. The last parameter is the number of days a "plateau" is maintained regardless of of prices moving against the trade. This is to take into account the fact that stops may only be extended in the direction of the trade. After prices have been moving against the trade for the number of bars that is specified by the third parameter it is assumed that the stop was triggered and normal calculation of new stops is resumed.

If this doesn't seem to make sense just plot this indicator and you will know what I am trying to say. :)

GT::Indicators::SafeZone::calculate($calc, $day)

282 GT::Indicators::StandardDeviation

Overview

Standard Deviation is a statistical measure of volatility. Standard Deviation is typically used as a component of other indicators, rather than as a stand-alone indicator. For example, Bollinger Bands are calculated by adding a security's Standard Deviation to a moving average.

Interpretation

High Standard Deviation values occur when the data item being analyzed (e.g., prices or an indicator) is changing dramatically. Similarly, low Standard Deviation values occur when prices are stable.

Many analysts feel that major tops are accompanied with high volatility as investors struggle with both euphoria and fear. Major bottoms are expected to be calmer as investors have few expectations of profits.

Links

http://www.equis.com/free/taaz/standardevia.html

283 GT::Indicators::StandardError

Overview

Standard Error is a statistical measure of volatility. Standard Error is typically used as a component of other indicators, rather than as a stand-alone indicator. For example, Kirshenbaum Bands are calculated by adding a security's Standard Error to an exponential moving average.

Calculation

Calculate the L-Period linear regression line, using today's Close as the endpoint of the line. Note : The term "linear regression" is the same as "least squares" or "best fit" line in some textbooks.

Calculate d1, d2, d3, ..., dL as the distance from the line to the Close of each bar which was used to derive the line. That is, d(i) = Distance from Regression Line to each bar's Close.

Average of squared errors (AE) = (d1š + d2š + d3š + ... + dLš) / L
Standard Error = Square Root of AE

284 GT::Indicators::T3

Overview

T3 is an excellent data-fitting technique by Tim Tillson (cf. "Smoothing Techniques For More Accurate Signals" in Technical Analysis of Stocks and Commodities - January 1998)

Calculation

N is the Exponential Moving Average Period a is the amplification percentage of the filter's response to price movement (0 < a < 1)

e1 = N-days EMA of Close Prices e2 = N-days EMA of e1 e3 = N-days EMA of e2 e4 = N-days EMA of e3 e5 = N-days EMA of e4 e6 = N-days EMA of e5

c1 = (-a)^3 c2 = 3 * a^2 c3 = - 6 * a^2 - 3 * a - 3 * a^3 c4 = 1 + 3 * a + a^3 + 3 * a^2

T3 = c1 * e6 + c2 * e5 + c3 * e4 + c4 * e3

GT::Indicators::T3::calculate($calc, $day, $args, $key, $data)

285 GT::Indicators::TDREI

Tom Demarks REI

DESCRIPTION

A new oscillator introduced by Tom DeMark.

Parameters

Momentum (default 2)
Period (default 10)

Creation

```
GT::Indicators::TDREI->new()
```

Links

286 GT::Indicators::TETHER

Tether Line

DESCRIPTION

The Tether Line is one of the three indicators used in Trend Following System (TFS), designed by Bryan Strain.

CALCULATION

Tether Line = (Highest High (n) + Lowest Low (n)) / 2

PARAMETERS

The standard Tether Line works with a 50-day parameter : n = 50

EXAMPLE

GT::Indicators::TETHER->new() GT::Indicators::TETHER->new([30])

GT::Indicators::TETHER::calculate($calc, $day)

287 GT::Indicators::TMA

Overview

Triangular Moving Averages (TMA) place the majority of the weight on the middle portion of the price series.

Calculation

TMA(5) = (1/9) * (1 * Close(i) + 2 * Close(i - 1) + 3 * Close(i - 2) + 2 * Close(i - 3) + 1 * Close(i - 4))

Examples

GT::Indicators::TMA->new() GT::Indicators::TMA->new([50]) GT::Indicators::TMA->new([30], {I:Prices OPEN})

Links

http://www.equis.com/free/taaz/movingaverages.html http://www.ivorix.com/en/products/tech/sm

GT::Indicators::TMA::calculate($calc, $day)

288 GT::Indicators::TP

Overview

The Typical Price indicator provides a simple, single-line plot of the day's average price. Some investors use th Typical Price rather than the closing price when creating moving average penetration systems.

The Typical Price is a building block of the Money Flow Index.

Calculation

The Typical Price indicator is calculated by adding the high, low and closing prices together, and then dividing by three. The result is the average, or typical price.

Note

The Typical Price is sometimes called "Pivot Point".

Validation

This indicator is indirectly validatet by I:CCI.

GT::Indicators::TP::calculate($calc, $day)

289 GT::Indicators::TR

True Range

DESCRIPTION

The True Range (TR) is designed to measure the volatility between two days.

Calculation

The True Range is defined as the greatest of the following :

- The current high less the current low. - The absolute value of : current high less the previous close. - The absolute value of : current low less the previous close.

Validation

The TR is not directly validated but the ATR matches the data from comdirect.de.

Links

http://www.stockcharts.com/education/What/IndicatorAnalysis/indic_ATR.html
http://www.equis.com/free/taaz/avertrurang.html

290 GT::Indicators::TRIX

TRIX-Indicator from Jack Hutson

DESCRIPTION

The TRIX-Indicator was developed by John Hutson. It is a 1-day-ROC of a threefold EMA (A EMA of a EMA of a EMA). It is calculated as:

```
TRIX = 100 * ( EMA3(t) - EMA3(t-1) ) / EMA3(t-1)
```

The standard interpretation is that a signal is generated if the TRIX cuts the zero-line. It is a relative stable yet not very effective indicator because it generates the signals very late. You can combine the TRIX with an EMA[9] to generate MACD-like signals.

Parameters

Period (default 5)

The first argument is the period used to calculed the average.

Creation

```
GT::Indicators::TRIX->new()
GT::Indicators::TRIX->new([20])
```

Link

```
http://www.incrediblecharts.com/technical/trix_indicator.htm
```

291 GT::Indicators::Test

Indicator to test embedding of indicators

DESCRIPTION

This indicator functions as a test rig to ensure that another indicator is robust in the presence of complex embedding into other indicators.

It is not generic, unfortunately, but tests indicators only with respect to their first argument.

Use it as follows:

```
./display_indicator.pl --start=2000-06-16 --end=2000-06-29 \
I:Test 13000 60 60 60 SMA Generic::MaxInPeriod
```

This will test the SMA; the second indicator Generic::MaxInPeriod is only there to add complexity. It defaults to Generic::MinInPeriod.

This test will do the following:

1. Apply Arg5 to Arg6, using Arg1 as parameter (by default: {I:Generic:MinInPeriod 5 {I:Prices CLOSE}} 2. Smooth the result by Arg4, using Arg2 as parameter (by default: {I:EMA 3 ...} 3. Smooth the result by Arg4, using Arg3 as parameter (by default: {I:EMA 3 ...}

The first output is the result (3), the second output is the result (1).

GT::Indicators::Test::calculate($calc, $day)

292 GT::Indicators::UI

Overview

The Ulcer Index (UI) is a risk measurement tool superior to the standard deviation because it differentiates between rising returns and losses. Investors do not view a set of rising returns as a negative sign of volatility, after all; one of the investor goals is to avoid losses.

Calculation

It is the square root of the average of the squared retracements from the latest high.

Example

GT::Indicators::UI->new()

GT::Indicators::UI::calculate($calc, $day)

293 GT::Indicators::VHF

Overview

The Vertical Horizontal Filter (VHF) can tell you whether a market is going through a trending or congestion phase, and whether you should use trend-following indicators if the markets are trending or congestion-phase indicators if markets are in a trading range.

Calculation

VHF = (Highest Close (n) - Lowest Close (n)) / (Sum of absolute value of the one-day price change for the range (n))

Parameters

The standard VHF works with a 28-days parameter : n = 28

Example

GT::Indicators::VHF->new() GT::Indicators::VHF->new([50])

Links

http://www.equis.com/free/taaz/verthorizfilter.html http://www.finance-net.com/apprendre/techniques/vhf.phtml

GT::Indicators::VHF::calculate($calc, $day)

294 GT::Indicators::VOSC -

OVERVIEW

CALCULATION

EXAMPLES

GT::Indicators::VOSC->new() GT::Indicators::VOSC->new([20])

LINKS

295 GT::Indicators::VROC

Overview

The VROC is the Volume Rate Of Change.

Calculation

VROC = ((Volume(i) * 100) / Volume(i-n)) - 100

Parameters

The standard VROC is equal to GT::Indicators::ROC->new([12], "VOLUME", $GET_VOLUME)

Example

GT::Indicators::VROC->new() GT::Indicators::VROC->new([20])

GT::Indicators::VROC::calculate($calc, $day)

296 GT::Indicators::WMA

Overview

The Weighted Moving Average (WMA) is designed to put more weight on recent data and less weight on past data. A weighted moving average is calculated by multiplying each of the previous day's data by a weight.

Calculation

WMA(5) = (1/15) * (5 * Close(i) + 4 * Close(i - 1) + 3 * Close(i - 2) + 2 * Close(i - 3) + 1 * Close(i - 4))

Examples

GT::Indicators::WMA->new() GT::Indicators::WMA->new([50]) GT::Indicators::WMA->new([30], "OPEN", $GET_OPEN)

Links

http://www.equis.com/free/taaz/movingaverages.html

GT::Indicators::WMA::calculate($calc, $day)

297 GT::Indicators::WTCL

Overview

The Weighted Close indicator is simply an average of each day's price. It gets its name from the fact that extra weight is given to the closing price. The Median Price and Typical Price are similar indicators.

