it's about time

# Developer Brief

## Introduction to Strategy Back-Testing in q

**Author:**

Daniel Walsh has worked as a kdb+ Consultant for First Derivatives since 2009. Daniel is currently based in New York and in his time with FD has worked on kdb+ algorithmic trading systems for both the equity and FX markets.

## TABLE OF CONTENTS

## 1 INTRODUCTION

The purpose of this whitepaper is to describe some of the capabilities of kdb+ for developing and testing trading strategies and to provide some examples of how to effectively test the performance of a strategy.

Back-testing is a key component of bringing any trading strategy to market. In order to grasp how strategies will perform it is necessary to test them in an environment as close as possible to market realities. The changing nature of algorithmic trading means that these conditions are constantly evolving. As a result, the metrics used for developing and testing strategies will need to adapt also.

We outline in this whitepaper how to create a dummy environment to monitor the performance of a particular set of strategies. Here they are applied to the foreign exchange (FX) market, but can be adapted to facilitate other asset classes. The strategies make decisions on whether or not to accept trades based on specified input parameters.

While market conditions cannot be replicated with complete certainty, we ultimately show one way of how a simple algorithmic strategy can be back-tested against historical FX quotes and trade data. The performance of the strategy highlights the strengths and weaknesses in a pre-production environment.

All tests were run using kdb+ version 3.2 (2015.01.14)

## 2 COMPONENTS OF A TEST ENVIRONMENT

In the FX market there are a number of different providers of market data. There are also several platforms where trades can be executed. These platforms are commonly referred to as an Electronic Communication Network (ECN). Our strategy will perform Complex Event Processing (CEP) using a market data adaptor and orders from an ECN.

There are different rules for different ECNs, such as:

- Certain exchanges support certain types of orders. Good Till Canceled (GTC), Immediate Or Cancel (IOC), Fill Or Kill (FOK) etc.
- Minimum Quote Life (MQL) is enforced on certain ECNs. This means that when an order is placed in the market, neither the price nor size of the order can be amended for a defined period of time.

There are also different methods of receiving market data. In some cases, there are snapshots given, others provide updates as they come in.

- HotspotFX has two feeds, providing market data snapshots every 50ms or 250ms; the market data could have changed at any point over the time period but the update is not distributed until the next window has finished.
- The Chicago Mercantile Exchange (CME) futures data feed provides continuous market data updates as they see the market change.

### 2.1 Strategy

For the strategies that we are looking to test:

- There will be market data received via a continuous price feed with real time updates, no snapshots.
- The strategy will receive routed orders from an ECN that it can either accept or reject. If the strategy decides to fill the order, it will have to fill the entire order, no partial fills allowed.

When performing our tests, we will make an assumption that we are streaming a price to the ECN that is always the top of book price. This means that since we are the best price we will always be routed the order. The idea of being routed an order by an ECN and responding with a fill or reject is unique to the FX market. Usually an order is entered in the market and the ECN will report back on whether it was filled or not.

The conditions under which the strategy is willing to accept an order are based on the parameter settings that we specify. In this case these parameters will be:

1. 'Hedge' – a true or false flag to determine whether we are currently hedging a position.
2. 'Gap' – the absolute difference between the top level bid and ask price, the spread, of the order book.
3. 'Pip Range' – the maximum difference between the rate on the order versus the reference price provided that we are willing to accept an order at.

These strategy parameters will determine whether to accept an order or not. When each order arrives, the strategy will perform two calculations.

- If 'Hedge' is set to false then we calculate the spread on the reference price and compare this to our 'Gap' parameter.
- We calculate the difference between the rate on the order versus the reference price provided and this value is compared to the 'Pip Range' parameter.

If these conditions are met, the trading engine will accept the order.

