Introduction
To
OpenQuant Strategy Development

Version 2007-02-02

1 Introduction

OpenQuant is a modern IDE development system for designing and executing computerized quantitative trading strategies, based on the more powerful SmartQuant core framework.

Here is a list of other OpenQuant documents. If you're a new OpenQuant user, you might consider reading the documentation in the following order to maximize your learning speed.

1. **OpenQuant Getting Started Manual**. This manual shows you how to use the IDE to configure and execute a simple strategy. You can choose a financial instrument, run a strategy, and inspect the simulation results.

2. **OpenQuant Strategy Development Manual**. This manual shows you how to design and code strategies on user-specified financial instruments. The manual begins with an overview of trading system design, then moves on to OpenQuant system concepts, and finally discusses the OpenQuant code for several strategies in depth.

1.1 Goals of This Document

This document is about how to write code for the OpenQuant system. It talks about the macro code structure of strategy frameworks. It talks about the micro code implementations of several example strategies, and talks about code events that you can use to implement their strategies.

After reading this document, you should have a basic understanding of trading system design, including some critical things that you need to get right and some pitfalls that you need to avoid. You should also have a feel for the general theories behind popular types of strategies, and how to implement simple strategies—in code—within the OpenQuant system. Once you finish reading this document, it should be easy for you to write and back test your first few trading strategies.

In particular, this document discusses the following subject areas:

1. **General Trading System Design Issues.** Compatibility between you and your trading system; major types of trading systems; money and exposure management; diversification; general trading system pitfalls such as over fitting and brittle systems.

2. **OpenQuant Framework Code Blocks.** A complete framework code example; structure and content of C# code files; declarations; initialization method; system events (OnBarOpen, OnPositionOpened, OnBar, etc); code for placing trading orders; code for drawing a series on a price chart;

3. **OpenQuant Example Strategies.** Code for Breakout strategies; confirmation methods (e.g. 3 consecutive closes above a breakout point); oscillator strategies; moving average strategies; Gap closing strategies; Altucher strategies from *Trade Like a Hedge Fund* book; Chande strategy from the *Beyond Technical Analysis* book.

4. **Terminology.** Definitions of terms used in this document; references to books on trading systems and strategy development.

1.2 Intended Audience

This document is an introductory document about how to write code for the OpenQuant system, so it only covers basic automated trading system concepts and some typical strategy scenarios. It does not cover coding techniques for advanced strategy development issues or data management.

2 General Trading System Design Issues

The goal of this section is to provide a perspective on the "big picture" of strategy development, before digging into the details of how to code strategies with the OpenQuant system. The big picture discussion talks about two important things:

1. Why it is important that you and your strategy are compatible, and what you can do about increasing the match between you and your trading system;

2. A list of some popular kinds of trading systems, so that you can pick a trading system that is compatible with your trading beliefs.

2.1 You and Your Trading System

Successful trading is not just about the trading system itself—instead, it's about the *combination* of you *and* your trading system.

Many systems can provide good results according to their design. But that doesn't mean that *you* can make money with them. Maybe you can't stand the tedium of making too few or too many trades. Maybe you can't stand the huge draw downs during hard times. Maybe you can't let the

trades run when they're showing huge profits. All sorts of reasons and pressures can convince you to abandon a working trading system, and maybe to make it unprofitable for *you*.

So it's not just about the trading system—it's about your ability to trade it too.

2.1.1 Compatibility with Your Trading Beliefs

The main compatibility problem is that in the heat of trading, you might want to do something different than the trading system wants to do. Who gives the orders?

For example, if you're in a deep drawdown with fully 50 percent—half!—of your equity gone into losing trades, what do you do when the trading system is telling you to keep on riding the existing (and future) trades that the system is thinking up? Your risk control alarms are screaming to close the positions, because the "portfolio heat" is too much for you. What do you end up doing? How can you mitigate or avoid this problem?

The main solution to the compatibility problem is to trade a system that you believe in, so that you can stick with it during hard times. You can't do anything about the market, but you can do something about yourself and your trading system. Pick an approach that you believe in, and then test it extensively to deepen your understanding and belief in it. Run it with reasonable money management (trade sizing) and exposure (portfolio risk) constraints. Only then will you have a chance of sticking with the system when your emotions run high and your equity runs low.

2.1.2 Use Technical Methods that You Believe In

To increase your chances of trading success, pick a trading approach that you believe in. There are many types of profitable systems to choose from—here is a list of some of them:

- Breakout Systems
- Trend Following Systems
- Trend/AntiTrend (Reversal) Systems
- Range Trading Systems
- Gap Closing Systems
- Volatility Systems
- Intermarket Correlation Systems
- Arbitrage Systems

More information on these trading systems is provided in the sections below. Read the general description of each system, and see which ones intuitively appeal to you the most. Perhaps those systems might be good starting points for your next strategy development project.

You might also consider the list of book references near the beginning of this document for more ideas and deeper information on trading systems of various kinds. Many of the books in the list are excellent sources of useful trading information.

2.1.3 Testing Gives You Emotional Strength for the Hard Times

Once you've learned enough about trading systems to pick one that you like, the most important thing you can do to deepen your belief in the system is to back test it against various sets of historical data for various markets.

If you do a good job of testing, so that you *know* that your system is robust enough to survive (and be profitable) in a wide variety of past market situations, then you will have more belief in it's behavior and abilities in future markets, during horrendous draw downs.

In particular, once you are satisfied with your back testing results on historical data, the next step forward is to "paper trade" your strategy against real time data (but with fake money). This is an important step to take because it exposes your strategy (and you) to the ups and downs of the real market, in real time. Once you are satisfied with your paper trading results for a few days or weeks or months (the interval must fit your emotional needs), then you can start live trading your strategy and feel confident about your expected results.

2.2 Major Types of Trading Systems

This section summarizes some popular kinds of trading systems—what concepts they're based on; what they're good at; and what they're not good at. These summaries are not comprehensive. They are only intended to draw a "big picture" landscape of possible trading system types, from which you can choose particular strategies that interest you. All of these strategies are explained in the book reference list near the beginning of this document.

2.2.1 Systems Bet With or Against a Situation

To help novices characterize trading systems, it is worth mentioning that there are two general classes of trading systems—those that expect a situation to continue (to trend forward), and those that expect (or try to force by trading) a situation to change.

Systems that bet with a situation are all trend following systems of one kind or another. They use different technical mechanisms to enter and follow and exit a trade, but they all go with the initial change, and hope it becomes a trend that they can follow for profit. Two examples of these systems include the Breakout and Trend following systems described below.

In contrast, systems that bet against a situation include many systems that are qualitatively different from each other. Some systems bet against simple price direction (Anti-trend systems). Some bet against open gaps (Gap Closing systems). Some bet against differences in prices for the same thing in different markets (Arbitrage systems). And some bet against the average value of something (Mean Reversion systems), where the average value can be anything from the direction or distance between two price series to the volatility of a single price series or of the distance between two series.

Furthermore, systems that bet with a situation (such as trend following systems) usually want the situation to continue for a long and profitable time, whereas systems that bet against a situation want the situation to change as soon as possible (so as to capture a profit from the change as quickly as possible). So if the markets permit it, and if the strategy entry criteria permit it, "bet with" systems will make fewer and longer trades to earn their profit, while "bet against" systems will usually try to make more trades and shorter trades to earn their profit. (The trade entry criteria are important—for example, your gap-closing strategy could wait for months before it sees a 5% gap down to bet against).

2.2.2 Systems are Long, Short, or Out

Systems can also be classified by their possible trading states—in or out of the market (state), and (direction) long or short the market.

Single state systems are always in the market. Single state systems are not selective about market conditions. They can only decide to be long or short, even if market conditions do not favor any trades at all (such as a very flat market).

Two-state systems can be in or out of the market. These systems can be patient and selective about which trades they want to enter. They have the option of sitting out unfavorable market conditions, so they are more flexible than single state (always in the market) systems.

A unidirectional system can go long, or short, but is restricted so that it cannot switch between long or short. For example, a conservative system might only allow long trades. In contrast, a bidirectional system can go long or short as it pleases.

Note that single state unidirectional systems make no sense at all—they would have to be permanently in the market long or short forever. So for practical purposes, all single state systems must be bidirectional (like a trend-reversal or trend-flipping system).

2.2.3 Breakout Systems

The concept of a breakout system is that when something significant happens to an instrument, the price will break out (either up or down) of its previous trading range. Breakout systems monitor the usual width of the price channel (the range), and open new positions when the price breaks out of the channel.

One of the oldest futures trading systems was based on this idea. The typical trading rules were something like: Buy when the price breaks above the highest high of the past four weeks, or below the lowest low of the past four weeks. Sell when you reach a profit target, or when a trailing stop loss was hit.

Breakout systems are good for capturing moves that break out and take off some distance from the channel, and they save you from trying to trade small moves as prices whipsaw back and forth within the price channel. But they can whipsaw you too, by providing false breakouts that suddenly return back into the channel after you've open a position in the breakout direction.

Some breakout systems use extra criteria to "confirm" that the breakout is valid. For example, you might require that one, two, or three bars close outside the channel before opening a position. Further, you might require that all three bars close farther away from the channel each day.

In all strategies, adding more selective confirmation criteria will usually decrease the number of trades made because more trades (both winning and losing) are excluded. But if the criteria ideas work well, both the accuracy (% winning trades) and profitability of the system will increase.

2.2.4 Trend Systems

The concept of a trend following system is to open a position in the direction of the current trend, and then keep the position open until the trend reverses. Usually trend systems are constructed

from a pair of moving averages that produce trading signals when they cross over each other. One moving average is faster (fewer days in the length of the average) than the other, so it hugs the price movement more closely. The other average is slower (more days, more length), and so takes a different "path" through the price chart than the fast average does.

A typical set of trading rules is like this: Buy when the fast average crosses above the slow average, and sell when the fast average crosses below the slow average. (If you were trading both long and short sides with the system, you would open a short position here too, and reverse it back to long at the next crossover point.)

Trend following systems can generate huge profits on long trends that run for months or years, if the system doesn't exit prematurely on price pullbacks that force the moving averages to cross and exit the trade. There is somewhat of an art to choosing the best types of averages for a particular market—simple, exponential, or weighted—and lengths of averages, to allow enough pullback room so your position is not taken out on small price moves in a longer trend.

The biggest problems with trend systems is that they also give up a significant amount of money at tops and bottoms while waiting for the averages to cross over, and they can whipsaw you into heavy losses in a flat, range-trading channel kind of market as they force you into unprofitable trades when there is not enough price movement between crossover points.

2.2.5 Anti-Trend Systems

The concept of an anti-trend system is to open a position *against* the direction of the trend, in the expectation that the trend will reverse enough to make the position profitable. A key point of this concept is that only *short term* trends are considered, because short term trends often do reverse, and short term anti-trend trades often do earn a profit. Of course, if the trend doesn't reverse soon enough, or if it continues too far in the original trend direction, the position is closed at a loss.

Many different technical mechanisms can be used to construct anti-trend systems. But generally speaking, a moving average of some kind is used to represent the center of the trading channel. This might be an 18 or 30 day moving average, for example. In addition, two other lines, bands, or boundaries are used to represent the outer range of the average trading channel. When prices move far enough away from the average toward one of the outer channel boundaries, a trade signal is triggered.

A typical set of trading rules might be: Buy when prices trend down below the moving average by more than 1 standard deviation of the normal trading range. Sell when prices return to the moving average, or when a profit target is reached. Also sell if the trend continues on too far in the original direction, or if the trade is still open 1 or 2 bars later.

Anti-trend systems are good for flat "sideways" markets that have enough movement in them to make anti-trend trades worthwhile. But be aware, anti-trend systems are susceptible to real trends that keep on going, forcing you to close positions at a loss.

2.2.6 Gap Closing Systems

The concept of a gap closing system is that gapping prices will retreat back to close the gap sooner or later. By opening a position against the movement that caused the gap, you can make a profit when prices return to fill the gap. Both overnight and intraday gaps are tradable.

A typical set of trading rules might be: Buy when prices gap down more than 2% below the previous bar close. Sell when prices rise to the previous close, or sell at the end of the trading day. Additional confirmation conditions might be to require the previous day to be a down day, or a down day for both the market and the stock.

Gap closing systems are very popular with many kinds of traders. That's because traders can make profits on gaps that are caused by news events or strong (but short term) price pressures. However, gaps can lose traders money too, if gaps are caused by legitimate price pressures that force prices to continue on in the original trend. Worse yet, because gap pressures are strong, prices can move fast against you and generate losses for you quite quickly if you're not careful.

2.2.7 Spread Trading Systems

The concept of a spread trading system is to trade the spread between any two things that normally travel together. When the two things get too far away from their normal relationship, you open a position that anticipates their return to normal conditions. Spread trading systems usually have a structure that includes a spread between two instruments or price series.

For example, you might make a trade that says that the Dow Jones index normally trends in the same direction with the S&P500 index, or that two currencies from adjacent countries travel closely together. If the two get too far apart, you would buy one and sell the other, anticipating a return to normal separation distance at a later time (hopefully soon).

A typical set of trading rules might be: Buy the Nasdaq 100 (QQQQ) and sell the S&P500 (SPY) when the ratio between the QQQQ and SPY is more than 1.5 standard deviations below the normal 10-day moving average value of the ratio, and if the QQQQ has had a big -2% down day the day before. Close the position when the ratio moves back to the 10-day moving average.

To be clear, 1.5 standard deviations equates to 90% of a Normal Distribution, so a move that is greater than 1.5 standard deviations from the mean is (in theory) supposed to be greater than 90% of all the moves that the ratio has ever made. So that kind of a move doesn't happen very often.

Spread trading systems are popular. You can spread trade (or ratio trade) any two things that are correlated closely enough to travel together a large percentage of the time. For example, two closely competitive stocks, two closely related currencies, or two closely related indexes can be used to trade this kind of strategy.

Notice that to neutrally trade a spread between two things, you must buy one and sell the other simultaneously. That's the only way to isolate the spread between two moving price streams. Then if you enter the trade because the spread is too wide, you don't care if both prices rise or fall while you're waiting for the spread to close. Gains or losses in one instrument will be almost perfectly offset by corresponding losses or gains in the other instrument. All you care about is whether the spread between them gets narrower or wider.