Calculation

The Weighted Close indicator is calculated by multiplying the close by $weight, adding the high and the low to this product, and dividing by (2 + $weight). The result is the average price with extra weight given to the closing price.

Parameters

The standard Weighted Close is configured with : $weight = 2.

Links

http://www.equis.com/free/taaz/weightedclose.html http://www.futuresource.com/industry/wtcl.as

GT::Indicators::WTCL::calculate($calc, $day)

298 GT::Indicators::WWMA

Welles Wilder Moving Average (WWMA) is a modified verson of the EMA.

Calculation

WWMA(i) = (1/n) * Close(i) + (1 - 1/n) * WWMA(i-1)

Examples

GT::Indicators::WWMA->new() GT::Indicators::WWMA->new([15]) GT::Indicators::WWMA->new([30], "OPEN", $GET_OPEN)

299 NOTICES

this version of Welles Wilder Moving Average (WWMA) is depreciated in favor of Wilders (GT::Indicators::Wilders).

GT::Indicators::WWMA::calculate($calc, $day)

300 GT::Indicators::Wilders

Wilder's smoothing (aka wells wilders moving average)

DESCRIPTION

Wilder's smoothing is using simple averages for the initial calculation. For the subsequent average calculations, he drops 1/14th of the previous average value and adds 1/nth of the new value. This is the "classic" exponential average, in which the smoothing factor is 1/n, instead of the "modern" exponential average, in which the smoothing factor is 2/(n+1).

Wilders(i) = (1/n) * Close(i) + (1 - 1/n) * Wilders(i-1)

Parameters

Period (default 14)

The first argument is the period used to calculed the average.

Other data input

The second argument is optional. It can be used to specify an other stream of input data for the average instead of the close prices. This is usually an indicator, including I:Prices.

Creation

```
GT::Indicators::Wilders->new()
GT::Indicators::Wilders->new([20])
```

If you need a 30 days Wilders of the opening prices you can write the following line:

```
GT::Indicators::Wilders->new([30, "{I:Prices OPEN}"])
```

A 10 days Wilders of the RSI could be created with :

```
GT::Indicators::Wilders->new([10, "{I:RSI}"])
```

301 GT::Indicators::WilliamsR

Overview

Williams %R is a momentum indicator developed by Larry Williams that measures overbought/oversold levels.

Calculation

The formula used to calculate Williams' %R is similar to the Stochastic Oscillator :

Williams %R = - 100 * ((Highest High (n) - Close) / (Highest High (n) - Lowest Low (n)))

Parameters

The standard Williams %R works with a 14-days parameter : n = 14

Validation

This Indicator was validated by the data available from comdirect.de: The DAX at 04.06.2003 (data from yahoo.com) had a Williams %R of -5.99. This is consistent with this indicator: -5.99328026152501

Links

http://www.equis.com/free/taaz/williamspercr.html

GT::Indicators::Williams%R::calculate($calc, $day)

302 GT::Indicators::ZigZag

DESCRIPTION

The Zig Zag indicator filters out changes in an underlying plot (e.g., a security's price or another indicator) that are less than a specified amount. The Zig Zag indicator only shows significant changes.

Parameters

Percentage change (10%)

The first argument is the percentage change required to yield a line that only reverses after a change from high to low of 10% or greater.

Creation

```
GT::Indicators::ZigZag->new()
GT::Indicators::ZigZag->new([5])
```

If you need an 8 % ZigZag indicator of the opening prices you can write one of those lines :

```
GT::Indicators::SMA->new([8], "OPEN", $GET_OPEN)
GT::Indicators::SMA->new([8, "OPEN"])
```

A ZigZag indicator with a 20 % threshold of the Volume could be created with :

```
GT::Indicators::ZigZag->new([20, "{I:Volume}"])
```

303 GT::List

List of symbols (shares)

DESCRIPTION

This package provide some simple functions to work with a list of symbols.

Example

Create an empty GT::List object : my $list = GT::List->new();

Load data from a list of symbol : $list->load("/bourse/listes/us/nasdaq");

Add a symbol in a list : $list->add("GeniusTrader");

Remove a symbol in a list : $list->remove("GeniusTrader");

Save list in a file : $list->save("/bourse/listes/us/nasdaq");

Find how many symbols are in a list : $list->count();

Get symbol number $i : $list->get($i);

Functions

`$list->get($i)`
: Get symbol number $i.

`$list->add("symbol")`
: Add a symbol in a list.

`$list->remove($i)`
: Remove the symbol number $i.

`$list->count()`
: Find how many symbols are in the list.

`$list->load("list_of_symbol.txt")`
: Load data from a list of symbol.

`$list->save("list_of_symbol.txt")`
: Save list in a file.

304 GT::MarketCalculator

Calculator-like for Markets

DESCRIPTION

THIS OBJECT IS NOT USED ANYWHERE AT THE PRESENT TIME

This is a facility object to ease the collaboration of between GT::List, GT::CacheValues and GT::MetaInfo

`my $market = GT::Markets->new($list [, $name])`

Create a new GT::Markets object with $list used for calculations. The market is associated to market $name.

`$market->indices()`

`$market->indicators()`

`$market->signals()`

`$market->metainfo()`

Return the corresponding object that is part of GT::Markets.

`$market->set_name($name)`

Set market object name.

`$market->name()`

Return market object name.

305 GT::MetaInfo

Keep various meta informations

DESCRIPTION

Goal

This object is used to gather meta informations of different kinds applying to various objects (state of a trading system, history of order, support & resistance of prices, top & bottom prices on a period, lines drawn on a graph, ...). It stores various informations in an internal XML structure that can be stored in a file and reloaded later.

Informations are stored on a key/value basis. Key is a path in an XML DOM tree (/ is the separator like for xpath expressions). Several values can be stored in a single key if you use attributes to distinguish them.

API

`my $info = GT::MetaInfo->new;`

Create a new empty GT::MetaInfo object.

`$info->set($key, $value, { "attrname" => "attrvalue" });`

Stores $value corresponding to $key with an attribute "attrname" (whose value is "attrvalue"). The third parameter is optional if you don't need attributes for this key.

If the key already existed, then the value is replaced.

`$info->get($key, { "attrname" => "attrvalue" });`

Get the value corresponding the $key and the attributes indicated in the second optional argument.

`my @list = $info->list($key, { "attrname" => "attrvalue" });`

List all the set of attributes available for elements corresponding to the $key. You can eventually restrict the list to a subset og them by specifying one or more attributes.

Each element of @list is a hash describing the attributes. If you want the value of that node you have to use $info->get(...).

`$info->load("/path/to/file.xml")`

Load the XML file as MetaInfo object.

`$info->save("/path/to/file.xml")`

Save the current GT::MetaInfo object in the given XML file. All values previously set can be reloaded later with $info->load(...).

`$info->dump`

Output the current internal XML file to the standard output. Useful for debug purposes.

306 GT::MoneyManagement

Money management rules (risk management)

DESCRIPTION

Money management rules decide or modify the sum of money placed on each trade. On the extreme side, they can cancel an order by deciding that 0 shares should be bought/sold.

`$mm_rule->manage_quantity($order, $i, $calc, $portfolio)`

> Return the quantity that the money management rule would put on the given order. $order->{'quantity'} may be already set by a previous money management rule. Never modify the quantity directly but return the new proposed quantity.

307 GT::MoneyManagement::Alembert

Introduction

The d'Alembert System is a progression system which tries to win back your losses in small steps instead of all at once like the Martingale. It was designed for use on the even chance bets on a roulette table but can be used on any even chance bet.

Concept

The d'Alembert System works under the assumption that over a period of time there will be an equal number of 'reds' and 'blacks'. We start the session by placing one unit ($1, $5 or any other value) on one of the even chance bets (e.g. 'red'), after a losing spin we increase the next bet by one unit and after a winning bet we decrease the next bet by one unit. So if we were betting oin 'red' and the spins were - black, black, black, red, black, red, red, black, red, red, red - then the bets placed would be as follows (the numbers in brackets show the level of your bankroll after the spin):

1 (-1), 2 (-3), 3 (-6), 4 (-2), 3 (-5), 4 (-1), 3 (+2), 2 (+0), 3 (+3), 2 (+5), 1 (+6)

This sequence would end with a win of $6. As you can see, as soon as the number of 'reds' is equal to the number of 'blacks' plus one then the sequence ends with a win. You may also notice that after the 7th, 9th and 10th spins we were also showing a profit, this is because the bets placed on winning spins are one unit greater than the previous losing spin. Having the possibility of a positive bankroll before the sequence is complete allows us to choose to cut the session short and take a smaller win rather than risking the chance of the session ending badly.

Links

http://www.casino-help.com/systems/dalembert.shtml

308 GT::MoneyManagement::AntiMartingale

Introduction

Before reading this you should look at the Martingale system. This system is designed for use on the even chance bets on a roulette table but can be used on any even chance bets.

Concept

Playing the Anti-Martingale is precisely what the name suggests, it's the Martingale in reverse. Instead of doubling your bets after a losing spin you double your bet after a winning spin. Basically you place a bet on an even chance and 'let it ride' for a set number of spins. Before starting the system you must decide how many spins you are going to play for, alternatively you may decide to remove your winnings when your nerve breaks or even remove part of the winnings after a number of spins and let the remaining bet continue for a little bit longer.