## 2.2 Test Environment Design

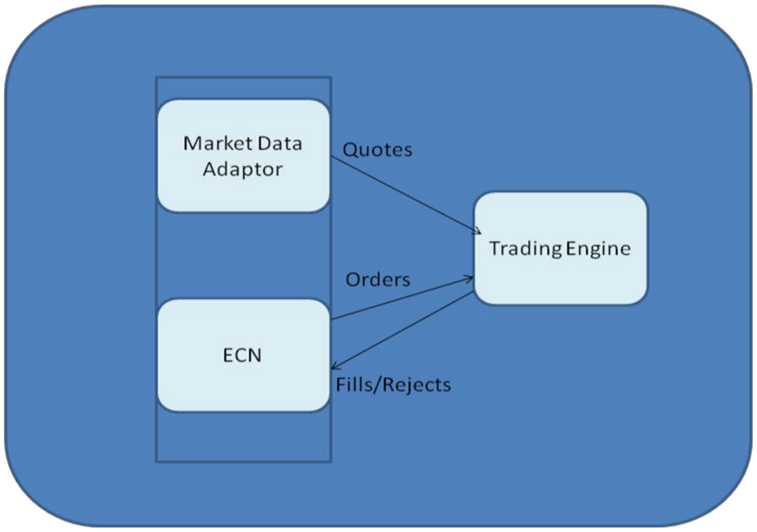
In our test environment, we will have three kdb+ processes:

1. A market data adaptor that publishes quotes. The market data will have two levels of depth and everything will be streamed as it arrives, no snapshot etc.
2. An ECN platform that publishes orders and processes order responses.
3. A trading engine where the strategy will run.

The trading engine will need to:

- Connect to the adaptor and ECN.
- Subscribe to market data updates from the adaptor.
- Respond to orders as they are sent from the ECN. These trade decisions will be based on the conditions of the strategy.

**Figure 1: Test Environment Setup**



### 2.2.1 Schema

The schema of the adaptor and the trading engine will differ for the following reasons:

- The strategy is only interested in the most recent market data update. This means it will only keep a quote table keyed by sym.
- The adaptor will publish data to the trading engine as well as storing records in local tables.

The trading engine process will have two additional tables, `fill` and `rejected`. As the names suggest, these tables will keep a record of what trades were filled and what trades were rejected. The `aprcs` and `bprcs` columns will contain the different levels of pricing where `aprcs` contains prices on the offer side and `bprcs` contains prices on the bid side.

```
// Adaptor Schema
quote:([]time:`time$();sym:`$();bpx:`float$();bsize:`float$();apx:`flo
at$();
asize:`float$();bszs:();bprcs:();aszs:();aprcs:());

trade:([]time:`time$();sym:`$();side:`$();rate:`float$();amount:`float
$());

// Trading engine schema
quote:update `u#sym from `sym xkey
([]time:`time$();sym:`$();bpx:`float$();bsize:`float$();apx:`flo
at$();
asize:`float$(); bszs:();bprcs:();aszs:();aprcs:());

trade:update `u#sym from`sym xkey
([]time:`time$();sym:`$();side:`$();rate:`float$();amount:`float$());

fill:update `g#sym from
([]time:`time$();sym:`$();side:`$();rate:`float$();amount:`float$());

rejected:update `g#sym from
([]time:`time$();sym:`$();side:`$();rate:`float$();amount:`float$());
```

### 2.2.2 Attributes

From the trading engine schema above, notice the attributes applied to the different tables.

- Since the `quote` and `trade` tables are to be keyed in the trading engine, this allows us to take advantage of the unique attribute. When we know that the entries of a list are unique, it allows q to exit some comparisons early.
- The grouped attribute has been applied to the `fill` and `rejected` tables. These tables will not be keyed as we'll want to see the result of each order rather than just the most recent entry. The grouped attribute here applies a structure to a table. This will improve query performance when executing `select` queries against the table.

### 2.2.3 Adaptor

In a typical set up there will be a feed handler acting as an adaptor that will be connected to an ECN. Each ECN will have a preferred method of transmitting the data.

These methods include:

- FIX protocol
- XML
- ITCH

kdb+ has built-in parser functions for FIX and XML data strings that are used by the feed handler to parse the data before being published to kdb+.

```
// Adaptor set up
\p 1234
upd:{[t;x]
        t insert x;
        .send[t][x];
        }

//list of subscribers
subs:()

//function called by subscribers
sub:{subs::distinct subs,.z.w}

.send.quote:{[x]
        {[h;x]neg[h](`upd;`quote;x)}[;x]each subs
        }

.send.trade:{[x]
        {[h;x]neg[h](`upd;`trade;x)}[;x]each subs
        }
```

In the adaptor described above, once a message is received from the market data provider and has been parsed into kdb+, the `upd` function will be called. At this point we can store the message and publish it to the adaptor subscribers. These subscribers will have registered for updates by opening a handle to the adaptor. Maintaining the subscriptions and handles between trading engines, adaptors and ECNs can be developed in-house, however there are products available such as First Derivatives' Delta Algo that can improve time-to-market.