If you want to take a ratio between two price streams, and then trade standard deviations on the ratio using only one instrument, that's a volatility trade, which is not neutral to rising or falling prices in the two underlying instruments. Volatility trades are discussed below.

2.2.8 Volatility Systems

The concept of a volatility system is to trade the amount of volatility that occurs in a single series of instrument prices. When you trade volatility, you're actually trading the volatility of a thing against itself (usually against a moving average of itself).

One example of trading volatility is to trade the Nasdaq 100 (QQQQ) prices against a moving average of those prices. Another example—that was made famous by the Long Term Capital Management story—is to trade the implied volatility in equity options prices against the historical averages of the volatility. (The market was pricing in roughly 20% implied volatility, but the historical average was more like 15%, so LTCM shorted option prices, anticipating closing their positions when volatility—and therefore option prices—returned to the historical average. The strategy was sound, but during their trading timeframe, the volatility kept getting worse instead of better, eventually losing $4B in 4 weeks, and forcing them to close their company.)

A typical set of trading rules might be: Buy QQQQ when the price drops more than 1.5 standard deviations below the 10-day moving average of QQQQ. Sell QQQQ when the prices return back to the average, when a profit target has been reached, when a stop loss has been reached, or when a month of trading days has passed. (Remember that 1.5 standard deviations equates to 90% of a normal distribution, so a move that is greater than 1.5 standard deviations from the mean is supposed to be greater than 90% of all the volatility price moves that QQQQ has ever made.)

Notice that while the trade above is in effect, profits and losses are subject to rising and falling prices of QQQQ. The current QQQQ price might be 2 or 3 standard deviations below the moving average of QQQQ (enabling you to enter the trade long on QQQQ), but you will still lose money if QQQQ keeps dropping like a stone for a few days before the moving average catches up and the current price moves back to within 1.5 standard deviations of the moving average.

Volatility systems are popular with many traders, because tradable volatility is always present in the market somewhere. It works best with instruments or series that have a very low probability of maintaining high volatility for long periods of time (or worse yet, of increasing volatility so that 2.0 standard deviation events (>94% of all moves) are much more common than usual.)

Volatility trades do not work well with series that have "fat tails" on their distribution curves, because event distributions that have fat tails have many more events out there on the tails of the distribution curve (>1.5 or >2.0 standard deviations from the mean) than theory would predict. And since fat tails can cause losses (massive in the case of LTCM), you should try to allow for fat tails as much as is feasible when you're trading volatility.

2.2.9 Pattern Matching Systems

The concept of pattern matching systems is to recognize a particular bar pattern in market data and then make a trade that anticipates a future (profitable) situation that is suggested by the pattern. For example, a pattern might try to recognize double bottom chart patterns, or head and shoulder chart patterns, or other technical chart patterns (with or without the help of technical indicators such as moving averages, RSI indexes, or oscillators of various kinds).

A typical set of trading rules might be: After a trend upward of 15%, if you see a double-top chart formation, sell after the trailing edge of the double top breaks through the price level of the middle valley in the pattern.

These kinds of systems can be profitable (1) if the market serves up the desired chart patterns with enough frequency, (2) if the system can spot the pattern with enough accuracy, (3) if the pattern accurately predicts a future situation that is tradable, and (4) if the system can actually execute the trade profitably. As you might guess, this is a tough game to win.

2.2.10 Other Types of Systems

There are many other types of trading systems that were not discussed above. For example, inter-market systems trade instruments from multiple markets against each other, using either positive or negative price correlations or movements to enter, size, and exit trades. Cause-and-effect systems use movements in one instrument (e.g. bond prices) to anticipate movements in other instruments (e.g. S&P500 stock indexes) that are supposedly related by weak or strong cause-effect relationships. These systems are all beyond the scope of this introductory document.

2.3 Money and Exposure Management

This section talks about how you can limit the damage that your trading system can do to your equity curve (recall that *you* are a big part of the potential damage sources). Essentially, there are only two ways to limit risk and damage—restrict the amount of equity used (per trade, per sector, or per portfolio), and diversify among uncorrelated investments.

You might restrict the amount of equity used in *new* trading positions, so that when a losing trade is made, the loss on that trade is limited to the amount of the original equity put into the trade. A variation on limiting equity loss for a particular trade is to place a stop loss order on the trade, to stop losses from growing beyond a particular limit.

You might also restrict the amount of equity used in *existing* positions that get too big, especially when the market value of positions changes enough to violate a risk limitation rule. Such equity restrictions would initiate trades to reduce or correct the equity position to bring it back within limits. For example, if your technology investments increase in value so much that they exceed your rule of “no more than 20% invested in technology”, then your strategy would initiate trades to sell some technology investments to bring the portfolio total within limits.

Finally, you might diversify the use of equity in uncorrelated ways, so that not all your trades or positions are likely to lose at the same time. With reasonable diversification in place, it is more likely that you should have some winners to offset the losers, thereby reducing equity volatility.

2.3.1 Risk and Money Management

Money management is usually seen as the activity of sizing trades according to various monetary limits. For example, a typical money management rule might be “use 2% of equity per trade.”

Money management tends to operate at the scope of individual trades, rather than at higher levels that contemplate multiple instruments, multiple strategies, or multiple trading systems.

There are many ways to limit equity used—to name a few—restrict the equity used per trade, per instrument, per asset class, per strategy, per market, per portfolio, or per trading system. The main idea is to compartmentalize your equity, so that a single losing event in a single compartment

(trade, instrument, strategy, or whatever) can only destroy the equity in its own compartment, instead of destroying the equity in all compartments.

Money manager software components can perform risk management calculations on their own, in order to calculate the size of pending trades. But it is also quite common for Money components to call risk manager helper components to do the risk calculations, specify trade size limits, or to watch existing positions and set stop loss orders when position values go outside of limits. Risk calculations can look at whatever information is needed for the desired calculation—trade conditions, portfolio values, and rules of all kinds.

2.3.2 Exposure Management

Exposure management is similar to money management, but it operates at a higher level instead of at the trade sizing level. Exposure management looks at overall portfolios, to see if all existing portfolio positions are within exposure management rules and limits. For example, a typical exposure management rule might say "Limit technology to 20% of portfolio equity maximum."

For example, if portfolio exposure is too great in a particular area such as technology, an exposure manager would at least block incoming trades that sought to increase portfolio exposure to technology stocks even more. In addition, the exposure manager might also initiate trades to *reduce* (correct) the exposure to technology in order to bring it back within the exposure limits.

Exposure manager software components can perform risk management calculations in order to carry out their responsibilities. The risk calculations can look at whatever information is needed for the desired calculation—trade conditions, portfolio values, and rules of all kinds. The key point of this paragraph is that risk calculations support the activity of exposure management. That is, mere risk calculations are not a separate "risk management" function by themselves.

2.3.3 Diversification

Once you have split your equity into compartments, you should try to diversify the compartments among *uncorrelated or negatively correlated* trades, instruments, strategies, and systems. Diversification is based on the idea that there is a lower probability of simultaneous losing events in all of your *uncorrelated or negatively correlated* compartments of equity. Diversification (like money and exposure management) cannot change risk factors, but it can spread them around into separate compartments to reduce the probability of simultaneous failure in all compartments.

It is important to try to diversify using things that are uncorrelated (or at least, not highly correlated), because you want them to behave as independently as possible. That way, they are less likely to both go up or down at the same time.

Here are some poor examples of diversification, because the elements in these pairs normally travel together (high positive correlation): stocks of two disk drive manufacturers; two trend following systems with similar behavior; futures instruments for two closely related currencies; or exchange traded funds for the Dow Jones (DIA), the S&P500 (SPY), and the Nasdaq (QQQQ).

Here are some better pairs that show lower (or negative) correlation: a real estate stock and an industrial stock (real estate usually has a low correlation with the stock market); gold and bonds (when inflation goes up, gold goes up and bonds go down); a high-tech growth stock and a defensive "refrigerator" stock (when the economy goes down, the growth stock goes down, but

people still have to buy food, so the defensive stock goes up); a trend following system and a volatility trading system (one catches trends, one catches sideways volatility).

For example, if you diversify your equity into a trend following system compartment and a volatility trading system compartment, the odds are more favorable that your equity growth curve will be less volatile. This is because the trend system is more likely to earn money in a trending market, and the volatility system is more likely to earn money in a choppy market. So the systems will tend to offset each other, reducing volatility in your equity curve.

Pay close attention to money and exposure management, because trading systems are not perfect. Compartmentalize and diversify your equity for better safety and smoother performance.

2.3.4 Stop Loss Orders

It is important to consider using stop loss orders on all your automated trades, to prevent large losses in cases where the trade goes against you. Although some kinds of trades will automatically limit your maximum losses to the original trade equity (e.g. long stocks, long calls and puts), many kinds of trades can expose you to theoretically unlimited losses (e.g. short stocks, uncovered short calls, futures).

So to avoid losing the entire equity in a trade, you should consider entering stop loss orders as soon as you open a new position. You will see later that the event handler OnPositionOpened is a good place to automatically enter stop loss orders, because that event is fired whenever a new position is actually opened after your strategy gets a confirmation from your broker.

2.4 General Strategy Pitfalls

The very first pitfall to avoid is to use bad data for development and back testing (“Garbage in, garbage out.”) So try to use reasonable data. Hopefully your data will be good, and your strategy will continue to work when you start paper trading (with live market data).

Another common pitfall is to over optimize a system to fit a particular set of historical market data, so much so that the system will not work well on new live market data. This condition is called “over fitting” or “over optimizing”. One way to counteract over fitting is to limit the number of tunable optimization parameters in your system, and to deliberately choose reasonable values manually instead of optimizing them to multiple decimal places.

A second way to avoid over optimize a strategy is to use “out of sample” data to validate your strategy after you develop the strategy using “in sample” data. The main idea here is to develop your strategy on one set of historical data, and then validate its behavior on an equivalent set of historical data that the strategy has never seen before. If the strategy still behaves well, it is *not* because you have tuned the strategy to the out of sample data, because the strategy was not developed using the out of sample data.

Another common pitfall is to build a brittle system—one whose performance “breaks” (varies widely in profitability) with minor changes in system parameter values. To detect brittle systems, vary each system parameter during testing to see how sensitive system performance is to minor variations in parameter values. If the system is brittle, you should try to modify the system to be less brittle and more robust (insensitive to minor parameter changes). If you don’t, system performance on new market data is likely to “break” easily, and become unprofitable.

Still another pitfall is to build complex systems that have too many trading rules, too many parameters to coordinate and control, or too many sensitivities to unpredictable market price variations. Systems like this can exclude (or execute) too many good (or bad) trades, leave trades earlier or later than desired, and can produce wild swings in equity because they make too many unprofitable trades. Many experienced system designers have the same advice—keep your systems as simple to maximize your chance of building a workable, tradable, robust system.

Yet another pitfall is to not consider all transaction costs when back testing. Your strategy might earn a small profit on each trade, but it is possible that the profits will be eaten up (and more) by price slippages, commissions, and other account costs such as margin interest costs. So be sure to model transaction costs during your back tests so that you have an idea of whether they are significant or not for your particular strategy. Paper trading your strategy with a live trading account can help to expose some of these "hidden" costs for you.

A final pitfall is to think that your strategy should behave very well in all market conditions—in strong trending, strong ranging, and high and low volatility markets. It is extremely difficult to build a computerized strategy that is stable and reasonably profitable in all kinds of markets and all kinds of market conditions.

2.5 Some Cautions on Assumptions

Now that we've talked about some general system design issues, it seems useful to caution readers against some intuitive assumptions about trading systems that are easy to make. To counteract these intuitive assumptions, we mention a few of them here explicitly.

First, for trading systems in a real market, selling short is not the exact opposite of buying long. Some strategies just don't work well on the short side, even though they work fine on the long side. Perhaps it's because of an upward bias in markets, or more bullish market players than bearish market players. No one knows why for sure. But be aware, you cannot always flip a long strategy to the short side and expect to make comparable profits.

Second, what works for one kind of futures market doesn't always work for stocks, bonds, indexes, or all other instruments (and vice versa). Technical analysis purists might argue that a price bar series is no more than a series of numbers, and that a good trading system will work on any series. And certainly much experience indicates that some trading systems will in fact work well on 10 or 20 different commodities or index futures markets, even including stock indexes. And even some stock trading systems will work on futures markets. But be aware, you won't be able to make this claim for *your* system unless you test it thoroughly.

Third, miscellaneous financial charges such as commissions, bid/ask spreads, liquidity effects, implied volatilities effects (for options), and trading slippage can all add up to significant amounts of money. Especially if your trading system has a low expected value per trade, these kinds of financial charges can make the difference between a profitable, tradable system and one that loses money. So be aware, your back testing runs should provide enough allowance for commissions, bid/ask spreads, and slippage effects to make the testing (and therefore the results) as realistic as feasible. Remember that when you test your system, you're trying to increase the depth of belief in your system—so knowing that you have not allowed for slippage costs in your results will act to reduce your belief in your system when you will need it the most (probably in your worst time of draw down need).

3 OpenQuant Strategy Structure

This section explains the code structure and general operation of a simple strategy project. The section talks about a complete code example, typical strategy events, and how to place buy and sell orders from within a strategy framework.

3.1 Section Outline

In what follows, we show a complete working strategy code example. We'll talk about the code structure of the component—code sections such as declarations, initialization, event handlers, helper code—and what general kinds of code go in those sections. We're interested in structure in at this point, not in the code details of this particular strategy.

Then we'll talk about common events, and where and why you should use them. In the process of explaining these events, you will get a feel for how the code works to implement the strategy.

Finally, we'll talk a bit about how to place orders by creating order objects and sending them directly to the execution provider (broker).

3.2 Using Daily Bars for Writing Convenience

For easy understanding, many examples in this document are presented using daily bars, because it makes intuitive sense that OnBarOpen could correspond to the idea of "on market open at the beginning of the day." This correspondence is easier for novice traders to understand.

Using daily bars also makes it easier to say things like "yesterday's close" and "today's high" instead of "the close of the previous bar" or "the high value of the current bar." So fewer words and shorter sentences are required if daily bars can be used to help express the ideas.