Links

http://www.casino-help.com/systems/martingale.shtml http://roulette.casino.com/article.pl/aid=m

309 GT::MoneyManagement::Basic

Basic and dumb money management rules. Invest all cash available (provided that no marged position block the cash - in this money management rule each dollar invested in marged position requires 1 dollar in cash).

310 GT::MoneyManagement::CheckCommissions

Overview

This money management rule will keep an eye to the size of each trade. Trade only when commissions represents less than a fixed percentage of the investment.

Parameters

By default, we will accept all trades where commissions represents less than 1 % of the investment and reject others.

311 GT::MoneyManagement::CheckVolumeAverage

Overview

This money management rule will keep an eye to the size of each trade to remain them below a fixed percentage of the n-days volume average.

Parameters

By default, we will accept all trades representing less than 1 % of the 5 days average volume and reject others.

312 GT::MoneyManagement::FixedFractional

Overview

This money management rule will allowed to each trade a fixed fraction of the current portfolio value.

313 GT::MoneyManagement::FixedRatio

Overview

This money management rule is described in Ryan Jones's book "The Trading Game" as an alternative to the standard Fixed Fractional type of money management rules.

314 GT::MoneyManagement::FixedShares

Overview

This money management rule will allowed to each trade exactly the same number of shares. The default number is set up to 100 shares.

315 GT::MoneyManagement::FixedSum

Overview

This money management rule will invest the same amount of money in each trade. The default value is set up to 1000.

316 GT::MoneyManagement::Martingale

Introduction

The Martingale system is probably the oldest of betting systems, many other systems are based on the basic theory of the Martingale, and so to evaluate most systems you need a full understanding of the Martingale.

Concept

The Martingale is a progression system (i.e. you increase your bet after a losing spin) played on the even chance bets on a roulette table although it can be used on even chance bets on other games, and the basic idea is that if you bet on one of the even chance bets (e.g. Red) eventually it will hit. With this in mind, if you increase your bets after each losing spin so that you win back all your losses plus one unit you will always walk away a winner. In order to win all previous losses back plus one unit you simply need to double your bet each time:

e.g. If you lost four consecutive spins and then won on the fifth spin the outcome of each spin would be (-1) + (-2) + (-4) + (-8) + (+16) = +1

You would have placed a total of 31 units at the start of the fifth spin, and when red hit on the fifth spin you would pick up 32 units.

Links

http://www.casino-help.com/systems/martingale.shtml http://roulette.casino.com/article.pl/aid=m

317 GT::MoneyManagement::OrderSizeLimit

Overview

This money management rule will keep an eye to the size of each order to remain them below a fixed percentage of the portfolio value.

318 GT::MoneyManagement::Basic

Basic and dumb money management rules (ie no rules).

319 GT::MoneyManagement::Basic

Basic and dumb money management rules (ie no rules).

320 GT::MoneyManagement::PositionSizeLimit

Overview

This money management rule will keep an eye to the size of each position to remain them below a fixed percentage of the portfolio value.

321 GT::MoneyManagement::RSI

Overview

This new money management technique utilize the Relative Strength Index (RSI) in order to improve the performance of trend-following trading.

References

"A New Money Management Technique" - Takehide Matoba Article found in http://www.erivativesreview.com

322 GT::MoneyManagement::STO

Overview

This new money management technique utilize the Stochastics (STO) in order to improve the performance of trend-following trading.

References

"A New Money Management Technique" - Takehide Matoba Article found in http://www.erivativesreview.com

323 GT::MoneyManagement::ShareMultiples

Overview

This money management rule will provide you a tool to buy/sell round lots of shares (ie: multiples of 5, 10 or 50).

Parameters

By default, the first parameter is initialized to 10 and will provide share multiples of 10 stocks. The second option is set up to zero and represent the calculation method, but look at all options :

0 : round to the nearest multiple 1 : round to the lower multiple 2 : round to the upper multiple

324 GT::MoneyManagement::VAR

Overview

This method uses market volatility and the concept of value at risk (VAR) to help determine meaningful stop-loss prices and position limits for trading securities.

References

"Value At Risk And Technical Analysis" by Luis Ballesca-Loyo Technical Analysis of Stocks and Commodities - August 1999

325 GT::OrderFactory

Create orders

An OrderFactory is used to create an order when a system has detected an opportunity. This order will then be sent to the PortfolioManager by the SystemManager.

```
$of->create_buy_order($calc, $i, $sys_manager, $pf_manager)
$of->create_sell_order($calc, $i, $sys_manager, $pf_manager)
```

> Those functions are called by the systems to launch an order. The SystemManager delegates this to an Order object. It will use the Order object given by set_default_order() or it will fallback to the order suggested by the system.

326 GT::OrderFactory::ChannelBreakout

DESCRIPTION

This module is able to set up a generic channel breakout strategy based on two generic limited price order, above and below current prices. Both levels will be defined with an indicator.

327 GT::OrderFactory::StopOnExtreme

DESCRIPTION

Create an order that will be x% above or below current close prices.

328 GT::OrderFactory::MarketPrice

DESCRIPTION

Create an order at market price.

329 GT::OrderFactory::MaximumSlippage

DESCRIPTION

In a Maximum Slippage test, we rig the software so that buy orders always suffer the maximum possible slippage : all buys occurs at the High of the day, and similarly all sells occurs at the Low of the day.

This idea came from author Fred Gehm; it's a torture test designed to see whether a system is robust against slippage.

330 GT::OrderFactory::MinimumSlippage

DESCRIPTION

In a Minimum Slippage test, we rig the software so that buy orders always catch the minimum possible slippage : all buys occurs at the Low of the day, and similarly all sells occurs at the High of the day.

331 GT::OrderFactory::SignalClosingPrice

DESCRIPTION

This module will send virtual order at the closing price when a new signal is given.

332 GT::OrderFactory::StopOnExtreme

DESCRIPTION

Create a "stop" order that will with the limit the high of the day (modulo x%) for a long position and the low of the day for a short position.

333 GT::Portfolio

A portfolio

DESCRIPTION

A Portfolio is used to keep track of orders. It can calculate a performance and give useful statistics about what you've done (average trade gain/loss, percentage of winning/losing trades, max draw down, ...).

`my $p = GT::Portfolio->new;`

Create a portfolio object without any open positions and without any pendings orders (ie an empty portfolio).

`$p->add_order($order)`

Add $order to the list of pending orders.

`$p->discard_order($order)`

Discard the order. Usually that means that it hasn't been executed or that it has been cancelled.

`$p->new_position($code, $source, $date)`

Create a new open position in the portfolio.

`$p->apply_order_on_position($position, $order, $price, $date)`

Add the given order to the position and modify the money available in the portfolio accordingly.

`$p->close_position($pos)`

Move the position from the list of open positions to the historic list. Update the cash with the marged gain.

`$p->apply_pending_orders($calc, $i, $source, $pf_manager, [ $cb ])`

Check the pending order for the value indicated by $calc, try to execute them on the day $i. It restricts itself to the orders coming from the indicated source. You can pass an optionnal callback for managing specially the position_opened callback. Not giving this arg and leaving to its default value is usually ok.

`$p->apply_pending_orders_on_position($position, $calc, $i)`

Apply all pending orders on the position, this does include the stop.

`$p->update_position_evaluation($position, $calc, $i)`

Update the evaluation of the position with data of day $i.

`$p->store_evaluation($date)`

Store the cash level and the evaluation of the portfolio for the indicated date.

`$p->current_cash()`

Returns the sum of cash available (may return a negative value if "effet de levier" is used).

`$p->current_evaluation()`

Returns the evaluation of all the open positions in the portfolio.

`$p->current_marged_gains()`

Returns the sum of gains (or losses if the number is negative) made with marged positions.

`$p->current_marged_investment()`

Returns the sum of gains (or losses if the number is negative) made with marged positions.

```
my($cash, $evaluation, $gains) = $p->get_historic_evaluation($date)
```

Return the historic information (cash and portfolio evaluation) about the portfolio.

`$p->has_historic_evaluation($date)`

Returns true if an evaluation of the portfolio exists for the given date.

`$p->list_pending_orders([$source])`

Returns the list of orders that are pending and that have been submitted by the corresponding source. If source argument is missing (or undef), returns all the pending orders.

`$p->list_open_positions([$source])`

Returns the list of positions that are open and that have been submitted by the corresponding source. If source argument is missing (or undef), returns all the open positions.

`$p->get_position($code, $source)`

Return the position (if any) corresponding to $code and $source. This assumes that only one such position exists.

`$p->list_history_positions($code, $source)`

Return the list of historical positions corresponding to $code and $source.

`$p->set_initial_value()`

Set the amount of money available initially on the portfolio.

`$p->set_broker($broker)`

Defines which broker to use for the calculation of order commissions and annual account charge.

`$p->get_order_cost($order)`

Apply all broker rules and return the amount ask by the broker for the given order.

`$p->real_global_analysis()`

`$p->real_analysis_by_code($code)`

Analyzes the evolution of the portfolio. Either globally or for each share individually.

Real analysis uses just the information provided. For a global analysis, it needs an initial value for the portfolio.