But using daily bars for writing convenience does not limit the OpenQuant framework in any way. Be assured, the OpenQuant system is a real time system that is intended for trading bars of any size, all the way from fast tick data to slower weekly bars.

3.3 A Complete Code Example

To give you the big picture first, here is a typical strategy code example that implements a simple gap closing strategy. It opens a new position when a downward gap in prices is formed, and closes the position when the gap closes or at the end of the trading day, whichever comes first.

Every morning at market open (in OnBarOpen), this strategy looks for a downward gap size of 2% between yesterday's close and today's open. When the strategy spots such a gap, it buys 100 shares long to open a new position. If the new position is successfully opened, the strategy automatically enters a limit order (in OnPositionOpened) to sell if the stock price recovers to the level of yesterday's close during the trading day (thereby closing the gap). If today's bar arrives (in OnBar) and the open position still exists (the gap was not closed), the limit order is cancelled, and a market order is issued to close the position.

Like many C# code files, the following code file is organized into four major sections. (We identify these four parts with big comments in this example to emphasize the code structure for novice programmers.) The four sections are:

- Namespace references ("using…"),
- Class / variable / parameter declarations, and
- Event handlers (OnBarOpen, OnBar, OnPositionOpened, etc)
- Helper classes and methods (none in this example)

```
//
// Part 1 NAMESPACE REFERENCES
//

// OpenQuant namespace reference
using OpenQuant.API;

//
// Part 2 CLASS HEADERS, VARIABLE DECLARATIONS, INITIALIZATION
//

public class
Tech1_System1_ATS : Strategy {

// class variable declarations
private double      prevClose;              // updated in OnBar
private double      desired_gap = 0.02;     // gap size of 2 percent
private double      qty = 100;              // n shares to order
private SingleOrder sellOrder;

// parameter declarations (these show up in the IDE properties panel)
[Parameter ("Quantity")]
public double Qty {
    get { return qty; }
    set { qty = value; }
}
[Parameter ("Percent")]
public double Percent {
    get { return percent; }
    set { percent = value; }
}

// dummy init routine to show you where it goes in the file
// (it is not required for this example strategy)
public override void
Init () {
    // how to draw a simple moving average on the price chart
    // sma = new SMA (Bar, smaLength);
    // sma.Color = Color.Blue;
    // Draw (sma, 0);
}


//
// Part 3 EVENT HANDLERS (most code goes here for simple strategies)
//

// event handlers for buying and selling
public override void
OnBarOpen (Bar bar) {
```

```
    // calculate the size of the gap down since last bar
    double gap_size = (prevClose - bar.Open) / prevClose;
    // issue a buy order if the gap is big enough
    if (gap_size > desired_size) {
        MarketOrder order = MarketOrder (Side.Buy, qty);
        order.Text = "Tech1_System1 - Buy";
        order.Send ();
    }
}

public override void
OnPositionOpened () {
    // issue a stop limit order whenever a position is opened
    sellOrder = LimitOrder (Side.Sell, qty, prevClose);
    sellOrder.Text = "Tech1_System1 - Sell";
    sellOrder.Send ();
}

public override void
OnBar (Bar bar) {
    // update the prevClose to be today's close
    prevClose = bar.Close;
    // if we still have a position open, close the position
    if (HasPosition) {
        sellOrder.Cancel ();
        MarketOrder order = MarketOrder (Side.Sell, qty);
        order.Text = "Tech1_System1 - Sell";
        order.Send ();
    }
}

//
// Part 4 HELPER CODE
//

// but this strategy is so simple it doesn't need any helper functions
}
```

3.4 Defining Variables, Properties, and Parameters

As usual in normal C# code files, you define code variables at the top of the code file.

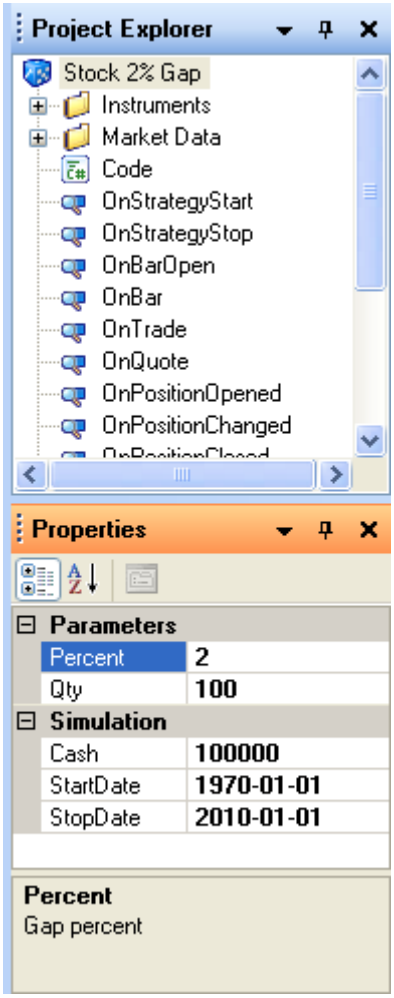
You can define strategy parameters that users can set through the Properties pane in the IDE. To do this, add the "[Parameter ("description")] attribute to your variable as shown in the code examples below. If you set a custom value for a parameter in the Properties panel, the IDE will save the specified value when you close the solution, so that when you reload the solution next time, your custom value will still be there.

Here is a code example with a corresponding screenshot, for the parameter "Percent (Gap Percent)". Notice how the variable name Percent and the description string "Gap Percent" show up in the Properties pane.

```
// parameter declarations (these show up in the IDE properties panel)
[Parameter ("Gap Percent")]
double Percent;

[Parameter ("Quantity to Trade")]
```

```
double Qty = 100;
```



3.5 Initialization

This section shows you how and where to declare variables and strategy parameters (as user-settable properties in the Properties panel of the IDE).

3.5.1 Defining Variables and Parameters

As usual in C# code files, you can define code variables at the top of files, as shown below.

```
// class variable declarations
private double      prevClose;       // previous bar close
private SingleOrder sellOrder;

// parameter declarations (these show up in the IDE properties panel)
[Parameter ("Quantity to trade")]
double Qty = 100;

[Parameter ("Gap Percent")]
```

```
double Percent = 2;
```

3.5.2 Grouping Parameters into Property Pane Groups

If you want to group your strategy parameters into several sections in the Properties pane, you can add a second parameter to the Parameter attribute, as shown below. This example shows how to put the first parameter into one group, and the other parameter into a second group.

```
[Parameter ("Gap Percent", “Group1”)]
double Percent = 2;

[Parameter ("Gap Percent for Group 2", “Group2”)]
double Group2_Percent = 2;
```

3.5.3 Optimizing Parameters

The IDE can optimize the values of variables that have been coded as parameters as shown above. Optimization is useful for determining the approximate range of values that generate the most profit for your strategy. But you should be careful not to “over fit” parameters to any one set of market data—this practice can lead to brittle trading systems that fail on any other market data that is not like the data that was used for optimization.

One approach to optimization is to optimize your parameters, and then manually change them to some nearby “rounded” number that is not exactly the one picked by the optimizer. At least this way your strategy will not get to depend on optimized parameter values to three decimal places.

To code a parameter for optimization, add the Optimize attribute to the parameter, as shown below. Also specify a range of values to use for optimization—one value for the start of the range, one value for the end of the range, and a third value for increments within the range.

Here is a code fragment that shows how to set up a gap size for optimization. The default gap size in percent is 2, set when the variable is declared. The optimization range begins at 1, ends at 5, and increments by 0.5 percent each time. This means the optimizer will run your strategy 9 times (from 1.0 to 5.0, by 0.5) during optimization, and then choose the best value for you.

```
// optimization range for parameter is 1.0, 1.5, 2.0, 2.5,. . .,5.0
[OptimizationParameter (1, 5, 0.5)]
[Parameter ("Gap Percent", “Group1”)]
double Percent = 2;
```

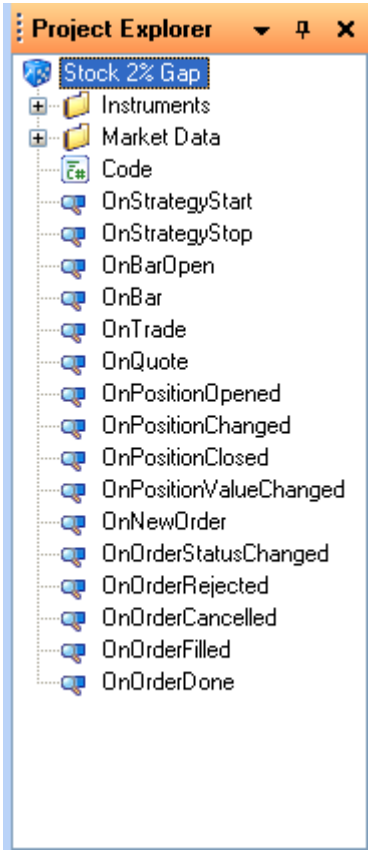
3.6 Common Strategy Events

Here is a very short list of useful event handlers that are called during a typical trading session, for both partitioned and integrated frameworks. This list is very minimal, and is intended only to give you a simple understanding of the main event handlers (and their typical uses) in a strategy. Of course you might end up using many more kinds of event handlers in other ways, depending on the complexity of your strategy.

- OnStrategyStart – called when strategy first starts up, before the first bar

- OnStrategyStop – called when the strategy shuts down, after the last bar
- OnBarOpen – called on leading edge of all bars (e.g. for buy at market open, daily bars)
- OnBar – called on trailing edge of all bars (e.g. for buy at market close on daily bars)
- OnBarSlice – called when all per-instrument bars have been emitted
- OnPositionOpened – called whenever a new position is confirmed opened
- OnPositionChanged – called whenever a position is increased or decreased
- OnValueChanged – called when portfolio value changes (which is with every quote!)

Here is a picture of more events that you can double click on in a strategy project. A template event handler will be created for you in the code file if one does not already exist. After double clicking, your cursor will be located at the desired event handler.



The main idea of the OpenQuant event model is to match how things actually happen in the real trading world. Events do a good job of modeling changing prices, changing orders, changing order statuses, changing positions, and changing position values.

Several kinds of events are fired in the OpenQuant framework during the course of buy-sell transactions involving market data providers, execution providers, and your strategies. Some (pre-trade) events are fired to model market data provider actions, some (trade) events are fired to model execution provider actions, and still other (post-trade) events are fired to model updates to portfolios and positions.

A strategy can subscribe and listen to all of these kinds of events, so that a strategy event handler (named OnXXX, where XXX is the event name) can be called to implement a strategy action

when the event fires. For example, when an event for a trade in MSFT is fired, the OpenQuant framework invokes the appropriate event handler in the appropriate component (that was created for the MSFT instrument).

Because the framework automatically invokes the event handler in the right component for the instrument, your strategy code can be much simpler—you don't have to worry about indexing all actions with if-else clauses that select the right code for the current instrument. Instead, the framework does that for you. You can usually write most of your code without referencing any particular instruments, because a separate instance of each component is created for each instrument that is traded by the strategy.

3.7 OnStrategyStart

The OnStrategyStart event handler is where your code creates new objects and attaches them to class variable declarations, and where you specify what series to draw on price charts. The OnStrategyStart method is called by the IDE before the strategy itself is started.

Here is example OnStrategyStart code that creates a new SMA (simple moving average) object for the default bar series, colors the moving average line Blue, and then draws the moving average line on the default price chart (pad 0). If you want a line on the volume chart, use pad 1. If you want it on a chart of its own, use pad 2 (or some other number greater than 1).

```
private SMA sma;                    // simple moving average variable
private smaLength;                  // length of moving average

public override void
OnStrategyStart () {
    // how to draw a simple moving average on the price chart
    sma = new SMA (Bars, smaLength);
    sma.Color = Color.Blue;

    Draw (sma, 0);
}
```

The SMA object constructor takes two parameters—a reference to the BarSeries (a time series of prices) that should be averaged, and the length of the moving average to calculate. Here, we use the name "Bars" to refer to the default bar series (object type BarSeries). If you want to add a color to the same code line, a second constructor that takes a color, as shown below.

```
    // or you can declare the color in the constructor, too
    sma = new SMA (Bars, smaLength, Color.Blue);
```

"Bars" is actually a property of the strategy base class; it returns the current instrument series that the component is working with, so you don't have to know the name of the instrument in order to request its bar series in your code. In some strategies, you might want to create your own BarSeries objects, in which case you would pass your own BarSeries to SMA constructors.

If you want to draw averages (or other technical indicator lines) on pads below the price chart, just increment the pad number to something greater than zero. Pad 0 is the price chart. Pad 1 is the volume chart. Pads 2+ can be whatever you want them to be.

3.7.1 OnBarOpen

OnBarOpen is generally fired before every bar that is received by your strategy. In Simulation Mode, OnBarOpen is fired exactly on bar boundaries, because simulation mode has total control over the input bars that are sent to your strategy. For example, if we are using half-hour bars, and the next bar period is 10:00am to 10:30am, the simulation engine will fire OnBarOpen at exactly 10:00am, and your strategy OnBarOpen event handler will be executed at 10:10am.

But in Live Trading mode, different behaviors are possible. To be precise, in Live Trading Mode OnBarOpen is fired whenever the opening price for the next bar is known. But since the opening price of a bar is calculated from the first Trade that actually occurs within the bar period, OnBarOpen cannot fire until the price of the first trade in the bar interval is known.

This means that if we are using half-hour bars, and the next bar is expected to last from 10:00am to 10:30am, OnBarOpen may not fire at precisely 10:00am. Instead, if the first trade within the bar period arrives at 10:20, that is the first point in time (within the bar period) where an opening price for the bar becomes known. So OnBarOpen will fire at 10:20, and your strategy event handler will receive the OnBarOpen event at 10:20.

The purpose of OnBarOpen is to give you a chance to do something on the leading edge of a bar, before the bar has been created. (In contrast, OnBar is fired on the trailing edge of a bar.)

For example, assuming that daily bars are being used, OnBarOpen is the place where you would put code that wants to buy or sell when the market first opens. This is because for daily bars, you only get one bar for the whole trading day. So you want your “buy at the open” action to take place on the leading edge of the day’s bar (using OnBarOpen), rather than on the trailing edge of the day’s bar (using OnBar).

If daily bars are not being used, then OnBarOpen just becomes a place where you can take action on the leading edge (the “open”) of the next bar that will be produced.

In many cases where buying at the open is not important, your code won’t need to use this event.

3.7.2 OnBar

OnBar is fired whenever the market data provider gives your strategy a new completed bar. Each bar contains OHLCV (open high low close volume) data for the past bar interval. Accordingly, the OnBar event fires on the trailing edge of the current bar. If you need to take action on the leading edge of the next bar, use the OnBarOpen event.

OnBar is a good place to put your routine strategy code that works with each new bar in a series of bars, such as for comparing the current bar prices with past bar prices, for calculating or updating technical indicators with new bar data, for calculating moving averages, for making trading decisions, and for issuing trading signals or orders.

3.7.3 OnPositionOpened

OnPositionOpened is fired whenever a new position is established as a result of a completed trade. The trade order must complete in order for a position to be opened. All positions must have non-zero position sizes in them. When a position size is reduced to zero (by closing the position

with appropriate trade orders), the position objects are destroyed. All that remains are logs of all trades that established (or tried to establish) the position, and trades that closed the position, if it was ever created.

Here is an example of how to automatically create a stop limit order when a long position is first created. The code below constructs a sell order object and sends it directly to an execution provider for trade execution. “Qty” means “quantity”, and is a positive integer that specifies the size of the trade (number of shares, or contracts). “prevClose” is the limit price for the order.

```
// create and issue a sell limit order in an integrated framework
public override void
OnPositionOpened () {
    sellOrder = LimitOrder (Side.Sell, qty, prevClose);
    sellOrder.Text = "Tech1_System1 - Sell";
    sellOrder.Send ();
}
```

3.7.4 OnValueChanged

OnValueChanged is fired whenever a new incoming trade price changes the value of a position. The price is taken from an actual trade (the “last price” on most trading consoles), not from the current bid or ask prices. Accordingly, this event will fire, and the value of your affected positions will change, many more times than OnBar events occur. This is because bars are usually created from many trades—that is, from all the trades that fall within a bar period.

OnValueChanged is a good place to adjust positions or trading orders, if they depend on the overall value of the portfolio components, rather than just on the price of the current bar. For example, if you wanted to issue trade orders to close all positions if the value of the portfolio falls below $10,000.00, this would be a good place to put that code.

3.7.5 OnPositionChanged

OnPositionChanged is fired whenever the size of a position is changed, one way or the other. For example, this event will fire every time a partial fill confirmation updates the position size.

OnPositionChanged is a good place to adjust the size of stop loss orders that must track partial fills. Every time a new partial fill confirmation comes in, you can add the confirmed quantity to the size of your outstanding stop loss order. This way, your stop loss order will more accurately reflect the actual fill status of your original order.

3.8 How to Place Orders

Here are short code fragments that show how to place typical trading orders to buy/sell long, and to sell/cover short. These fragments are just a combination of code examples that show the general syntax of orders. Sometimes multiple statements with slightly different syntaxes are used, just to show more than one possible way of doing things.

Here is a list of order types and actions: Buy, BuyLimit, BuyStop, BuyStopLimit, Sell, SellLimit, SellStop, SellStopLimit, SendLimitOrder, SendMarketOrder, SendStopLimitOrder,

SendStopOrder, StopLimitOrder, StopOrder, and TrailingStopOrder. Most of these orders have similar syntax (see the API reference for more details).

Here is some simple example code that shows how to buy and sell long, using orders sent directly to the execution provider. (To sell and cover short, reverse the sequence of the buy/sell orders).

```
// an order variable declaration
Order myOrder;

// a market order to sell
myOrder = MarketOrder (Side.Sell, qty);
myOrder.Text = "this text shows in the trade log";
myOrder.Send ();

// Order var type holds any order type object (long,shrt,mkt,limit)
myOrder = LimitOrder (Side.Sell, qty, prevClose);

// another syntax, showing more arguments
myOrder = LimitOrder (position.Instrument, Side.Sell, qty, prevClose);
myOrder.Text = "Tech1_System1 - Sell";
myOrder.Send ();

// how to cancel an order that has been sent previously
myOrder.Cancel ();
```

Order is a generic variable type that can be used to hold market, limit, stop, and stop limit order objects. Notice that the code examples above do not call the “new” operator to create a new order object. Instead, the code examples call methods of the strategy class. These methods (such as MarketOrder) create an order, insert default values for the current strategy Execution Provider and strategy portfolio, and then register the order with the strategy.

4 OpenQuant Example Strategies

This section provides documented code for several different kinds of trading strategies. Some are strategies that ship with the OpenQuant system. One is from Chande’s book *Beyond Technical Analysis*, and most others are from Altucher’s book *Trade Like a Hedge Fund.*

4.1 Summary of Strategy Techniques by Strategy

For your convenience, here is a cross reference between some useful strategy coding techniques and the strategy examples that follow. The associations below tell you which strategies use which strategy techniques to implement the strategy.

Techniques are usually only mentioned in the first strategy that uses the technique, even if the technique is used again by other strategies.

4.1.1 5% Panic Recovery

- Uses OnBar
- Define a strategy variables as a user-settable parameter
- Test for an open position (if HasPosition)

- Calculate a price 5% below the current bar
- Send a limit order
- Send a market order
- Cancel a buy order

4.1.2 Four Days Down and Long

- Count number of down days
- Discusses how bars and orders and times are related

4.1.3 Four Days Up and Short for 1% Profit

- Uses OnBar, OnPositionOpened
- Use of a profit target 1%
- Use limit order for a profit target

4.1.4 Breakout with 4% Entry Limit

- Uses OnBar, OnBarOpen (to catch leading edge of bar)
- Calculate 4% above previous closing price

4.1.5 Breakout with Multiple Exits

- Uses OnBar, OnPositionOpened, OnPositionClosed
- Uses OnStopExecuted
- Uses look back channel of 30 bars (find highest high in a series)
- Uses ATSStop as internal stop reminder
- Uses OCA One Cancels All order groups
- Exit after N bars have elapsed
- Exit after a profit target has been reached
- Exit when a trailing stop is executed

4.1.6 Bollinger Bands

- Uses Bollinger Bands for range trading
- Uses SMA simple moving average
- Uses Series.Contains(datetime) to check for enough bars

4.1.7 Bollinger Bands with Profit Target

- Uses penetration level parameter for strength of band crossing
- Exits on profit target
- Exit after 4 days

4.1.8 Simple Moving Average Crossover

- Uses two SMA moving averages
- Uses ECross objects to detect crossovers
- Exits on moving average crossover
- Exits on internal trailing stop *indicator* (ATSStop)
- Exits on external trailing stop order (resides on exchange servers)

4.1.9 Slow Turtle Trend Following

- Trades only on Monday mornings
- Uses very long averages (100 and 385 days)
- Reverses positions by trading 2 times current position size

4.1.10 Chande's 65sma_3cc Crossover Strategy

- Uses OnBar, OnBarOpened, OnStopExecuted, OnPositionChanged
- Uses 3 consecutive closes above SMA for confirmation of price move
- Uses ADX indicator (Average Directional Index)
- Uses RAVI indicator (Range Action Verification Index)
- Uses Position.Side to test for position type (long or short)
- Uses helper methods to manage exits, positions
- Exits on moving average crossover
- Exits after N bars
- Exits on internal trailing stop

4.1.11 Stock 2% Gap

- Enters when prices gap down 2%
- Exits (issues the order) on the very next bar

4.1.12 Stock Down, Stock 2% Gap

- Checks for a down day previous to the 2% gap down day

4.1.13 QQQQ Gap Down 0.5%, Stock Down, Stock Gap 5%

- Works with two instruments at once
- Coordinates several entry conditions among two instruments

4.1.14 QQQQ Gap 0.5%, Stock Down, Stock Gap 5%, Hold Overnight

- Holds a position overnight (one more bar, for daily bars)

4.1.15 QQQ Crash, QQQQ Trade

- Uses Bollinger bands
- Catches drastic price moves (1.5 std dev) using a short moving average (10 days)

4.1.16 QQQ Crash, Stock Trade

- Uses multiple instruments simultaneously
- Trades a volatile index stock when the index itself crashes

4.1.17 Bollinger Bands with 5-Minute Bars

- Uses OnTrade to look at trade data during bar formation
- Uses OnPositionValueChanged to monitor profit for profit target
- Uses OnPositionOpened to SetStop to set a time interval stop (not a price stop)
- Uses OnStopExecuted to set an exit flag after N minutes have elapsed
- Intended for quick intraday trades of 5 or 10 minutes in length

- Enters on a 3% drop
- Exits on a profit target of 1%
- Exits after 2 bars have passed
- Uses flags to control trade entry and exit

4.2 Pattern Matching Strategies

A pattern matching strategy is one that looks for special patterns in the incoming market data, and tries to make profitable trades on those special patterns. We start our strategy examples with pattern matching strategies, because they provide a gentle introduction to syntax.

4.2.1 5% Down-In-One-Day Panic Recovery

This first strategy is really simple, so it has been placed first in the document. The concept behind this strategy is that sometimes the market panics over some news, and hammers down a stock price for a short time. After a few minutes or hours, people recover from their panic, realize that the stock price is too low, and buy long to bid the price back up.

At the end of each trading day (we assume daily bars here, like Altucher did in his book), this strategy receives the daily bar in the OnBar event handler. If no position is open, the strategy uses today's closing price to calculate a new limit price that is 5% lower than today's close, and issues a limit buy order to be executed at market open the following day.

In contrast, if a position is open at the end of the day, it means that a 5% gap happened sometime during today, and so the strategy immediately closes the position with a market order (which will be executed at market open the next morning). The thinking here is that the panic prices would have been corrected upward during the day to something more reasonable.

Notice how the timing of strategy decisions and actions is affected by when the bars arrive, and when the orders are executed. With daily bars, the strategy receives the daily bar at the end of each trading day, in the OnBar event handler. The OnBar event handler is executed on the trailing edge of each bar, at time that corresponds to the end of the trading day for daily bars. If you want to do something on the leading edge of a bar—meaning at the market open before the daily bar is constructed—then you should put your code in an OnBarOpen event handler.

```
// 5% percent down in one day panic recovery
using OpenQuant.API;

public class
MyStrategy : Strategy {

[Parameter ("Buy when asset prices drop more than this in one day")]
double Percent = 5;

[Parameter ("Order quantity (number of contracts to trade)")]
double Qty = 100;

Order buyOrder;

public override void
OnBar (Bar bar) {
    // if we do not have a position, update the limit buy order to be
```

```
        // 5% above today's close
        if (!HasPosition) {
            // cancel the old limit order (it's out of date now)
            if (buyOrder != null)
                buyOrder.Cancel ();

            // issue a new buy order at 5% below today's close this order
            // will execute tomorrow if price is matched
            double buy_price = bar.Close * (1 - (Percent / 100));
            buyOrder = BuyLimitOrder (Qty, buy_price, "Entry");
            buyOrder.Send ();
        }

        // else we opened a position today using our limit order from
        // yesterday, so now close the position at the end of today.
        // We expect that such a big drop was freaky, and that prices
        // recovered during the day. If not, this order stops further losses.
        else
            Sell (Qty, "Exit");
    }
    }
```

4.2.2 Four Down Days and Long

The concept of this strategy is to open a long position after a major market index has had 4 down days in a row. The theory is that 4 days of market momentum in a major market index is hard to maintain, and that an up day is soon to follow. You might have to wait a day or two for it to show up, but it will show up eventually.

This implementation buys at the market open on day 5 after 4 down days, and doesn't wait around for multiple days for an up day. It just automatically issues a market sell order at the end of day 6 (when day 6 bar is received). The sell order will actually be executed by the exchange when the market opens on day 7.

Notice how the timing of strategy decisions and actions is affected by when the bars arrive, and when the orders are executed. With daily bars, the strategy receives the daily bar at the end of each trading day, in the OnBar event handler. The OnBar event handler is executed on the trailing edge of each bar, at time that corresponds to the end of the trading day for daily bars. If you want to do something on the leading edge of a bar—meaning at the market open before the daily bar is constructed—then you should put your code in an OnBarOpen event handler.

```
// four down days and long
using OpenQuant.API;

public class
MyStrategy : Strategy {

[Parameter ("Order quantity (number of contracts to trade)")]
double Qty          = 100;

[Parameter ("Number of consecutive down closes")]
int ConsClosesCount = 4;

int    count;
double prevClose;
```

```
public override void
OnStrategyStart () {
    prevClose = -1;
    count = 0;
}

public override void
OnBar (Bar bar) {
    // we need at least one bar to go by before we do anything
    if (prevClose != -1) {
        // if we don't have a position open,
        // increment the count of down days (or reset it to zero)
        if (!HasPosition) {
            if (prevClose > bar.Close)
                count++;
            else
                count = 0;

            // if this is the fourth down day, issue a market
            // order to open long tomorrow morning, on day 5
            if (count == ConsClosesCount)
                Buy (Qty, "Entry");
        }

        // else if we have a position open, close it now
        // (today is the day after the trade was entered),
        // so "now" is actually end of day 6, and the trade
        // will be executed by the market on open day 7.
        else
            Sell (Qty, "Exit");
    }

    // today's close now becomes the previous close for
    // the down day calculation
    prevClose = bar.Close;
}
}
```

4.2.3 Four Up Days and Short for 1% Profit

This strategy is a variation on the Four Down Days strategy. A plain Four Up Days (flip) of the down days strategy did not produce good results for Altucher, reinforcing his statement that going short is not the exact opposite of going long. So he provides this “up day” variation, which uses up days instead of down days, but it also enforces two extra conditions.

The first condition is that the fourth day up has to be a huge one (2% or more). After 3 days up already, going up 2% on the fourth day would be quite impressive. Theoretically, such a jump on the fourth day would be a strong invitation to long stock holders to take some profits off the table the next morning, thereby driving down the price. So this strategy issues a short order on the close of the fourth day, for execution at the open of the fifth day.