The informations calculated are :

```
- global gain/loss (sum & percentage)
- number of winning trades
- number of loosing trades
- average loss (percentage)
- average gain (percentage)
- max gain in single trade (percentage)
- max loss in single trade (percentage)
- max global gain
- max global loss
- max draw down (biggest cumulated loss after a new high) (percentage)
```

334 GT::Portfolio::Order

An order within the portfolio

DESCRIPTION

Internal structure

```
 {
    "order" => "B",         # Buy/Sell
    "type" => "L",          # Limited|Stop|APD|ATP|TR
    "code" => "13000",
    "quantity" => 100,
    "price" => "12.4",      # Main price
    "price2" => "12.6",     # Second limit (if needed)
    "source" => "Trend",    # Trading system that opened the position
                            # maybe "manual"
    "date" => "2001-07-01", # date of submission
    "validity" => "2001-07-02", # valable until this day
    "no_discard" => 1,      # don't remove the order automatically next day
    "id" => 123             # id automatically assigned when added to
                            # the portfolio
 }
```

Functions

`$o = GT::Portfolio::Order->new;`

`$o->set_sell_order()`

`$o->set_buy_order()`

`$o->is_sell_order()`

`$o->is_buy_order()`

`$o->set_type($type)`

`$o->get_type()`

Manage the type of the order. Valid types are :

M
: Market price

L
: Limit

S
: Stop

APD
: A plage de déclenchement (french market only)

TR
: Tout ou rien (french market)

ATP
: A tout prix (french market)

`$o->set_type_{limited,market_price,stop,stop_limited}()`

`$o->is_type_{limited,market_price,stop,stop_limited}()`
: Change/checks the type of the order.

`$o->set_code($code)`

`$o->code()`
: Set/get the symbol of the traded share.

`$o->set_quantity($quantity)`

`$o->quantity()`
: Set/get the quantity of shares.

`$o->set_price($price)`

`$o->price()`

`$o->set_second_price($price)`

`$o->second_price()`
: Set/get the prices on the order.

`$o->set_source($source)`

`$o->source()`

`$o->set_submission_date($date)`

`$o->submissiont_date()`

`$o->set_indicative_stop($price)`

`$o->indicative_stop()`

`$o->set_not_discardable()`

`$o->set_discardable()`

`$o->discardable()`
: A normal order has a validity of one period (ie one day usually). If you want to place an order that should be kept until it's executed (a close on target for example) you need to modify the order by calling this function on it.

`$o->set_id($id)`

`$o->id()`

`$o->set_attribute($key, [ $value ]);`

`$o->has_attribute($key);`

`$o->attribute($key);`

`$o->delete_attribute($key);`

An order can have "attributes" associated to keep track of its status in various strategies. has_attribute returns only true if the attribute exists (whatever its value is). attribute returns the attribute value if it exists or undef otherwise.

`$o->set_timeframe($timeframe)`

`$o->timeframe()`

Set and return the timeframe associated to this order.

`$o->set_marged()`

`$o->set_not_marged()`

`$o->is_marged()`

A marged order will not cost cash since the cash is "rented" until the position is closed.

`$o->is_executed($calc, $i)`

Returns the price of execution if the order has been executed. Otherwise returns 0.

335 GT::Portfolio::Position

An open position within a portfolio

DESCRIPTION

Internal structure

```
  {
    "long" => 1,            # 1:Long 0:Short
    "code" => "13000",
    "quantity" => 100,
    "initial_quantity" => 100, # Quantity when position was opened
    "open_price" => 12.4,      # Price when position has been taken
    "close_price" => 13.2,     # Prices when position has been closed
    "source" => "Trend",       # Which trading system opened the position
    "open_date" => "2001-07-01",
    "close_date" => "2001-07-04",
    "stop" => 12               # Stop is at 12
  }
```

Functions

```
$p = GT::Portfolio::Position->new($code, $source, $date)
$p->set_long()
$p->set_short()
$p->is_long()
$p->is_short()
$p->set_code($code)
$p->code()
$p->set_quantity($quantity)
$p->quantity()
$p->set_initial_quantity($quantity)
$p->initial_quantity()
$p->set_open_price($price)
$p->open_price()
$p->set_close_price($price)
$p->close_price()
$p->set_source($source)
$p->source()
```

`$p->set_open_date($date)`

`$p->open_date()`

`$p->set_close_date($date)`

`$p->close_date()`

`$p->set_id($id)`

`$p->id()`

`$p->set_stop($price)`

`$p->update_stop($price)`

`$p->force_stop($price)`

`$p->stop()`

set_stop and update_stop modifies the stop level but it won't let you further the stop, you can only bring it nearer. If you want to further the stop level use force_stop.

`$p->set_attribute($key, [ $value ]);`

`$p->has_attribute($key);`

`$p->attribute($key);`

`$p->delete_attribute($key);`

A position can have "attributes" associated to keep track of its status in various strategies. has_attribute returns only true if the attribute exists (whatever its value is). attribute returns the attribute value if it exists or undef otherwise.

`$p->set_timeframe($timeframe)`

`$p->timeframe()`

Set and return the timeframe associated to this position.

`$p->set_marged()`

`$p->set_not_marged()`

`$p->is_marged()`

A marged position is constitued of shares that have been "rented" (or bought with rented cash).

`$p->apply_order($order, $price, $date)`

Update the position with the corresponding order. The order has been executed at the given date and at the given price.

`$p->apply_pending_orders($calc, $i)`

`$p->add_order($order)`

Add a new pending order to the position.

`$p->delete_order($order)`

`$p->discard_order($order)`

`$p->list_pending_orders()`

Returns the list of pending orders on the position. Those orders have not yet been executed.

`$p->list_detailed_orders()`

Returns the list of detailed orders on the position. Those orders have already been executed.

`$p->is_open()`

Returns true if the position is still open (ie if quantity != 0)

`$p->set_intent_to_close()`

Mark the position as being in the process of being closed. This will let the system detect new opportunities as soon as possible.

`$p->set_no_intent_to_close()`

DeMark the position as being in the process of being closed. This will let the system manage the position without knowing that some orders have been placed to close the position (ie those orders may or may not be executed).

`$p->being_closed()`

Returns true if the position is in the process of beeing closed.

`$p->stats($portfolio, [ $quantity_factor ])`

Calculate some statistics about the position.

336 GT::PortfolioManager

Manages a portfolio

DESCRIPTION

A PortfolioManager is an entity interacting between a Portfolio, a Trading System, money management rules and trade filters.

When it comes to starting a new position (ie submitting an order to start a position), the money management system comes in again to decide how much to put on the trade.

Filters can be applied to accept/refuse trades proposed by the various trading systems.

`my $manager = GT::PortfolioManager->new($portfolio)`

Create a new portfolio manager that implements a money management strategy.

`$manager->set_portfolio($portfolio)`

Change the portfolio managed.

`$manager->portfolio()`

Returns the managed portfolio.

```
$order = $manager->buy_market_price($calc, $source)
$order = $manager->buy_limited_price($calc, $source, $price)

$order = $manager->buy_conditional($calc, $source, $price [, $price2])

$order = $manager->virtual_buy_at_open($calc, $source)
$order = $manager->virtual_buy_at_high($calc, $source)
$order = $manager->virtual_buy_at_low($calc, $source)
$order = $manager->virtual_buy_at_close($calc, $source)
$order = $manager->virtual_buy_at_signal($calc, $source)
$order = $manager->sell_market_price($calc, $source)
$order = $manager->sell_limited_price($calc, $source, $price)

$order = $manager->sell_conditional($calc, $source, $price [, $price2])

$order = $manager->virtual_sell_at_open($calc, $source)
$order = $manager->virtual_sell_at_high($calc, $source)
$order = $manager->virtual_sell_at_low($calc, $source)
```

`$order = $manager->virtual_sell_at_close($calc, $source)`

`$order = $manager->virtual_sell_at_signal($calc, $source)`

Those functions are used to create orders that may be modified and submitted later.

`$manager->set_order_partial($order, $ratio)`

When you want to close a position, you may want to not close it fully. With this function, you indicate how much of the initial position you want to close.

`$manager->discard_all_orders($calc, $source)`

Discards all orders concerning this share and this source.

`$manager->submit_order($order, $i, $calc)`

`$manager->submit_order_in_position($position, $order, $i, $calc)`

Submit the prepared order, either an order that will start a new position or as an order that will modify an existing position.

`$manager->add_money_management_rule($mm_rule)`

`$manager->delete_all_money_management_rule()`

`$manager->default_money_management_rule($mm_rule)`

Use a money management rule and remove all money management rules currently used.

`$manager->decide_quantity($order, $i, $calc)`

Apply the various money management rules and decide the size of the position.

`$manager->finalize()`

Finalize the setup of the manager. Calculate its name. You can get its name afterward using $manager->get_name.

`$manager->get_name()`

Return the name of the system.

`$manager->setup_from_name($name)`

Setup the portfolio manager according to the name, it will create the corresponding money management rules.

337 GT::Prices

A serie of prices

DESCRIPTION

GT::Prices stores all historic prices (open, high, low, close, volume, date).

`my $p = GT::Prices->new()`
: Create an empty GT::Prices object.

`$p->at(i)`
: Get the prices of the corresponding day. The indice can be obtained from the dates by using $q->date('YYYY-MM-DD').

`$p->at_date('YYYY-MM-DD')`
: Get the prices of the corresponding date.

`$p->has_date('YYYY-MM-DD')`
: Return true if the object has prices for the corresponding date.

  NOTE: If we test for an item that is larger than the last entry in the prices array, then a new empty entry is created (and numerous error messages as well).

`$p->date('YYYY-MM-DD')`
: Get the indice corresponding to the date 'YYYY-MM-DD'.

`$p->add_prices_array([@price_array])`

`$p->add_prices([$open, $high, $low, $close, $volume, $date])`

`$p->count()`
: Get the number of prices availables.

`$p->set_timeframe($timeframe)`

`$p->timeframe()`
: Defines the time frame used for the prices. It's one of the value exported by GT::DateTime;

`$p->sort()`
: Sort the prices by date.

`$p->reverse()`
: Reverse the prices list.

`$p->convert_to_timeframe($timeframe)`
: Creates a new Prices object using the new timeframe by merging the required prices. You can only convert to a largest timeframe.

`$p->find_nearest_following_date($date)`

`$p->find_nearest_preceding_date($date)`

`$p->find_nearest_date($date)`
: Find the nearest date available

`$p->loadtxt("cotationsfile.txt")`
: Load the prices from the text file.

`$p->savetxt("cotationsfile.txt")`
: Save the prices to the text file.

`$p->dump;`
: Print the prices on the standard output.

`$p->_binary_search($array_ref, $value)`
: Searches for the given $value in the $DATE position of the prices array. This is an internal function, meant to be used only inside this object.

338 GT::PricesTools

Utility functions for manipulating GT::Prices

DESCRIPTION

This package provide some simple functions to merge data from an existing GT::Prices to a new GT::Prices object. It's especially usefull to convert daily data to a new time frame (ie: weekly/monthly).

Examples

```
convert_prices_in_a_new_time_frame($prices, $WEEKLY);
select_prices_by_period($prices, "2000-01-01", "2000-12-31");

multiply_prices_by_number($prices, 3.45);
divide_prices_by_number($prices, 2.5);

reverse_prices($prices)
```

Ideas for later

```
adjust prices by the last rate of a currency
adjust prices by daily historical rate of a currency
```

339 GT::Registry

Generic registry functions

DESCRIPTION

This module is used by GT::Indicators, GT::Signals and GT::Systems to keep a list of available objects. Those objects can be reused with different datas.

`GT::Registry::get_registered_object($repository, $name)`

Returns the object corresponding to $name if available. Otherwise returns undef.

`GT::Registry::register_object($repository, $name, $object)`

Register the object $object under the name $name. Replaces any previous object registered under the same name.

`GT::Registry::get_or_register_object($repository, $name, $object)`

If an object corresponding to name $name is already registered then returns this object. Otherwise register $object under the name $name.
This function is intented to be used by constructor of objects. Once the constructor know the name of the object, it uses this function to bless the object reference. It will check if an object with the same name exists. In that case the registered object is used instead of creating a new one. Otherwise the object in creation is blessed, stored in the registry and returned.
Example:

```
sub new {
    my $type = shift;
    my $class = ref($type) || $type;

    # Get the name of the object
    my $name = get_indicator_name(...);
    my $self = {};

    # Check the registry and register it
    $self = GT::Indicators::get_or_register_object($name, $self);

    return $self;
}
```

`GT::Registry::manage_object($repository, \@NAMES, $obj, $class, $args, $key)`

Manage the creation of a new object. Build their names, stores and/or retrieve the object from the database. Calls initialize for a new object.

`GT::Registry::build_object_name($encoded, [ @args ], $key)`

Returns the real $name of the object by substitution of #1, #2 (and so on) by the real values of the parameters (given in the second argument). "($key)" is appended at the end. It's used to differentiate similar objects but using a different input method for example (think about indicators like "Average" working on prices or any other value).

`Method for "named" objects`

```
get_name() or get_name($i)
get_nb_values()
```

340 GT::Report

Generate visual report of common objects

DESCRIPTION

This modules provides various functions to dump to GT::Report::OUT (by default STDOUT) various objects in a nice formatted text.

`GT::Report::Portfolio($portfolio)`
: Prints the content of a portfolio in text format.

`GT::Report::PortfolioHTML($portfolio)`
: Prints the content of a portfolio in HTML format.

`GT::Report::OpenPositions($portfolio, $detailed)`
: Display the list of open positions.

`GT::Report::PortfolioAnalysis($analysis)`
: Pretty prints the results of the analysis of the portfolio.

`GT::Report::AnalysisList`
: Display the results of the backtest. Results per code and per system.

`GT::Report::SimplePortfolioAnalysis`
: Pretty prints only the main results of the analysis.

`GT::Report::CacheValues`
: Prints a summary of the content of the cache.

341 GT::Serializable

Add XML serialization functions to any object

DESCRIPTION

The functions available in GT::Serializable can add serialization support to any simple perl object.

The various functions will serialize any hash, array or scalar value blessed as an object.

Any object can be made serializable by adding GT::Serializable in @ISA :

```
our @ISA = qw(GT::Serializable);
```

All hash items whose names start with an underscore won't be stored in the serialization process.

RESTRICTIONS

Any reference to something else than a hash, array or scalar value will be ignored (including function and file descriptor).

HOOKS

Once an object is created from scratch based on a serialization dump, $object->init_after_load() is called so that the object has a chance to restore things that may have not been stored (such as reference to internal functions).

FUTURE

We may define later other hooks that will let the module personalize the name of elements used to store the object in the XML file.

FUNCTIONS

```
$self->as_string()
```

```
my $a = Module->create_from_string()
```

```
$self->store("file" | \*FILE)
```

```
my $a = Module->create_from_file("file" | \*FILE)
```

342 GT::Signals

Base module for all signals

DESCRIPTION

Overview

Signals are objects that returns true/false for each quotation. This value doesn't have any other direct meaning (ie it's not buy/sell). However those results will probably be used by trading systems (in cunjunction with other informations) to decide what to do (buy/sell/update a stop/nothing).

Detailed description

`my $sig = GT::Signals::AnExample->new([ @args ])`
: Create a signal object with the appropriate parameters.

`$sig->get_name or $sig->get_name($i)`
: Get the name of the signal. If the signal returns several values, you can get the name corresponding to any value, you just have to precise in the parameters the index of the value that you're interested in.

`$sig->get_nb_values`
: Return the number of different values produced by this signal that are available for use.

`$sig->initialize()`
: This callback function is called at creating time. Since the "new" function is inherited, you should do the initialization via this function.

`$sig->detect($calc, $i)`
: Stores the value of the signal for the day $i.

`$sig->detect_interval($calc, $first, $last)`
: Stores the value of the signal for all the days of the specified interval.

General exported functions

`build_object_name($encoded, [ @args ], $key)`
: Generate the name of a signal based on its "encoded" name.

Functions to manage a repository of signals

```
GT::Signals::get_registered_object($name);
GT::Signals::register_object($name, $object);
GT::Signals::get_or_register_object($name, $object);
GT::Signals::manage_object(\@NAMES, $self, $class, $args, $key);
```

343 Above Generic Signal

Overview

This Generic Signal will be able to tell you when a specific indicator is above a given value which could be an other indicator or a fixed limit.

EXAMPLE

You can check if the RSI is above 80 with this signal:

```
S:Generic:Above {I:RSI} 80
```

344 And Combination Signals

Overview

This Generic Signal will be give a positive signals only when all mentionned signals also give positive signals.

EXAMPLE

You can use this signal to check if the closing prices is above 10 and below 15 :

```
S:Generic:And {S:Generic:Above {I:Prices CLOSE} 10} {S:Generic:Below {I:Prices CLOS
```

345 Below Generic Signal

Overview

This Generic Signal will be able to tell you when a specific indicator is below something else which could be an other indicator, a limit or current prices.

EXAMPLE

You can check if the Security is trading below the 200 day Exponential Moving Average with this signal:

```
S:Generic:Below {I:Prices CLOSE} {I:EMA 200}
```

346 CrossOver Generic Signal

Overview

This Generic Signal will be able to tell you when a specific indicator is crossing down an other one.

EXAMPLE

You can check test if the closing price has crossed under the 14 day EMA, with this signal:

```
S:Generic:CrossOverDown {I:Prices CLOSE} {I:EMA 14}
```

347 CrossOver Generic Signal

Overview

This Generic Signal will be able to tell you when a specific indicator is crossing down an other one.

EXAMPLE You can check test if the closing price has crossed over the 14 day EMA, with this signal:

```
S:Generic:CrossOverUp {I:Prices CLOSE} {I:EMA 14}
```

348 Decrease Generic Signal

Overview

This Generic Signal will be able to tell you when a specific indicator is decreasing from its previous level.

EXAMPLE

You can use this signal to determine if the securities closing price is decreasing

```
S:Generic:Decrease {I:Prices CLOSE}
```

349 Equal Generic Signal

Overview

This Generic Signal will be able to tell you when a specific indicator is equal to something else which could be an other indicator, a limit or current prices.

EXAMPLE

You can use this signal to test if the securities closing price equals 1.86

```
S:Generic:Equal {I:Prices CLOSE} 1.86
```

350 False

Always return false.

EXAMPLE

```
S:Generic:False
```

351 Increase Generic Signal

Overview

This Generic Signal will be able to tell you when a specific indicator is increasing from its previous level.

EXAMPLE

You can use this signal to determine if the securities closing price is increasing.

```
S:Generic:Increase {I:Prices CLOSE}
```

352 NewTimeFrame Generic Signal

Overview

This signal will tell you when you entered a new timeframe (e.g. the next month/week/year).

353 Not Signal Negation

Overview

This Generic Signal will reverse the value of the signal parameter it receives.

EXAMPLE

You can use this signal to check if the closing prices are not decreasing:

```
S:Generic:Not {S:Generic:Decrease {I:Prices CLOSE}}
```

354 Or Combination Signals

Overview

This Generic Signal will be give a positive signals when anyone of the mentionned signals also give a positive signal.