The second condition is that the strategy exits when it reaches a 1% profit target, and uses a limit order to implement the exit during trading day 5. If the position is still open at the end of day 5, the strategy issues a market order to close the position, for execution at market open on day 6. Keep in mind that this discussion assumes the use of daily bars (as the book author did).

```
// four up days and short
using OpenQuant.API;

public class
MyStrategy : Strategy {

[Parameter ("Order quantity (number of contracts to trade)")]
double Qty          = 100;

// exit with a profit target of 1%
[Parameter ("Profit Target")]
double ProfitTarget = 1;

// last day up must be a big 2% up day
[Parameter ("Up Move")]
double UpPercent    = 2;

// Number of consecutive down closes
[Parameter ("Number of consecutive down closes")]
int    ConsClosesCount = 4;

// count of up days
int    count;
double prevClose;

// orders
Order  buyOrder;
Order  sellOrder;

public override void
OnStrategyStart () {
    prevClose = -1;
    count = 0;
}

public override void
OnBar (Bar bar) {
    // we need to let a bar go by to capture the prev close
    if (prevClose != -1) {
        // if we don't have a position open, update the count
        // of up days, and try to enter a trade
        if (!HasPosition) {
            if (prevClose < bar.Close)
                count++;
            else
                count = 0;

            // if we have seen 4 up days, AND if the last day
            // up was 2% or more, then open a new position,
            // going short on the day's close
            if (count == ConsClosesCount) {
                if ((bar.Close - prevClose) / prevClose
                      >= UpPercent / 100) {
                    sellOrder = MarketOrder
                            (OrderSide.Sell, Qty, "Entry");
                    sellOrder.Send ();
                }
            }
        }

        // if we have a position open, cancel our previous
        // 1% profit target order, and close using a market order
```

```
        else {
            buyOrder.Cancel ();

            Buy (Qty, "Buy Cover");
        }
    }

    // now today's close becomes the previous close
    prevClose = bar.Close;
}

public override void
OnPositionOpened () {
    // when we open a position, immediately issue a limit order
    // for our 1% profit target
    double target_price = sellOrder.AvgPrice * (1 - ProfitTarget / 100);
    buyOrder = BuyLimitOrder (Qty, target_price, "Exit (Take Profit)");
    buyOrder.Send ();
}
}
```

4.3 Breakout Strategies

The concept of a breakout strategy is to monitor prices over a period of time (called the look back time, or the breakout channel length) so that you can determine a trading range for past prices. Then when a new price move suddenly “breaks out” of the usual trading channel, it could be the beginning of a big new price move upward.

So when a breakout strategy sees current prices break out of the usual trading channel, it opens a trading position and holds it until some exit criterion is satisfied. Many exit criteria are possible—you can exit after N bars go by, after a profit target has been reached, or after the trade goes against you and your stop loss order is executed.

4.3.1 Breakout with 4% Entry Limit

The rationale behind this simple breakout strategy is to take advantage of a big (4%) daily move upward that would likely squeeze all those short sellers who were hoping the stock would go down. This strategy thinks that moving 4% up in a single day is such a big move that it would scare all the short sellers into covering their positions by buying long, thereby driving the price up even further the next day. (Of course, contrarian traders would be thinking of going short the next day, to take advantage of a price recovery to lower levels.)

This strategy uses the OnBarOpen event handler, so that all trading events take place at the beginning of the day, at market open (on the leading edge of the daily bar).

As each day begins, the strategy calculates an entry price 4% higher than yesterday’s close, and issues a limit order to open a long position if that price is reached. The next morning, if a position exits (opened during yesterday’s trading), the strategy issues a market order to close the position.

```
// Breakout with 4 percent entry limit
using OpenQuant.API;

public class
```

```
MyStrategy : Strategy {

[Parameter ("Order quantity (number of contracts to trade)")]
int    Qty             = 100;

// enter trade on 4 breakoutPercent breakout
[Parameter ("Percent")]
double BreakoutPercent = 4;

// orders and trade quantity
Order  buyOrder;
// for calculating breakout price limit
double prevClose;

public override void
OnStrategyStart () {
    prevClose = -1;
}

public override void
OnBarOpen (Bar bar) {
    // we need to let the first bar go by before we can
    // calculate the breakout limit
    if (prevClose != -1) {
        // if we do not have a position, then cancel the
        // previous limit order (it is out of date)
        if (!HasPosition) {
            if (buyOrder != null)
                buyOrder.Cancel ();

            // now try to enter a trade by setting a limit order
            // to automatically buy in if the big 4% jump arrives.
            // This order will reside on the exchange servers, and
            // will execute during the day if the limit is triggered.
            double breakout_fraction = 1 + (BreakoutPercent / 100);
            double breakout_price = prevClose * breakout_fraction;
            buyOrder = BuyStopOrder (Qty, breakout_price, "Entry");
            buyOrder.Send ();
        }

        // if we have a position open, then close it now.
        // Now (which is the leading edge of today's daily bar)
        // is the start of the day after the trade was opened.
        else
            Sell (Qty, "Exit");
    }
}

public override void
OnBar (Bar bar) {
    // update the prevClose value for the breakout calculation
    prevClose = bar.Close;
}
}
```

4.3.2 Breakout with Multiple Exits

The following example implements a simple breakout strategy using an integrated framework. (As you will see, all the examples use the integrated framework because the code is so much easier to write.) By default, this strategy looks back 30 bars to detect breakouts, and exits when one of its three possible exit criteria is satisfied. It can exit after 10 bars have gone by. It can exit when a profit target has been reached. And it can exit when a trailing stop limit is reached.

Notice the use of a stop indicator, rather than a stop order, to exit the trade with a trailing stop. The stop is just a stop that is maintained by the strategy framework (not the broker). The trailing stop fires when an incoming trade price reaches the stop limit, and the OnStopExecuted event handler is called. The event handler issues a market order to close the position. This way, no stop order is ever issued to the broker (perhaps some brokers or exchanges don't take trailing stops).

```
// Breakout with Multiple Exits
using OpenQuant.API;

public class
MyStrategy : Strategy {

[Parameter ("Order quantity (number of contracts to trade)")]
double Qty = 100;

[Parameter ("Length")]
int Length = 30;

[Parameter ("Stop OCA Level", "OCA")]
double StopOCALevel = 0.98;

[Parameter ("Length", "OCA")]
double LimitOCALevel = 1.05;

[Parameter ("Stop Level", "Stop")]
double StopLevel = 0.05;

[Parameter ("Stop Type", "Stop")]
StopType StopType = StopType.Trailing;

[Parameter ("Stop Mode", "Stop")]
StopMode StopMode = StopMode.Percent;

[Parameter ("OCA Exit Enabled", "Exit")]
bool OCAExitEnabled = true;

[Parameter ("Time Exit Enabled", "Exit")]
bool TimeExitEnabled = false;

[Parameter ("Stop Exit Enabled", "Exit")]
bool StopExitEnabled = false;

[Parameter ("Bars to Exit", "Exit")]
int BarsToExit = 10;

Order limitOrder, stopOrder;

int OCACount      = 0;
int barCount;

bool entryEnabled = true;
```

```
double highestHigh;

public override void
OnBar (Bar bar) {
    // entry is disabled if we have an existing trade in play
    if (entryEnabled) {

        // don't trade until we accumulate enough bars in our series
        // for a proper look back of "length" bars into the past
        // channel that we are trying to break out of
        if (Bars.Count > Length)


            // if today's high is higher than all bars in the
            // lookback period of length "length" (30 by default)
            if (bar.High > highestHigh) {

                // then we are breaking out long, so issue a long
                // side market buy order to open a position
                Buy (Qty, "Entry");


                // if one cancels all exit method is desired, we
                // also issue a limit (profit target) order, and
                // a stop loss order in case the breakout fails.
                // The OCA exit method uses a real stop loss order.
                // The Stop exit method uses a stop indicator.
                // Use either the OCA or Stop method, not both at once.
                if (OCAExitEnabled) {

                      // create and send a profit limit order
                      limitOrder = SellLimitOrder (Qty,
                            LimitOCALevel * bar.Close,
                            "Limit OCA " + OCACount);
                      limitOrder.OCAGroup
                        = "OCA " + Instrument.Symbol + " " + OCACount;

                      // create and send a stop loss order
                      stopOrder = SellStopOrder
                            (Qty, StopOCALevel * bar.Close,
                              "Stop OCA " + OCACount);
                      stopOrder.OCAGroup
                        = "OCA " + Instrument.Symbol + " " + OCACount;

                    limitOrder.Send ();
                    stopOrder.Send ();
                      // bump the OCA count to make OCA groups unique
                    OCACount++;
                }
                // prevent further entries since we have entered already
                entryEnabled = false;
                barCount = 0;
            }
    }
    // if entry is disabled on this bar, we have an open position
    else {
        // increment bar count seen while position is open
        // this count is used for the bar count exit method
        barCount++;

        // if we want to exit the trade based on time, we use the
        // bar count as a proxy for elapsed time, and issue a market
```

```
            // sell order. Only one of OCA, Stop, or Time exit methods
            // should be enabled at the same time.
            if (TimeExitEnabled && barCount > BarsToExit) {
                Sell (Qty, "Time Exit");
                // also enable entries, since we closed our position
                entryEnabled = true;
            }
        }

        // update the breakout limit to watch for
        if (Bars.Count >= Length)
            highestHigh = Bars.HighestHigh (Length);
    }

    public override void
    OnPositionOpened () {
        // if we want to exit trades using the Stop method, set a
        // a trailing stop indicator when the position is
        // first opened. The stop indicator is not a stop loss
        // order that can be executed by a broker. Instead, the stop
        // just fires the OnStopExecuted event when it it triggered.
        if (StopExitEnabled)
            SetStop (StopLevel, StopType, StopMode);
    }

    public override void
    OnPositionClosed () {
        // when a position is closed, cancel the limit and stop
        // orders that might be associated with this position.
        // But only cancel if the order has not been filled or
        // not been cancelled already.
        if (OCAExitEnabled && !(limitOrder.IsFilled ||
                                limitOrder.IsCancelled))
            limitOrder.Cancel ();

        // allow entries once again, since our position is closed
        entryEnabled = true;
    }

    public override void
    OnStopExecuted (Stop stop) {
        // if our trailing stop indicator was executed,
        // issue a market sell order to close the position.
        Sell (Qty, "Stop Exit");
    }
    }
```

Of course, the breakout strategy shown above could be modified to use different channel lengths, different means of calculating the breakout channel limits (e.g. if prices went outside Bollinger bands, or if a particular volume was required on the breakout), and could use different means of maintaining or exiting the trade.

4.3.3 Confirmation Methods

When a potential trade signal is at hand, such as prices breaking out of a channel, some strategies prefer to “confirm” the breakout action by checking for additional conditions that must be satisfied before entering a trade. These conditions are not part of the breakout signal itself, but

instead are confirmations obtained from other sources that are different than the source of the breakout signal itself.

For example, suppose a breakout signal was defined as in the example above, such that a bar with a High > the highest High of the past 30 bars met the definition of a breakout signal. How could a strategy further verify that the breakout was a quality breakout by checking other conditions?

One possible confirmation could be to require that the current High exceed the highest High in the look back period by a particular amount. More of a gap could imply more probability of success.

Another method might be to require that the current bar Close be higher than the highest High. Yet another method might be to require 3 more consecutive closes that are each higher than the one before, where the first one is the breakout signal. This "3cc" method is used in a strategy example later in this document.

Still another method might be to check several other technical indicators, moving averages, or other sources of information to detect and filter out low quality breakouts. Many different kinds of confirmation methods are possible, limited only by your imagination.

4.4 Range Trading Strategies

When instrument prices trade sideways within a range of prices, we say that the prices are "in a trading range." We can model the range technically using trading *bands* or *envelopes* of various kinds to model the upper and lower limits of the price trading range.

Most band and envelope models use a central moving average to represent the center of the trading range. Obviously the moving average can drift upward or downward to some extent as it follows prices. If it drifts up or down too much, we would say that prices are trending rather than ranging, but that doesn't matter for the purposes of this discussion.

The outer bands are commonly calculated using various mixes of percentages of the moving average price, the standard deviation of the price series, or the highest high and lowest low of the past N bars in the series. Each of these methods has advantages and disadvantages.

As a general rule, long range trading strategies try to buy at the low end of the range, and sell at the middle (moving average line) or high end (upper band limit). Short range trading strategies try to do the opposite—sell high first, and then buy back when prices move back to the center line.

4.4.1 Bollinger Bands

Bollinger bands are calculated using the standard deviation of the difference between prices and a central moving average line. Since Bollinger bands use standard deviations, they move apart during volatile price times, and get narrower during quiet price movement times.

Since range trading is basically a volatility technique (you trade edges against a center line), it works well in sideways markets where your open trades are not exposed to very much trending price action. That is, in sideways markets, there is a lower probability that your trade will go against you. But be aware, if a breakout occurs against your range trade, you can lose lots of money if you don't have a stop loss in place.

The implementation shown here does not use stop losses, so it has no downside protection.

In the code that follows, the strategy tries to trade long by buying at the lower Bollinger band limit and selling when prices go up and reach the moving average line. The strategy constantly watches prices (bars), so it can update both the buy point and sell point as prices move around.

The strategy constantly updates the buy limit order at the broker while no position is open. When a buy occurs, the OnPositionOpened event handler is called to issue a sell limit order. From that point on, the strategy updates only the exit limit order, until the sell point is reached and the position is closed. Then the cycle repeats.