EXAMPLE

You can use this signal to check if the closing prices is below 10 or above 15 :

```
S:Generic:Or {S:Generic:Below {I:Prices CLOSE} 10} {S:Generic:Above {I:Prices CLOSE
```

355 GT::Signals::Generic::Repeated

Detect repetition of a given signal

DESCRIPTION

This generic Signal will give a positive signal when the mentionned signal has been positive for the last X days (where X is the second parameter of this signal with a default value of 2).

EXAMPLE

You can check if the RSI has been above 80 for the last 3 days with this signal:

```
S:Generic:Repeated {S:Generic:Above {I:RSI} 80} 3
```

356 True

Always return true.

EXAMPLE

```
S:Generic:True
```

357 GT::Signals::Graphical::CandleSticks::BearishEngulfingLi

Overview

The Bearish Engulfing Line is a strongly bearish if it occurs after a significant up-trend (i.e., it acts as a reversal pattern). It occurs when a small bullish (empty) line is engulfed by a large bearish (filled-in) line.

Engulfing Lines are one of the most important and powerful candle patterns. To form the pattern, today's range encloses or "engulfs" the prior day's range, thereby indicating great market strength in the direction of today's close.

From a psychological perspective, Engulfing Lines indicate that the market opened in the same direction as the prior day, but then reversed sentiment to close strongly in the opposite direction, overcoming and reversing yesterday's assumption regarding value.

Construction

If yesterday closed lower, a Bullish Engulfing Line will form when today's open is below yesterday's close abd today's close is above yesterday's open. The Bearish Line is just the opposite, where a black body enckises a previous white body.

Representation


```
Bearish Engulfing Line
```

Links

http://www.equis.com/free/taaz/candlesticks.html

358 GT::Signals::Graphical::CandleSticks::BearishHarami

Overview

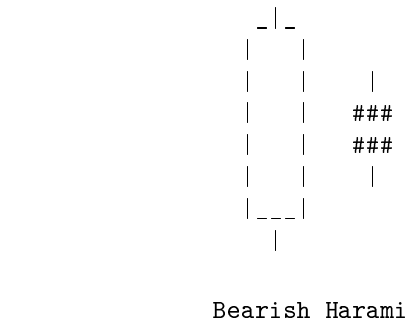
The Bearish Harami signifies a decrease of momentum. It occurs when a small bearish (filled-in) line occurs after a large bullish (empty) line in such a way that close of the bullish line is above the open of the bearish line and the open of the bullish line is lower than the close of the bearish line.

The Bearish Harami is a mirror image of the Bullish Engulfing Line.

Construction

If yesterday closed higher, a Bearish Harami will form when today's open is below yesterday's closed and today's close is above yesterday's open.

Representation



Bearish Harami

References

1. More information about the bearish harami on Page 33 of the book "Candlestick Charting Explained" by Gregory L. Morris. Morris says that this pattern suggests a trend change.

2. Steve Nison also says that the Harami Patterns suggest a trend change. This is on page 80 of his book "Japanese Candlesticks Charting Techniques".

3. http://www.equis.com/Customer/Resources/TAAZ/.

359 GT::Signals::Graphical::CandleSticks::BullishEngulfingLin

Overview

The Bullish Engulfing Line is a strongly bullish pattern if it occurs after a significant downtrend (i.e., it acts as a reversal pattern). It occurs when a small bearish (filled-in) line is engulfed by a large bullish (empty) line.

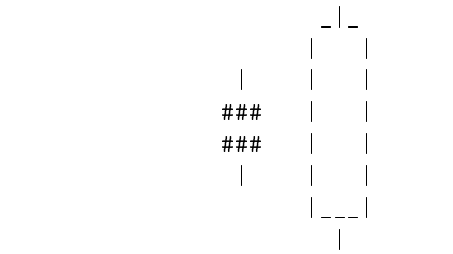
Engulfing Lines are one of the most important and powerful candle patterns. To form the pattern, today's range encloses or "engulfs" the prior day's range, thereby indicating great market strength in the direction of today's close.

From a psychological perspective, Engulfing Lines indicate that the market opened in the same direction as the prior day, but then reversed sentiment to close strongly in the opposite direction, overcoming and reversing yesterday's assumption regarding value.

Construction

If yesterday closed lower, a Bullish Engulfing Line will form when today's open is below yesterday's close abd today's close is above yesterday's open. The Bearish Line is just the opposite, where a black body enckises a previous white body.

Representation



```
Bullish Engulfing Line
```

Links

http://www.equis.com/free/taaz/candlesticks.html

360 GT::Signals::Graphical::CandleSticks::BullishHarami

Overview

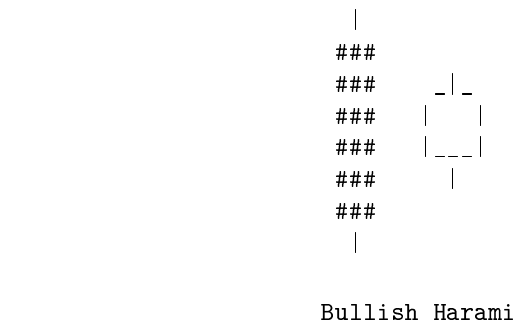
The Bullish Harami signifies a decrease of momentum. It occurs when a small bullish (empty) line occurs after a large bearish (filled) line in such a way that close of the bullish line is above the open of the bearish line and the open of the bullish line is lower than the close of the bearish line.

The Bullish Harami is a mirror image of the Bearish Engulfing Line.

Construction

If yesterday closed higher, a Bullish Harami will form when today's open is above yesterday's close and today's close is above yesterday's open.

Representation



Bullish Harami

Links

1. More information about the bullish harami on Page 33 of the book "Candlestick Charting Explained" by Gregory L. Morris. Morris says that this pattern suggests a trend change.

2. Steve Nison also says that the Harami Patterns suggest a trend change. This is on page 80 of his book "Japanese Candlesticks Charting Techniques".

3. http://www.equis.com/Customer/Resources/TAAZ/.

361 GT::Signals::Graphical::CandleSticks::GravestoneDoji

Overview

The Gravestone Doji is a reversal pattern that signifies a turning point. It occurs when the open, close, and low are the same, and the high is significantly higher than the open, low, and closing prices.

Representation

```
  |
  |
  |
---

79
```

Links

http://www.equis.com/free/taaz/candlesticks.html

362 GT::Signals::Graphical::CandleSticks::Hammer

Overview

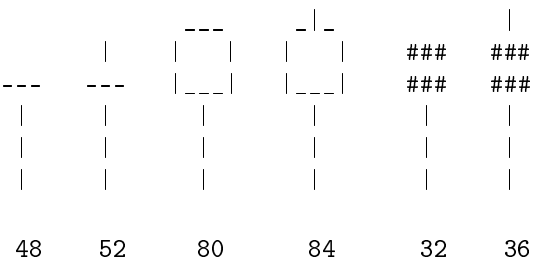
The Hammer Pattern is formed by a short body at the top of a long trail. Hammers must occur at the end of significant trends to have meaning.

Hammers indicate indecision in the direction of the trend. A black (solid) hammer which occurs at the end of an uptrend is called a Hanging Man. Thsi type of Hammer indicates the market's propensity to sell off sharply. However, one should wait for the next session to confirm the bearish mood (i.e., for the market to open below the close of the hammer). On the other hand, white (open) Hammers which occur at the end of downtrends show strength for a reversal to the upside, since the bulls are clearly bucking the downtrend to close near the open for the session.

Construction

A Hammer occurs when the high, open and close occur at roughly the same price, but the low of the day is far below.

Representation



Links

http://www.equis.com/free/taaz/candlesticks.html

363 GT::Signals::Graphical::CandelSticks::InvertedHammer

Overview

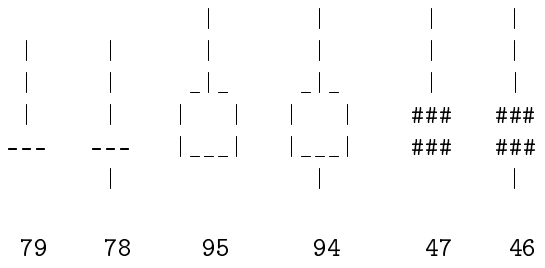
Inverted Hammers are just the opposite of Hammers (see GT::Signals::Graphical::CandleSticks::Hamm
i.e., a small body occurs at the bottom of a long trail. Black Inverted Hammers occuring at the end of uptrends are clearly bearish, since the markets fails in its attempt to rally higher, closing near the open.

However, white Inverted Hammers at the end of downtrends are more subtle, and it is important to establish the following day as bullish. It is likely that a short-covering rally will ensue, thereby confirming the reversal.

Construction

An inverted Hammer is formed when the open, close and low occur at approximately the same price, with a high extended significantly above the three. The farther away the high is for the day, the more signifiant the pattern in terms of forecasting a reversal.

Representation



364 GT::Signals::Indicators::RSIDown

Signal when we cross down a limit on the RSI. Limit is 70 by default.

365 GT::Signals::Indicators::RSIDown

Signal when we cross up a limit on the RSI. Limit is 30 by default.

366 GT::Signals::Advance

DESCRIPTION

The Advance Signal will be able to tell you if a security is advancing more than x % or not from the previous period. Advance, Decline and Unchange Signals are basics signals and will be your row materials for designing lots of market indicators.