This strategy is effective in sideways markets and upward trending markets, if false breakouts and down trending markets don't go against you too often. Of course you should add a stop loss mechanism to this strategy for downside protection before using it in real life. Also pay close attention to transaction costs, because small price moves don't usually provide much profit.

```
// Bollinger Bands
using OpenQuant.API;
using OpenQuant.API.Indicators;

using System.Drawing;

public class
MyStrategy : Strategy {

[Parameter ("Order quantity (number of contracts to trade)")]
double Qty    = 100;

[Parameter ("Length of SMA")]
int SMALength = 20;

[Parameter ("Order of BBL")]
double BBLOrder = 2;

// indicators
BBL bbl;
SMA sma;

Order buyOrder;
Order sellOrder;

public override void
OnStrategyStart () {
    // set up the moving averages
    sma = new SMA (Bars, SMALength);
    sma.Color = Color.Yellow;
    Draw (sma, 0);
    // set up bollinger bands
    bbl = new BBL (Bars, SMALength, BBLOrder);
    bbl.Color = Color.Pink;
    Draw (bbl, 0);
}

public override void
OnBar (Bar bar) {
    // always a good practice to be sure a series contains
    // a bar for a particular date before you try to use it
    if (bbl.Contains (bar.DateTime)) {
```

```
        // We are always trying to buy at the lower Bollinger
        // limit, and sell when the price goes up to the
        // latest SMA value. So we are constantly
        // updating both the buy point and the sell point.

        // if we don't have an open position in this instrument,
        // update the buy point to the latest lower bbl limit
        if (!HasPosition) {
            if (buyOrder != null)
                buyOrder.Cancel ();
            buyOrder = BuyLimitOrder (Qty, bbl.Last, "Entry");
            buyOrder.Send ();
        }

        // else if we already have a position going, update
        // the sell point to follow the latest SMA value
        else
            UpdateExitLimit ();
    }
}

public override void
OnPositionOpened () {
    UpdateExitLimit ();
}

private void
UpdateExitLimit () {
    // cancel old exit point order, if it exists
    if (sellOrder != null)
        sellOrder.Cancel ();
    // Issue a new order with the latest SMA value. This is
    // kind of a "trailing SMA sell order" that follows the SMA.
    sellOrder = SellLimitOrder (Qty, sma.Last, "Exit");
    sellOrder.Send ();
}
}
```

4.4.2 Bollinger Bands with Profit Target

This example is like the basic Bollinger Band example above, but it adds three refinements. First, it demands that prices really punch through the lower Bollinger limit before buying. If 0 is the low limit, and 100 the high limit, this strategy buys only if prices are 20% of the band width below the lower limit. Second, it exits when a profit target of 15% is reached. Third, it exits after 4 days if the profit target has not been reached.

```
// Bollinger bands with Profit Target
using OpenQuant.API;
using OpenQuant.API.Indicators;

using System.Drawing;

public class
MyStrategy : Strategy {

[Parameter ("Order quantity (number of contracts to trade)")]
double Qty    = 100;
```

```
[Parameter ("Bollinger Band level to go long")]
double BLevel = -20;

[Parameter ("Percent profit target")]
double ProfitPercent = 15;

[Parameter ("Max number of bars, while position is active")]
int    MaxDuration = 4;

[Parameter ("Length of Bollinger Band")]
int    BLength = 10;

[Parameter ("Order of Bollinger Band")]
double BOrder = 1.5;

B   b;
int barsFromEntry = 0;
double exitPrice;
Order sellOrder;
Order buyOrder;

public override void
OnStrategyStart () {
    // set up bollinger bands
    BBL bbl = new BBL (Bars, BLength, BOrder, Color.Pink);
    Draw (bbl, 0);

    // set up a series for B (breakout force) value
    b = new B (Bars, BLength, BOrder, Color.Yellow);
    Draw (b, 2);
}

public override void
OnBar (Bar bar) {
    // it is a good practice to check if a series has
    // the date you are interested in before you try to use it
    if (b.Contains (bar)) {
        // if we don't have a position and prices are
        // below the lower band, open a long position
        if (!HasPosition) {
            if (b[bar.DateTime] * 100 <= BLevel) {
                buyOrder = BuyOrder (Qty, "Entry");
                buyOrder.Send ();
            }
        }
        else {
            barsFromEntry++;

            // if we _have_ reached the exit day (4 days after entry),
            // cancel the profit target sell order, and issue a
            // new market order to close the position now.
            if (barsFromEntry == MaxDuration) {
                barsFromEntry = 0;

                // cancel existing sell order if there is one
                if (sellOrder != null)
                    sellOrder.Cancel ();
                Sell (Qty, "Exit (Max Duration)");
            }
        }
    }
}
```

```
public override void
OnPositionOpened () {
    // when a position is opened, calculate profit target
    exitPrice = buyOrder.AvgPrice * (1 + ProfitPercent / 100);

    // cancel existing sell order if there is one
    if (sellOrder != null)
        sellOrder.Cancel ();

    // issue a new sell limit order at the profit target price
    sellOrder = SellLimitOrder (Qty, exitPrice, "Exit (Profit Target)");
    sellOrder.Send ();
}

public override void
OnPositionClosed () {
    barsFromEntry = 0;
}
}
```

4.5 Trend Following Strategies

Moving average strategies are probably the most well known of all computer trading strategies. Typically they are implemented with a goal of following a trend in price movement, so many moving average strategies are also trend following strategies. Notice that “trend following” is the goal of the strategy, and the implementation method is “moving average crossover.”

Of course moving average methods can be used with strategies whose goal is to NOT go with the trend. For example, maybe strategies want to use moving averages to go *against* the trend, or to use moving averages as confirmation signals, or exit-only signals. The main point here is that moving averages have many possible purposes beyond trend following.

4.5.1 Simple Moving Average Crossover

The following moving average crossover strategy ships with the OpenQuant system, and uses moving average crossovers to enter trades. This strategy (like the breakout strategy shown above) has three methods for exiting a trade. It can use the moving averages, the OCA One Cancels All method, or a trailing stop indicator method (which initiates a market order to close the position).

Notice the use of a trailing stop indicator, rather than a stop order, to exit the trade. The stop is just a stop that is maintained by the strategy framework (not the broker). The trailing stop fires when an incoming trade price reaches the stop limit, and the OnStopExecuted event handler is called. The event handler issues a market order to close the position. This way, no stop order is ever issued to the broker (perhaps some brokers or exchanges don’t take trailing stops).

```
// Simple Moving Average Crossover
using System;
using System.Drawing;

using OpenQuant.API;
using OpenQuant.API.Indicators;

public class
```

```
MyStrategy : Strategy {

[Parameter ("Length of SMA1", "SMA")]
int Length1 = 3;

[Parameter ("Length of SMA2", "SMA")]
int Length2 = 7;

[Parameter ("Color of SMA1", "SMA")]
Color Color1 = Color.Yellow;

[Parameter ("Color of SMA2", "SMA")]
Color Color2 = Color.LightBlue;

[Parameter ("Stop OCA Level", "OCA")]
double StopOCALevel = 0.98;

[Parameter ("Limit OCA Level", "OCA")]
double LimitOCALevel = 1.05;

[Parameter ("Stop Level", "Stop")]
double StopLevel = 0.05;

[Parameter ("Stop Type", "Stop")]
StopType StopType = StopType.Trailing;

[Parameter ("StopMode", "Stop")]
StopMode StopMode = StopMode.Percent;

[Parameter ("CrossoverExitEnabled", "Exit")]
bool CrossoverExitEnabled = true;

[Parameter ("OCA Exit Enabled", "Exit")]
bool OCAExitEnabled = true;

[Parameter ("Stop Exit Enabled", "Exit")]
bool StopExitEnabled = true;

// this strategy uses some of the same exit methods
// as the breakout strategy described earlier. Look
// there for additional documentation
// lengths and colors of the simple moving averages
SMA sma1;
SMA sma2;

// only one trade is allowed at a time
bool entryEnabled = true;

int OCACount = 0;

// for the orders used by this strategy
Order marketOrder,
    limitOrder,
    stopOrder;

[Parameter ("Order quantity (number of contracts to trade)")]
double Qty  = 100;

public override void
OnStrategyStart () {
    // set up the moving averages, based on closing prices
    sma1 = new SMA (Bars, Length1, Color1);
    sma2 = new SMA (Bars, Length2, Color2);
```

```
    // 0 means draw both averages on the price chart
    Draw (sma1, 0);
    Draw (sma2, 0);
}

public override void
```
OnBar (Bar bar) {
```
    // does the fast average cross over the slow average?
    // if so, time to buy long
    Cross cross = sma1.Crosses (sma2, bar);
    // we only allow one active position at a time
    if (entryEnabled) {
        // if price trend is moving upward, open a long
        // position using a market order, and send it in
        if (cross == Cross.Above) {
            marketOrder = MarketOrder (OrderSide.Buy, Qty, "Entry");
            marketOrder.Send ();
            // if one cancels all exit method is desired, we
            // also issue a limit (profit target) order, and
            // a stop loss order in case the breakout fails.
            // The OCA exit method uses a real stop loss order.
            // The Stop exit method uses a stop indicator.
            // Use either the OCA or Stop method, not both at once.
            if (OCAExitEnabled) {

                // create and send a profit limit order
                double profitTarget = LimitOCALevel * bar.Close;
                limitOrder = SellLimitOrder (Qty, profitTarget,
                    "Limit OCA " + OCACount);
                limitOrder.OCAGroup
                    = "OCA " + Instrument.Symbol + " " + OCACount;
                limitOrder.Send ();

                // create and send a stop loss order
                double lossTarget = StopOCALevel * bar.Close;
                stopOrder = SellStopOrder (Qty, lossTarget,
                    "Stop OCA " + OCACount);
                stopOrder.OCAGroup
                    = "OCA " + Instrument.Symbol + " " + OCACount;
                stopOrder.Send ();

                // bump OCA count to make OCA group strings unique
                OCACount++;
            }
            entryEnabled = false;
        }
    }
    // else if entry is disabled, we have an open position
    else {
        // if we are using the crossover exit, and if the fast
        // average just crossed below the slow average, issue a
        // market order to close the existing position
        if (CrossoverExitEnabled)
            if (cross == Cross.Below) {
                marketOrder
                    = MarketOrder (OrderSide.Sell, Qty,
                            "Crossover Exit");
                marketOrder.Send ();
            }
    }
}

public override void
```

```
OnPositionOpened () {
    // if we want to exit trades using the Stop method, set a
    // a trailing stop indicator when the position is
    // first opened. The stop indicator is not a stop loss
    // order that can be executed by a broker. Instead, the stop
    // just fires the OnStopExecuted event when it it triggered.
    if (StopExitEnabled)
        SetStop (StopLevel, StopType, StopMode);
}

public override void
OnPositionClosed () {
    // when a position is closed, cancel the limit and stop
    // orders that might be associated with this position.
    // But only cancel if the order has not been filled or
    // not been cancelled already.
    if (OCAExitEnabled && !(limitOrder.IsFilled
        || limitOrder.IsCancelled)) {
        limitOrder.Cancel ();
    }
    // allow entries once again, since our position is closed
    entryEnabled = true;
}

public override void
OnStopExecuted (Stop stop) {
    // if our trailing stop indicator was executed,
    // issue a market sell order to close the position.
    marketOrder = MarketOrder (OrderSide.Sell, Qty,
        "Stop Exit");
    marketOrder.Send ();
}
}
```

4.5.2 "Slow Turtle" Trend-Following

The original Turtle system was made famous because it was supposed to show that anyone could become a good commodities trader—all you needed was to follow some specific rules that implemented a good trading system.

The implementation shown below is based on Altucher's book, which uses its own choice of 22 *week* (100 day) and 55 *week* (385 day) moving averages. This is what he calls a "slow turtle" strategy that is intended to catch very long (and major) trends. He does this by using very long moving averages. The original Turtle system used shorter averages. (It is worth saying that there are many variations of the Turtle system on the Internet.)

There are three interesting things about this single-state (always in the market) implementation. It uses very long moving averages (100 and 385 days). It only trades on Monday mornings (in the OnBarOpen method). And it reverses existing positions by buying or selling a trade quantity that is twice the size of the existing position.

```
// Slow Turtle Trend Following
using OpenQuant.API;
using OpenQuant.API.Indicators;

using System.Drawing;
```

```
public class
MyStrategy : Strategy {

// quantity to buy on a trade
[Parameter ("Order quantity (number of contracts to trade)")]
double Qty       = 100;

// The bar block size tells the strategy to only
// trade on the boundaries of bar blocks that contain
// multiple bars. This method helps to avoid whipsaws
// because (on average) the strategy will have to
// wait (BarBlockSize/2) bars before taking a new
// trading action, to either open or flip a position.
// This way you won't flip positions on unprofitable
// back-to-back bars that whipsaw the strategy.
//
// In Altucher's book, the strategy only trades on Monday
// morning, which is why the bar block size is 6 days long.
[Parameter ("Bar Block Size")]
int BarBlockSize = 6;

[Parameter ("Length of SMA in blocks (weeks)", "SMA")]
int FastSMALength = 22;

[Parameter ("Length of SMA in blocks (weeks)", "SMA")]
int SlowSMALength = 55;

int  positionInBlock = 0;
bool buyOnNewBlock;
bool sellOnNewBlock;

// two moving averages
SMA  fastSMA;
SMA  slowSMA;

public override void
OnStrategyStart () {

    // set up the fast average
    fastSMA = new SMA (Bars, FastSMALength * 7, Color.Yellow);
    Draw (fastSMA, 0);

    // set up the slow average
    slowSMA = new SMA (Bars, SlowSMALength * 7, Color.Pink);
    Draw (slowSMA, 0);
}

public override void
OnBarOpen (Bar bar) {
    // calc quantity to reverse a position
    double orderQty = 2 * Qty;
    if (!HasPosition)
        orderQty = Qty;

    // if we have no open position, maybe open a position
    if (positionInBlock == 0) {
        if (buyOnNewBlock) {
            Buy (orderQty, "Reverse to Long");
            buyOnNewBlock = false;
        }
        if (sellOnNewBlock) {
```

```
            Sell (orderQty, "Reverse to Short");
            sellOnNewBlock = false;
        }
    }
}

public override void
OnBar (Bar bar) {
    // if our SMAs contain the current bar date
    if (fastSMA.Contains (bar.DateTime)
        && slowSMA.Contains (bar.DateTime)) {

        // see which one is above the other
        Cross cross = fastSMA.Crosses (slowSMA, bar);

        // set flags to say which way to flip position
        // when the start of the next bar block arrives
        if (cross == Cross.Above)
            buyOnNewBlock = true;
        if (cross == Cross.Below)
            sellOnNewBlock = true;
    }

    // This strategy uses an interesting method of only
    // considering 6-bar blocks to trade. This might help
    // to reduce whipsaws. The code below first increments
    // the number of bars that we have seen in the current
    // block, then takes a modulo of the number to keep the
    // block count numbers between 0 and the block size limit.
    positionInBlock = (positionInBlock++) % BarBlockSize;
}
}
```

4.5.3 Chande's 65sma_3cc Strategy

This strategy is a moving average crossover strategy from Chande's *Beyond Technical Analysis* book. It uses a 65 day simple moving average (65sma), and an addition confirmation condition of "three consecutive closes" (3cc). Thus the name of this strategy is called "65sma_3cc."