367 GT::Signals::Decline

DESCRIPTION

The Decline Signal will be able to tell you if a security is declining mor thant x % or not from the previous period. Advance, Decline and Unchange Signals are basics signals and will be your row materials for designing lots of market indicators.

368 GT::Signals::GapDown

DESCRIPTION

Gaps form when opening price movements create a blank spot on the chart. Gaps are especially significant when accompanied by an increase of volume.

A down gap forms when a security opens below previous period's low, remains below the previous low for the entire period and close below it.

Down gaps can form on daily, weekly or monthly charts and are generally considered bearish.

369 GT::Signals::GapUp

DESCRIPTION

Gaps form when opening price movements create a blank spot on the chart. Gaps are especially significant when accompanied by an increase of volume.

An up gap forms when a security opens above previous period's high, remains above the previous high for the entire period and close above it.

Up gaps can form on daily, weekly or monthly charts and are generally considered bullish.

370 GT::Signals::Prices::InsidePrevious

The InsidePrevious signal gets triggered if a security's high is lower than or equal to the previous period's high, and the low is higher than or equal to previous period's low.

371 GT::Signals::Unchange

DESCRIPTION

The Unchange Signal will be able to tell you if a security is unchange or not from the previous period.

Advance, Decline and Unchange Signals are basics signals and will be your row materials for designing lots of market indicators.

372 GT::Signals::Swing::Trend

373 GT::Signals::Swing::TrendUpEnding

An up-trend is going on and a little candle is constated. The trend may be ending ...

374 GT::Signals::MacdDiff

375 GT::Signals::Trend::HilbertChannelBreakout

376 GT::Signals::Trend::TTT

Trade The Trend !! Use Hilbert period to detect the start of a trend.

377 GT::Signals::Volatility::NR

DESCRIPTION

NR is for Narrowest Range. It is parametered with the period length to look at for the size of ranges.

378 GT::SystemManager

Manages trading systems

DESCRIPTION

A SystemManager is an entity interacting between a PortfolioManager, a Trading System (signals), TradeFilters, OrderFactory and CloseStrategy.

Filters can be applied to accept/refuse trades proposed by the trading system.

A system manager is not completely defined until all desired objects have been "linked" to it using all the add_* and set_* functions. When all those calls have been made, you should call **finalize** to let the manager know that you've finished setting it up. After that, the system manager can be identified with an unique (and quite long) name.

Later you'll be able to setup the same system manager by using setup_from_name($name).

`my $sm = GT::SystemManager->new($system)`
: Create a new system manager used to control a trading system.

`$sm->set_system($system)`
: Define the system that is managed.

`$sm->system()`
: Return the system managed by this manager.

`$manager->add_trade_filter()`

`$manager->delete_all_trade_filter()`
: Use a trade filter and remove all trade filters currently used.

`$self->accept_trade($order, $i, $calc, $pf_manager)`
: Apply all the trade filters to the proposed tarde and return the result (accepted or not).

`$self->send_buy_order($calc, $i, $pf_manager)`

`$self->send_sell_order($calc, $i, $pf_manager)`
: Those functions are called by the systems to launch an order. The SystemManager delegates this to an Order object. It will use the Order object given by set_default_order() or it will fallback to the order suggested by the system.

`$self->set_order_factory($order_factory)`
: Defines which OrderFactory object will be used to send the orders.

`$self->add_position_manager($close_strategy)`

`$self->delete_all_position_manager()`
: Add a position manager to the chain of position manager. A position manager is better known as "CloseStragegy".

`$sm->manage_position($calc, $i, $position, $pf_manager)`

Manages a open position with the current system.

`$sm->position_opened($calc, $i, $position, $pf_manager)`

Has to be called once a position has been opened and wants to be managed by this system manager.

`$sm->get_indicative_stop($calc, $i, $order, $pf_manager)`

Get an indicative stop level for the position that will be opened by this order.

`$sm->apply_system($calc, $i, $pf_manager)`

This function will use the generated signals to pass the order. It delegates this responsibility to the PortfolioManager.

`$sm->precalculate_interval($calc, $i, $first, $last)`

`$sm->finalize()`

Finalize the setup of the manager. Calculate its name. You can get its name afterward using $sm->get_name

`$sm->get_name()`

Return the name of the system.

`$sm->setup_from_name($name)`

Setup the system manager according to the name.

`$sm->set_alias_name($name)`

`$sm->alias_name`

`my $sm = GT::SystemManager::get_registered_object($name)`

Returns the system manager corresponding to this name.

379 GT::Systems -

DESCRIPTION

Trading systems are systems that decide what to buy and sell at which prices and so on.

A system will propose buy/sell orders. The day after, those orders will be either cancelled or confirmed. If the order is confirmed, then a position is considered open. An open position will then be managed by a close strategy.

`$system->long_signal($calc, $i)` =**item** `$system->short_signal($calc, $i)`

The system can generate 2 signals (buy or sell). A signal is an intent to buy or sell. Those functions should be overriden by the specific system.

`$system->set_long_signal()`

`$system->set_short_signal()`

Facility function to set which signal is used to generate buy/sell signal. They are meant to be used in initialize only if long_signal and short_signal are not overriden.

`$system->precalculate_all($calc)` =**item** `$system->precalculate_interval($calc, $first, $last)`

If you run a system on a long period of time you may want to precalculate all the indicators in order to benefit of possible optimizations. This is the role of those 2 functions.

`$system->default_order_factory()`

Return an object OrderFactory that can be used if no other objects was to be used.

Functions to manage a repository of systems

```
GT::Systems::get_registered_object($name);
GT::Systems::register_object($name, $object);
GT::Systems::get_or_register_object($name, $object);
GT::Systems::manage_object(\@NAMES, $object, $class, $args, $key);
```

`GT::Systems::Module->new($args)`

Create a new Systems with the given arguments. $args is optional.

380 Trend following system

381 Trend following system 2

382 AlwaysInTheMarket Trading System

Overview

This system will generate every time it is called a long and a short signal.

Examples

To set up a Buy And Hold strategy : SY:AlwaysInTheMarket|CS:NeverClose|TF:LongOnly|TF:OneTr

To set up a Short And Hold strategy : SY:AlwaysInTheMarket|CS:NeverClose|TF:ShortOnly|TF:One

Note

In which way is a BuyAndHold or a ShortAndHold system usefull ?

The main purpose is to run the system on a list of securities, in order to get the portfolio performance. Let's try to catch indices performance with just a few stocks and beat the market only with money-management ! Welcome to the world of portfolio selection and portfolio optimization.

383 Generic System module

```
a system runs on two signals: long and short

these terms imply buying shares when the long signal is triggered
and selling shares short when the short signal is triggered. in
either case a new position is thus opened (or maybe added to)
for the security that caused the signal.

a generic system description requires two signals, the first
must define the long signal, the second defines the short signal.

out on a limb here -- a system description can specify only
one long and one short signal

nb: the short signal will not necessarily close a position
opened by a prior long signal. open position management is
controlled by a close strategy.

according to GT::Systems a signal is acted upon in the following
day (timeframe). if a position is opened (long or short) closing
that position is controlled by a close strategy (CS). therefore,
most system descriptions will also include a close strategy
description.

system description can specify multiple close strategies

system description can specify multiple money management (MM)
strategies

how does OrderFactory (OF) fit in with system description?
```

384 Generic System Examples

```
SY:Generic \
 {S:Generic:CrossOverUp   {I:SMA 20 {I:Prices CLOSE}} {I:SMA 60 {I:Prices CLOSE}}}
 {S:Generic:CrossOverDown {I:SMA 20 {I:Prices CLOSE}} {I:SMA 60 {I:Prices CLOSE}}}
| TF:OneTrade \
| CS:OppositeSignal

the CrossOverUp signal denotes the buy condition
the CrossOverDown signal denotes the sell condition
```

```
notice that the close strategy explicitly specifies that the short
signal is also used to close open positions, and conversely
short positions are closed with the long signal.

the trade filter (TF) (see perldoc ../GT/TradeFilters.pm) will limit
the number of open positions (either long or short) to one, however
it will allow a new open position and a closed postion in the same timeframe.

system descriptions can specify multiple trade filters

SY:Generic inherits new from GT::Systems::
```

385 Trend following system

386 Stochastic

387 Stochastic

388 Trend following system

389 Trend Following System (TFS)

390 Turtle Trading System (TTS)

Overview

The Turtle Trading System is a very simple and very easy to understand.

It's an Asymetric Channel Breakout :

* Enter long above the highest high of the previous X days and exit with a stop based on the lowest low of the Y previous days with Y < X

* Enter short below the lowest low of the previous W days and exit with a stop based on the highest high of the Z previous days with Z < W

Parameters

Y = length of channel for longs W = length of channel for shorts =cut

sub new { my $type = shift; my $class = ref($type) || $type; my $args = shift;

```
    my $self = { "args" => defined($args) ? $args : [ 55, 20 ] };

    return manage_object(\@NAMES, $self, $class, $self->{'args'}, "");
}
```

sub initialize { my ($self) = @_;

```
    $self->{'max'} = GT::Indicators::Generic::MaxInPeriod->new([ $self->{'args'}[0],
    $self->{'min'} = GT::Indicators::Generic::MinInPeriod->new([ $self->{'args'}[1],

    $self->add_indicator_dependency($self->{'min'}, 2);
    $self->add_indicator_dependency($self->{'max'}, 2);
    $self->add_prices_dependency($self->{'args'}[0] + 1);
    $self->add_prices_dependency($self->{'args'}[1] + 1);
}
```

sub precalculate_interval { my ($self, $calc, $first, $last) = @_; $self->{'max'}->calculate_interval($calc, $first, $last); $self->{'min'}->calculate_interval($calc, $first, $last);

```
    return;
}
```

sub long_signal { my ($self, $calc, $i) = @_;

```
    return 0 if (! $self->check_dependencies($calc, $i));
```

```
    if ( ( $calc->prices->at($i)->[$CLOSE] >
           $calc->indicators->get($self->{'max'}->get_name, $i - 1) )
       )
    {
        return 1;
    }
    return 0;
}

sub short_signal { my ($self, $calc, $i) = @_;

    return 0 if (! $self->check_dependencies($calc, $i));

    if ( ( $calc->prices->at($i)->[$CLOSE] <
           $calc->indicators->get($self->{'min'}->get_name, $i - 1) )
       )
    {
        return 1;
    }
    return 0;
}

1;
```

391 GT::Tools

Various helper functions

DESCRIPTION

This modules provides several helper functions that can be used in all modules and scripts.

There are 5 groupings

- math – min, max, pi, sign
- generic – extract_object_number
- conf – resolve_alias resolve_object_alias long_name short_name
- isin – isin_checksum isin_validate isin_create_from_local
- timeframe – get_timeframe_data parse_date_str find_calculator check_dates

math

It provides mathematical functions, that can be imported with use GT::Tools qw(:math) :

`PI()`
: Returns PI.

`min(...)`
: Returns the minimum of all given arguments.

`max(...)`
: Returns the maximum of all given arguments.

`sign($value)`
: Returns 1 for a positive (or null) value, -1 for a negative value.

generic

It provides helper functions to manage arguments in "Generic" objects. You can import those functions with use GT::Tools qw(:generic) :

`extract_object_number(@args)`
: Returns the number associated to the first the object described by the arguments.

conf

And a few other very-specific functions :

use GT::Tools qw(:conf) :

`resolve_alias($alias)`

Return the long name of the system as described in the configuration file.

`resolve_object_alias($alias, @param)`

Return the complete description of the object designed by "alias". @param is the array of parameters as returned by GT::ArgsTree::parse_args().

Object aliases can be defined in global files (as defined in the option Path::Aliases::<object_kind>), for each kind of object (e.g., Signals, Indicators, etc.), or in user-specific files (~/.gt/aliases/<object_kind>).

Such aliases are defined via the syntax <alias_name> <definition>

For example MyMean { I:Generic:Eval (#1 + #2) / 2 }

An object alias can also be defined in the option file with the syntax Aliases::<object_kind>::<alias_name> <definition>

For example:

```
Aliases::Indicators::MyMean  { I:Generic:Eval ( #1 + #2 ) / 2 }
```

Then you can use this alias in any other place where you could have used a standard indicator as argument. Here's how you would reference it with custom parameters :

```
{ @I:MyMean 50 {I:RSI} }
```

If you don't need any parameters then you can just say "@I:MyMean".

`my $l = long_name($short)`

`my $s = short_name($long)`

Most module names can be shortened with some standard abreviations. Those functions let you switch between the long and the short version of the names. The recognized abreviations are :

- Analyzers:: = A:
- CloseStrategy:: = CS:
- Generic:: = G:
- Indicators:: = I:
- MoneyManagement:: = MM:
- OrderFactory:: = OF:
- Signals:: = S:
- Systems:: = SY:
- TradeFilters:: = TF:

isin

use GT::Tools qw(:isin) :

`isin_checksum($code)`
: This computes the checksum of a given code. The whole ISIN is returned.

`isin_validate($isin)`
: Validate the ISIN and its checksum.

timeframe

use GT::Tools qw(:timeframe) :

`GetTimeFrameData ($code, $timeframe, $db, $max_loaded_items)`
: Returns a prices and a calculator object with data for the required $code in the specified $timeframe. It uses $db object to fetch the data. If for instance, weekly data is requested, but only daily data is available, the weekly data is calculated from the daily data.

  Optionally, you can set the configuration file directive DB::timeframes_available to specify which timeframes are available. For instance: DB::timeframes_available 5min,hour,day

`parse_date_str ( \$date_string, \$err_msg )`
: Returns 1 if \$date_string is valid parsable date, zero (or null) otherwise \$date_string will be altered to be a gt compliant date string on return \$err_msg is optional

  - input params must be references to the object
  - if called in void context with bad date value the internal error handling will put error message text on stderr and die called
  - date ref var may be altered to conform to std date-time format
  - error string will contain details about bad date-time string

  If the user has Date::Manip installed it allows the use of date strings that can be parsed by Date::Manip in addition the to defacto standard date-time format accepted by GT (YYYY-MM-DD HH:MM:SS) time part is optional

  Date::Manip is not required, without it users cannot use short-cuts to specify date strings. such short cuts include –start '6 months ago' –end 'today'

  The date string checking includes verifying the date string format is valid and the date is a valid date (and time if provided)

  Errors will be displayed and the script will terminate.

Application usage examples: with Date::Manip installed

```
%    scan.pl --timeframe day --start '6 months ago' \
        --end 'today' market_file 'today' system_file
```

without Date::Manip you will need to use:

```
%    scan.pl --timeframe day --start 2007-04-24 \
        --end 2007-10-24 market_file 2007-10-24 system_file
```

or

```
%    scan.pl --timeframe week --start 2007-04-24
        --end 2007-10-24 market_file 2007-10-24 system_file
```

Usage of parse_date_str in application script

```
use GT::Tools qw( :timeframe );
# tag name to get &parse_date_str visibility

my $err_msg;
# get date string from command line
my $date = shift;

my ( $d_yr, $d_mn, $d_dy, $d_tm );
if ( ! parse_date_str( \$date, \$err_msg ) ) {
  die "Error: $err_msg\n";
} else {
  ( $d_yr, $d_mn, $d_dy, $d_tm ) = split /[- ]/, $date;
}
```

`find_calculator($code, $timeframe, $full, $start, $end, $nb_item, $max_loaded_item)`

Find a calculator: Returns $calc (the calculator), as well as $first and $last (indices used by the calculator).

The interval examined (bound by $first and $last) is computed as follows (stop whenever $first and $last have been determined): 1. if present, use –start (otherwise default $first to 2 years back) and –end (otherwise default $last to last price) 2. use –nb-item (from first or last, whichever has been determined), if present 3. use first or last price, whichever has not yet been determined, if –full is present 4. otherwise, use two years worth of data.

Note that the values given to –start and –end are relative to the selected time frame (i.e., if timeframe is "day", these indicate a date; if timeframe is "week", these indicate a week; etc.). Format is "YYYY-MM-DD" for dates, "YYYY-WW" for weeks, "YYYY-MM" for months, and "YYYY" for years.

`check_dates($timeframe, $start, $end, $date)`

Converts the given dates into the proper dates relative to the chosen time frame, if necessary. For example, if a date 2000-02-01 is given with –timeframe=week, this date is converted to 2000-05.

Verifies that the start date is before the end.

If a third date is given, verifies that this date is between the start and end dates.

392 GT::TradeFilters

Filters to accept or refuse trades

DESCRIPTION

Trade filters are used to decide whether or not a trade is accepted. It can for example refuse trade going against the current trend. You can use several trade filters simultaneously.

`$filter->accept_trade($order, $i, $calc, $portfolio)`

`$system->precalculate_all($calc)`

`$system->precalculate_interval($calc, $first, $last)`

If you run a system on a long period of time you may want to precalculate all the indicators in order to benefit of possible optimizations. This is the role of those 2 functions.

393 GT::TradeFilters::AcceptAll

Accept all trades

DESCRIPTION

TradeFilter accepting all trades.

394 GT::TradeFilters::AroonTrend

Allow only trades following the trend defined by Aroon

DESCRIPTION

This filter tries to limit the risks by refusing trades againts the market (ie like buying in a bear market or selling in a bullish market).

395 GT::TradeFilters::FollowTrend

Allow only trades following the direction of an SMA

DESCRIPTION

This filter tries to limit the risks by refusing trades againts the market (ie like buying in a bear market or selling in a bullish market).

The first parameter is the number of days used to calculate the SMA.

396 TradeFilters::Generic

Accept or refuse trades based on specific signals

DESCRIPTION

This tradefilter takes two signals as parameter. The first decides if a buy order is allowed, the second one decides if a sell order is allowed. If you don't precise a parameter, the corresponding orders will be refused.

EXAMPLES

Allow buy orders only when SMA 20 is moving up and sell orders when SMA 20 is decreasing :

```
TF:Generic {S:Generic:Increase {I:SMA 20}} {S:Generic:Decrease {I:SMA 20}}
```

397 GT::TradeFilters::LongOnly

Only allow long trades

DESCRIPTION

This filter allows only long trades and reject short ones. This is especially usefull for people who don't want to short the market. Moreover, this filter is a must to find if a system perform better as a long only system than either a short only or a long and short trading system.

398 GT::TradeFilters::MaxOpenTrades

Refuse more than N trades

DESCRIPTION

This filter refuses a new trade if more than N positions are open. It will however accept it if it's in the process of being closed.

399 GT::TradeFilters::OneTrade

Refuse simultaneous trades

DESCRIPTION

This filter refuses a new trade if a position is actually open. It will however accept it if it's in the process of being closed.

400 GT::TradeFilters::ShortOnly

Only allow short trades

DESCRIPTION

This filter allows only short trades and reject long ones. This filter is a must to find if a system perform better as a short only system than either a long only or a long and short trading system.