Like other strategies shown previously, this strategy enters on a single condition (moving average crossover with 3 consecutive higher closes), but can use three possible exit methods. It can exit after a particular number of bars go by, it can exit on a trailing stop, or it can exit on a moving average crossover in the opposite direction. The moving average crossover exit is called the "trend following exit" in the code.

This is a more complex strategy than the ones previously shown, because it uses several new technical indicators to control trade entries and exits. In particular, to avoid trading whipsaws, this strategy uses two trade entry filters based on ADX and RAVI index values.

ADX stands for Average Directional Index. This is a technical indicator that helps to recognize trending markets, so that trend following systems can avoid being whipsawed by fast reversals in moving average crossovers in flat or sideways markets. When this index rises, it indicates a trending market; when it falls, a sideways market. Typically, when the ADX indicator is above 40, and then falls, a sideways or consolidating market is emerging. Conversely, when the

indicator is below 20 and then rises, a trending market is emerging. The 65sma_3cc strategy uses an ADX filter to avoid entering new trades in a sideways market.

RAVI stands for Range Action Verification Index. This is a technical indicator that also helps to recognize trending markets, for the same purpose of helping trend following systems to avoid entering whipsaw trades in sideways markets. The RAVI index is a moving average crossover system itself—it uses a 7-day fast average and a 65-day slow average. The RAVI index value is defined as the absolute value of the percentage difference between the 7 and 65 day averages. When a market is moving sideways, the two averages tend to have the same values, so the difference is small. Conversely, when the market is trending, the fast average rapidly pulls away from the slow average, producing a larger difference and larger index value. Generally speaking, a RAVI value below 3 percent indicates sideways prices, and above 3 percent, trending prices.

The properties parameters defined for the 65sma_3cc strategy enable you to choose which filter you want to use (ADX or RAVI), and also let you control a variety of moving average lengths, stop loss limits, and other strategy parameter values. Chande says that this strategy (not this particular implementation of it) has been tested on 20 years of data for 23 different markets, and that it was robust and profitable on each one. So this should be a good example of a computerized trend following system.

Finally, notice that this strategy has more user-settable properties and many more helper methods than do other strategies in this document. The helper methods increase code readability because they isolate utility code into small, manageable pieces.

Here is the complete code for the strategy.

```
// Chande's 65-sma 3cc Strategy
using System;
using System.Drawing;

using OpenQuant.API;
using OpenQuant.API.Indicators;

// this enum defines allowed filter values
public enum FilterType {
    None,
    RAVI,
    ADX
}

public class
MyStrategy : Strategy {

[Parameter ("Order quantity (number of contracts to trade)")]
double Qty           = 100;

[Parameter ("Bars Exit Level", "Bars Exit")]
bool BarCountExitEnabled;

[Parameter ("Bars to Exit Count", "Bars Exit")]
int  BarsToExitCount = 20;

[Parameter ("Consecutive Closes Count")]
int  ConsClosesCount = 3;

[Parameter ("SMA Length")]
```

```
int  SMALength       = 65;

[Parameter ("Stop Exit Level", "Stop Exit")]
double TrailingStopLevel = 500;

[Parameter ("Stop Exit Enabled", "Stop Exit")]
bool TrailingStopEnabled;

[Parameter ("Trend-Following Exit Enabled", "Trend-Following Exit")]
bool TrendFollowingExitEnabled;

[Parameter ("Trend-Following Exit Length", "Trend-Following Exit")]
int  TrendFollowingExitLength = 14;

[Parameter ("Filter Type", "Filter")]
FilterType FilterType = FilterType.None;

[Parameter ("ADX Length", "ADX")]
int  ADXLength       = 14;

[Parameter ("ADX Level", "ADX")]
double ADXLevel      = 20;

[Parameter ("Short SMA Length (RAVI)", "RAVI")]
int  ShortSMALength  = 7;

[Parameter ("RAVI Percent Level", "RAVI")]
double RAVILevel     = 0.5;

// the slow average is 65 bars long by default
SMA  sma;

// only enter new trades if no position exists
bool entryEnabled = false;

// for consecutive closes
int  ccCount = 0;

// record the crossing state
Cross smaCross = Cross.None;

// for the bar count exit method
int  barsFromEntry = 0;

// for the trailing stop exit method
Stop trailingStop;

// Exit when High/Low exceed previous price range
bool exitOnBarOpen = false;

// RAVI Filter parameters
SMA  shortSMA;

// ADX Filter parameters
ADX  adx;

// shares to buy, and trading orders
Order buyOrder;
Order sellOrder;

public override void
OnStrategyStart () {
    sma = new SMA (Bars, SMALength);
```

```
    sma.Color = Color.Yellow;
    Draw (sma, 0);

    // if required, set up the RAVI moving average
    if (FilterType == FilterType.RAVI) {
        shortSMA = new SMA (Bars, ShortSMALength);
        shortSMA.Color = Color.Pink;
        Draw (shortSMA, 0);
    }
    //if required, set up the ADX moving average
    if (FilterType == FilterType.ADX) {
        // ADX is a builtin function, like SMA
        adx = new ADX (Bars, ADXLength);
        adx.Color = Color.Wheat;
        Draw (adx, 2);
    }
}

public override void
OnBar (Bar bar) {
    if (sma.Count == 0)
        return;

    // if we are using a trend-following exit and have an open
    // positiong that we should close
    if (TrendFollowingExitEnabled && HasPosition
            && Bars.Count > TrendFollowingExitLength + 1) {
        // check if we are long and today's close is lower than
        // lowest low of the last "trendFollowingExitLength" bars.
        // If so, then exit on the next bar open
        if (Position.Side == PositionSide.Long) {
            double prevLow = Bars.LowestLow
                (Bars.Count  - TrendFollowingExitLength
                             - 1, Bars.Count - 2);
            if (bar.Close < prevLow)
                exitOnBarOpen = true;
        }

        // check if we are short and today's close is higher than
        // highest high of the last "trendFollowingExitLength" bars
        // If so, exit on the next bar open
        if (Position.Side == PositionSide.Short) {
            double prevHigh = Bars.HighestHigh
                (Bars.Count - TrendFollowingExitLength
                            - 1, Bars.Count - 2);
            if (bar.Close > prevHigh)
                exitOnBarOpen = true;
        }
    }

    // look for N consecutive closes after a crossover
    Cross cross = Bars.Crosses (sma, bar);
    // if any cross occurred, reset the consecutive close count,
    // and copy the cross value so we can reset our copy of it
    // without wiping out the original indicator.
    if (cross != Cross.None) {
        smaCross = cross;
        ccCount = 0;
    }

    // if a cross occurred, increment the cc count, because the
    // first bar counts as the first consecutive close
    if (smaCross != Cross.None)
```

```
            ccCount++;

        // if we have enough consecutive closes, it's time to trade
        if (ccCount == ConsClosesCount) {
            // if we have no position open, or if we have a position
            // that is opposite the cross direction (ie, we need to
            // close the position)
            if (!HasPosition || (Position.Side == PositionSide.Long
                                    && smaCross == Cross.Below)
                    || (Position.Side == PositionSide.Short
                            && smaCross == Cross.Above)) {
                switch (FilterType) {
                // enter a trade if no filters are being used
                case FilterType.None: {
                    entryEnabled = true;
                    break;
                }

                // enter a trade if the RAVI filter says ok
                case FilterType.RAVI: {
                    entryEnabled = FilterRAVI ();
                    break;
                }

                // enable a trade if the ADX filter says ok
                case FilterType.ADX: {
                    entryEnabled = FilterADX ();
                    break;
                }
                }

                // if an entry was enabled,
                // open a position on next bar open
                if (entryEnabled)
                    exitOnBarOpen = false;
                // and reset our copy of the cross status to none
                smaCross = Cross.None;
            }
            // reset the consecutive close count too
            ccCount = 0;
        }
    }

    public override void
OnBarOpen (Bar bar) {
        // if we should close our position because of
        // the trend-following exit
        if (exitOnBarOpen) {
            exitOnBarOpen = false;
            // if we have a position open, close it
            if (HasPosition) {
                ClosePosition ();
                return;
            }
        }

        // if we should enter a trade
        if (entryEnabled) {
            entryEnabled = false;
            // and if we have no existing position
            if (!HasPosition) {
                // go long if our bar is above the moving average
                if (Bars.Last.Close >= sma.Last)
```

```
                OpenPosition (OrderSide.Buy);
            // go short if our bar is below the moving average
            if (Bars.Last.Close <= sma.Last)
                OpenPosition (OrderSide.Sell);
        }
        // if we have an existing position, reverse it,
        // because the trend direction has changed.
        else
            ReversePosition ();
    }
    // else if we should be using the bar count exit instead
    // of the trend following exit
    else {
        // bars exit
        if (BarCountExitEnabled) {
            // if we have a position to close,
            // close the position and reset the bar counters
            if (HasPosition) {
                barsFromEntry++;
                if (barsFromEntry == BarsToExitCount) {
                    barsFromEntry = 0;

                    ClosePosition ();
                }
            }
            // else if we have no position open,
            // reset the bars count to zero for next time
            else
                barsFromEntry = 0;
        }
    }
}

public override void
OnPositionChanged () {
    // every time our position size or direction changes,
    // cancel the old trailing stop and set a new one
    CancelExit ();
    if (HasPosition)
        SetExit ();
}

private void
CancelExit () {
    // reset the bar counter and cancel the trailing stop
    barsFromEntry = 0;
    if (trailingStop != null)
        trailingStop.Cancel ();
}

private void
SetExit () {
    // reset the bar counter and set a new trailing stop
    // Notice this stop is just an internal signal, it is
    // not a real stop loss order. The real order is issued
    // in OnStopExecuted, when the stop is triggered.
    barsFromEntry = 0;
    if (TrailingStopEnabled)
        trailingStop
            = SetStop (TrailingStopLevel
              / Qty, StopType.Trailing, StopMode.Absolute);
}
```

```
public override void
OnStopExecuted (Stop stop) {
    // when the stop is triggered, close the position
    ClosePosition ();
}

private void
ClosePosition () {
    // create and send a market order to close the position
    if (Position.Side == PositionSide.Long)
        Sell (Qty, "Exit");
    else
        Buy (Qty, "Exit");
}

private void
OpenPosition (OrderSide side) {
    // create and send a market order to open the position
    Order order = MarketOrder (side, Qty, "Entry");
    if (side == OrderSide.Buy)
        buyOrder = order;
    else
        sellOrder = order;
    order.Send ();
}

private void
ReversePosition () {
    // reverse the position with a market order
    // Use double the position size to flip the position
    if (Position.Side == PositionSide.Long) {
        sellOrder = MarketOrder (OrderSide.Sell, Qty * 2,
                                 "Reverse the Position");
        sellOrder.Send ();
    }
    else {
        buyOrder = MarketOrder (OrderSide.Buy, Qty * 2,
            "Reverse the Position");
        buyOrder.Send ();
    }
}

private bool
FilterRAVI () {
    // calculate the latest RAVI value
    if (shortSMA.Count == 0)
        return false;
    double smaLast      = sma.Last;
    double shortSMALast = shortSMA.Last;

    double ravi         = Math.Abs (smaLast - shortSMALast)
                        / Math.Min (smaLast, shortSMALast) * 100;

    // return true to accept the trade, false to block it
    if (ravi >= RAVILevel)
        return true;
    else
        return false;
}

private bool
```

```
FilterADX () {
    if (adx.Count == 0)
        return false;

    // return true to accept the trade, false to block it
    if (adx.Last >= ADXLevel)
        return true;
    else
        return false;
}
}
```

4.6 Gap Closing Strategies

The main idea of a gap closing strategy is that prices often reverse to close a gap that opens up between adjacent bars. “Often” means something like 60% of the time, under the right circumstances, prices will retrace to close open gaps. You can think of a price gap as people or prices overreacting to news, or overshooting the mark because they move too far, too fast. The closing of the gap can be viewed as a corrective return to more sanity. Gap closing strategies are popular with day traders and hedge funds because they are easy to recognize, easy to trade, and can be profitable if executed carefully under the right circumstances. Gap closing strategies can also be automated fairly easily.

This section describes several gap closing strategies from the book *Trade Like a Hedge Fund*, by Altucher. Altucher was a hedge fund manager, and talks about 20 successful and uncorrelated strategies in his book (not all of them can be easily automated). This section describes the rules and OpenQuant code for several of his strategies.

All of the following strategies show code from an Strategy that implements the strategy with an integrated framework. First the entire code for the Strategy is shown, and then sections of the code are reproduced with explanations.

4.6.1 Stock 2% gap

This example watches for a stock to gap down 2%, then buys the gap (long) to close the position when the gap closes, or at the end of the next bar after the gap bar. So this strategy only holds a position open for 1 bar length of time. But we present the example here using daily bars, to make the explanation more intuitive.

Here is the code for the entire component, including the boilerplate sections of (1) assembly references and (2) class headers. To save space, later examples will omit the boilerplate assembly references, strategy component attributes, and class headers. These bits of code are all essentially the same, for all examples.

```
// Stock 2 Percent Gap
using OpenQuant.API;

public class
MyStrategy : Strategy {

[Parameter ("Gap percent")]
double Percent = 2;
```

```
[Parameter ("Order quantity (number of contracts to trade)")]
double Qty = 100;

double prevClose;
Order sellOrder;

public override void
OnBar (Bar bar) {
    prevClose = bar.Close;
    if (HasPosition) {
        sellOrder.Cancel ();

        Sell (Qty, "Exit");
    }
}

public override void
OnBarOpen (Bar bar) {
    if ((prevClose - bar.Open) / prevClose > Percent / 100)
        Buy (Qty, "Entry");
}

public override void
OnPositionOpened () {
    sellOrder = SellLimitOrder (Qty, prevClose, "Limit Exit");
    sellOrder.Send ();
}
}
```

We use the OnPositionOpened event handler to automatically issue a sell limit order after the position is created for the buy order issued in OnBarOpen. The limit sell order uses the previous day close as the limit price.

We use OnBar to update the trailing prevClose variable with the current bar.Close price. Recall that OnBar fires on the trailing edge of the bar, so the current bar.Close price becomes the prevClose price for all following events. OnBar is also used to sell at today's close if we still have a position open (maybe the limit order did not automatically fire during the construction of the current bar).

The overall sequence of events is as follows. If a suitable gap is seen between bars (remember we look for gaps in OnBarOpen), we issue an immediate market buy signal. The strategy sends the order to the broker, and when it receives confirmation, the framework opens a new position. The opening of a new position fires the OnPositionOpened event, which we catch and use to create and send a limit sell order to the broker for the buy we just made. (The broker actually passes the stop order on to the exchange servers, where the order resides until it is cancelled, expires, or is executed.) All this happens before the next bar arrives.

During the calculation of the next bar (which takes 15 minutes trading time for 15 minute bars, or all day for daily bars), the exchange is constantly comparing the limit sell order to current market prices. If the exchange prices trigger the limit order, the order is executed, the strategy is notified, and the position is closed.

When the next bar arrives, OnBar is fired, and we see the trailing edge of the bar. If we still have a position open, it means that the limit order at the exchange was never executed. So according to the strategy rules, we cancel the existing limit order and close the position at the market price.

4.6.2 Stock down, stock 2% gap

This example is just like the one above, except that it also requires that the previous bar (the previous day) was a down day for the stock, in addition to gapping down 2% at today's open.

```
// Stock down, Stock 2 Percent Gap
using System;
using System.Drawing;

using OpenQuant.API;
using OpenQuant.API.Indicators;

public class
MyStrategy : Strategy {

[Parameter ("Order quantity (number of contracts to trade)")]
double Qty     = 100;

[Parameter ("Percent")]
double Percent = 2;

private Order  sellOrder;
private double prevClose;
private bool   downDay = false;

// if a position exists, close it at end of trading today
// The limit order was not triggered during the current bar period.
// This means the position is held open a maximum of 1 bar of time
public override void
OnBar (Bar bar) {
    prevClose = bar.Close;

    // update flag to remember if stock went down today
    downDay = (bar.Open > bar.Close);
    if (HasPosition) {
        sellOrder.Cancel ();

        Sell (Qty, "Exit");
    }
}

// Buy long if yesterday was a down day, and if today's open
// price has gapped down more than 2% below yesterday's close
public override void
OnBarOpen (Bar bar) {
    if (downDay) {
        if ((prevClose - bar.Open) / prevClose > Percent / 100)
            Buy (Qty, "Entry");
    }
}

// Sell at yesterday's closing price if you reach it today (within the
// current bar period). Otherwise close position at end of trading day
// (at end of current bar).
// install a profit target limit order at yesterday's close
```

```
public override void
OnPositionOpened () {
    sellOrder = SellLimitOrder (Qty, prevClose, "Limit Exit");
    sellOrder.Send ();
}
}
```

4.7 Spread and Volatility Trading Strategies

The main concept behind spread trading systems is that you try to profit by trading the spread between two instruments (or two derivations from other sources of your choice). Spread trading systems are similar to gap closing systems in that when the spread (or gap) gets too big, you open a position that will profit from a closure of the spread (or gap) back to normal conditions.

To implement a spread trading strategy, one common approach is to calculate a ratio between the two price series that define the spread, and then trade the volatility of the ratio against itself (compared to its normal range of values).

For example, suppose the ratio normally has a value of 1.2 with a standard deviation of 0.2. This means that 68% of the time the spread—as indicated by the value of the ratio—would be between 1.0 and 1.4 (one standard deviation from the mean of 1.2). It also means that 94% of the time the value would be between 0.8 and 1.6 (two standard deviations from the mean of 1.2).

So if your strategy saw a ratio value of 2.0 during trading, the strategy could recognize that the current ratio value (the spread) was WAY outside of normal bounds, and could open a position to profit from a return to normal ratio conditions. A true spread trading strategy would go long on the instrument with the low price, and short on the instrument with the high price of the spread. That way, the spread strategy would not be exposed to long term price moves while the trading position was open.

The following strategy follows the trading algorithm described above—it first calculates a ratio from the two spread series, and then trades against abnormal values of the ratio. But as we shall see, this example is not a true spread trading strategy because it only trades one instrument. This makes it more of a plain volatility strategy than a spread trading strategy.

4.7.1 QQQ Crash, QQQQ Trade

The Nasdaq 100 (QQQ) index is fairly volatile because it contains so many technology stocks whose prices move around so much more than average non-technology stocks. The volatility of the QQQ has many sources, including rapid technological advances, competition, and sudden news that can affect stock prices.

The strategy below waits for a big “crash” in the QQQ, where a crash is defined as QQQ prices moving below a lower Bollinger band set 1.5 standard deviations away from a 10-day moving average. While using 1.5 standard deviations is not unusual for Bollinger bands, the use of 1.5 with a short 10-day moving average is an unusual combination.

To move below a lower Bollinger band that is based on a 10 day average requires a pretty drastic price move, because a fast 10-day average will stick closely to price movements. So this strategy

looks for drastic drops in QQQ, buys long the morning after the crash (in OnBarOpen), and then trades the gap back up as usual. It exits when a profit is made, or after 30 days.

We put this strategy under the volatility section because it uses methods that deliberately select volatile moves (1.5 standard deviations, 10 day average). In contrast, our other Bollinger band examples are listed under the range trading section, because they focus on slower moving averages and less volatile price moves.

```
// QQQQ Crash, QQQQ Trade
using OpenQuant.API;
using OpenQuant.API.Indicators;

using System.Drawing;

public class
MyStrategy : Strategy {

[Parameter ("Order quantity (number of contracts to trade)")]
double Qty    = 100;

[Parameter ("Length of BBL")]
int BBLLength = 10;

[Parameter ("Order of BBL")]
double BBLOrder = 1.5;

[Parameter ("Max number of bars, while position is active")]
int MaxDuration = 20;

// bollinger band, 10 days, 1.5 std deviation
BBL    bbl;
// remember what you paid on entry
double entryPrice;
// exit after 20 days
int    barsFromEntry = 0;
// orders and quantity
Order  buyOrder;

public override void
OnStrategyStart () {
    // set up bollinger bands
    bbl = new BBL (Bars, BBLLength, BBLOrder);
    bbl.Color = Color.Yellow;
    Draw (bbl, 0);
}

public override void
OnBar (Bar bar) {
    // if we don't have a position and we have some bars
    // in the bollinger series, try to enter a new trade
    if (!HasPosition) {
        if (bbl.Count > 0) {
            // if the current bar is below the lower bollinger band
            // buy long to close the gap
            if (Bars.Crosses (bbl, bar) == Cross.Below) {
                buyOrder = MarketOrder (OrderSide.Buy, Qty, "Entry");
                buyOrder.Send ();
            }
        }
    }
```

```
    else {
        // else if we DO have an existing position, and if
        // today's bar is above our entry price (profitable),
        // then close the position with a market order
        if (entryPrice < bar.Close) {
            barsFromEntry = 0;

            Sell (Qty, "Exit (Take Profit)");
        }
        else
            barsFromEntry++;
    }
}

public override void
OnBarOpen (Bar bar) {
    if (barsFromEntry == MaxDuration)
        Sell (Qty, "Sell");
}

public override void
OnPositionOpened () {
    // when a position is opened, remember what we paid for
    // the instrument. Notice we use the average price of
    // all the partial fills that we might have received.
    entryPrice = buyOrder.AvgPrice;
}

public override void
OnPositionClosed () {
    barsFromEntry = 0;
}
}
```

4.7.2 Bollinger Bands with 5-Minute Bars

This Bollinger band strategy is intended for short intraday trades that last no longer than 5 or 10 minutes. It enters a trade if prices fall 3% below the lower Bollinger band, and exits a trade when a 1% profit target has been reached, or when two bars have passed since the trade was entered.

There are several interesting features of this system. It uses flags to say when a trade can be entered or exited (when two bars have passed). It uses the OnTrade event handler to look at trade data during bar formation intervals (so the strategy is working within two timeframes—trades and bars). And it uses SetStop method to set a stop reminder based on time (5 minutes) to remind the strategy to close the position automatically after 5 minutes have passed with an open position.

```
// Bollinger Bands with 5-Minute Bars
using OpenQuant.API;
using OpenQuant.API.Indicators;

using System.Drawing;

public class
MyStrategy : Strategy {

[Parameter ("Order quantity (number of contracts to trade)")]
double Qty     = 100;
```

```
[Parameter ("Percent")]
double Percent = 3;

[Parameter ("Profit Target")]
double ProfitTarget = 1;

[Parameter ("Length of BBL", "BBL")]
int    BBLLength = 10;

[Parameter ("Order of BBL", "BBL")]
double BBLOrder = 2;

BBL   bbl;
Order sellLimit;
Order buyLimit;
private int barsFromEntry = 0;

public override void
OnStrategyStart () {
    bbl = new BBL (Bars, BBLLength, BBLOrder);
    bbl.Color = Color.Yellow;
    Draw (bbl, 0);
}

public override void
OnBar (Bar bar) {
    if (bbl.Contains (bar.DateTime)) {
        if (!HasPosition) {
            // cancel previous buy limit
            if (buyLimit != null)
                buyLimit.Cancel ();

            // calculate limit price
            double buyPrice = bbl.Last * (1 - Percent / 100);

            // place new limit orders
            buyLimit = BuyLimitOrder (Qty, buyPrice, "Entry");
            buyLimit.Send ();
        }
        else {
            barsFromEntry++;

            // close position at the second bar after entry
            if (barsFromEntry == 2) {
                barsFromEntry = 0;
                sellLimit.Cancel ();
                Sell (Qty, "Exit (Second Bar After Entry)");
            }
        }
    }
}

public override void
OnBarOpen (Bar bar) {
    // place limit order at the start of next bar after entry
    if (barsFromEntry == 1)
        PlaceSellLimit ();
}

public override void
OnOrderFilled (Order order) {
```

```
    if (order == sellLimit)
        barsFromEntry = 0;
}

private void
PlaceSellLimit () {
    // calculate price that satisfies the profit target
    double sellPrice = buyLimit.AvgPrice
        * (1 + ProfitTarget / 100);

    sellLimit = SellLimitOrder (Qty, sellPrice,
        "Exit (Profit Target)");
    sellLimit.Send ();
}
}
```

Bibliography

Beyond Technical Analysis 2ed by Tushar S Chande. This is a superb book on developing and implementing trading systems (mechanical or manual). If you intend to spend any significant time or money on developing a trading system, this is a book that you should read first.

Trade like a Hedge Fund by James Altucher. This book provides "20 successful uncorrelated strategies and techniques to winning profits" (not all of which can be automated). Many of the strategies in this book are used as examples later in this document.

5 Terminology

Inconsistent terminology makes it harder for readers to understand new concepts of any kind. To help avoid that problem in this document, we provide the following consistent definitions for words that are used throughout OpenQuant documentation. We do not claim that the definitions below are universally accepted in the industry—but they are good enough and consistent enough to meet the needs of the discussions in this document.

Component. A component is a computer file that contains one or more software components. For example, a component could be an Entry or Exit component that is listed in the Components panel of the IDE. Component is a technical term that means "a computer file," not a functional strategy term that means a "functional component" such as Entry or Exit.

Partitioned Framework. A partitioned framework *partitions* (splits) the functions of a strategy (such as Entry and Exit) into *multiple* components (files). Using separate components makes it easier for developers to mix and match individual strategy functions into an overall strategy that they like (simply by right clicking and choosing the components of interest). When you create a new IDE strategy project, you must choose a partitioned or an integrated strategy framework.

Integrated Framework. An integrated framework *integrates* the functions of a strategy (such as Entry, Exit, Market Manager, and Risk Manager) into a *single* component. An integrated strategy framework makes it easier for developers to share trading information—such as instruments, prices, and trades—among the functions of a strategy. (In contrast, information is difficult to

share across component boundaries in the partitioned framework). When you create a new IDE project, you must choose a partitioned or an integrated strategy framework.

Unaggregated Market Data. Unaggregated market data originates with a trading exchange, passes through a market data provider (or the database equivalent of a provider, in simulation mode), and is received by your strategy without aggregation processing. That is, no aggregation operations are performed to group, simplify, or otherwise change the original data flow. The term unaggregated data typically refers to Quotes and Trades.

Quote Data. Quote data originates with a market data provider, and contains at least a bid/ask pair of prices and a timestamp. (OpenQuant quotes contain the best bid/ask quotes among active providers.) Quotes may contain other information as well. In a OpenQuant strategy component, incoming quotes are usually handled by an OnQuote event handler method.

Trade Data. Trade data originates with a market data provider, and contains at least an instrument name, trade price, timestamp, and volume (size) of the trade. In a OpenQuant strategy component, incoming trades are usually processed by an OnTrade event hander method. Trade data is also known as "Time and Sales" data.

Time and Sales Data: Time and sales data is another industry term for trade data. This definition is provided here for completeness—the term "trade and sales" is not used in this document.

TAQ Data. TAQ is short for "trades and quotes," and normally refers to the NYSE trade and quote database. The term TAQ is listed here for completeness—it is not used in this document.

Tick. A tick is the minimum price move allowed for a particular instrument. Tick sizes are different for different financial instruments. For example, a tick might be 0.01 on a stock price, 0.05 on a futures contract, or 0.10 on an index option. Ticks are properties or attributes of a financial instrument, and are defined by the exchanges that trade instruments. At least in this document, ticks have nothing to do with quotes or trades. (Although some people use the term "ticks" to refer generally to trade and quote TAQ data.)

Tick Data. Tick data is a general term used to refer to *unaggregated data*. That is, *tick data* can refer to quote data, trade data, market depth data, or any other kind of unaggregated data.

Aggregated Data. Aggregated data is formed by processing *tick data* (unaggregated quote and trade data) into structured data objects that are more useful for your strategies. For example, Bar data is the most obvious type of aggregated data.

Bar Data. Bar data originates in the OpenQuant BarFactory, and is constructed from unaggregated quote and trade data.

Instrument. A financial *instrument* is something that carries financial value, such as a stock, a bond, a futures contract, or an equity option.

Technical Indicator. A *technical indicator* is a mathematical, formula-calculated value that indicates some interesting property of the underlying financial instrument. For example, one technical indicator is a simple 10-day moving average of daily closing price.