For more on Delta Algo, see http://www.firstderivatives.com/downloads/Delta_Algo.pdf

## 2.3 Order Execution on the Test Environment

The adaptor will send each message to its subscribers through an asynchronous call to allow the adaptor to continue to publish the data as it arrives. This is especially useful when there are multiple processes subscribing to the data as with a synchronous call this could lead to a hold up at the adaptor level. For example, one of the trading engines may be busy processing earlier messages.

```
// Trading engine set up

/ Connection to adaptor
.limit.connections:(enlist `adaptor)!enlist `$"localhost:1234";

/ Open a handle to the adaptor, if this fails exit.
{[process]
  h:@[hopen;hsym process;{[x]show `$"No Connection, Exiting";exit[0]}]
  neg[h](`sub;::); }'[value .limit.connections];

/ Strategy parameters
.limit.parameters: (`hedge`gap`piprange)!(0b;0.0001;0.0002);


upd:{[t;x]
        upsert[t;x];
        .order[t][x;.limit.parameters];
        }

/ On receiving a trade
.order.trade:{[x;p]
        // Limit checks
        qt:quote[x`sym;(`sell`buy!(`apx`bpx))[sd:x`side]];
        // Market spread
        sprd:.[-;quote[x`sym;`apx`bpx]];
        // Market Gap
        gap:$[p`hedge;1b;sprd<=p`gap];
        // Price tolerance
        pricetol: ((neg[p`piprange];p`piprange)+\:qt);
        if[(x[`rate] within pricetol) & gap;:insert[`fill;enlist x]];
        // Trade rejected
        insert[`rejected;enlist x];
        }

/ On receiving a quote
.order.quote:{[x;p]}
```

When a quote is passed to the trading engine, there is no action taken in the `.order.quote` function. However when the quote initially arrives at the engine it is stored in a keyed table. This allows the trading engine to easily retrieve the data when the price check function, `.order.trade` is required.

The advantage here is that in the `.order.trade` function, we are only interested in the most recent value. Having a lookup on a keyed table rather than unkeyed tables reduces the time taken to retrieve the price.

## 3 STRATEGY PERFORMANCE

It is important to define the metrics by which a strategy will be judged:

- Profit and Loss (PNL) - both realized and unrealized PNL can be considered.
- Volume – the amount of business that was transacted.
- Attributions or the rate of execution on each order versus the next market data update.

There are cases where profit is not the most important metric on which strategy performance can be judged. For example, consider a Strategy A, designed to keep overall exposure low. It may be configured to trade very aggressively. This will not always be the most profitable situation. However, it could be for the benefit of Strategy B, running in parallel, whose success is dependent on Strategy A, keeping overall positions low.

For the purpose of our test exchange, we will consider two scenarios:

1) Hedging strategy
2) Take strategy

In each case, 'Volume' will be the metric by which their success will be judged.

### 3.1 Hedging Strategy

For the hedging strategy we will assume that we are carrying a large exposure and will look to take as much of the volume as possible. In a situation like this the strategy is focused on getting filled on as many orders as possible. The hedge condition will be turned on; this reduces the chances of us rejecting an order.

```
q)// Parameters
q).limit.parameters
        | ::
hedge   | 1b
gap     | 0.0002
piprange| 0.0005

q)// Fill ratios
q)tables[]!count each value each tables[]
fill  | 1234
quote | 8
reject| 1681
trade | 8
```

The results show that of the 2915 trades we took 42% of the volume. The parameters that were used in this example were quite wide. The strategy was allowing the rate on the order to be within a 10 pip range.

## 3.2 Take Strategy

For the take strategy, we will pay more attention to the quote feed we receive and base our trading decisions around it. In this case we will look to take when the reference feed is showing a very tight price.

```
q)// Parameters
q).limit.parameters
        | ::
hedge   | 0b
gap     | 0.0001
piprange| 0.0001

q)// Fill ratios
q)tables[]!count each value each tables[]
fill  | 0
quote | 8
reject| 2915
trade | 8
```

In this case, there were no fills as there was no order that fell within the 2 pip range off the reference price. This suggests that a strategy like this may not be suitable on any ECNs similar to our test environment.

## 4 CONCLUSION

This whitepaper is a brief introduction to a method of strategy development using kdb+. It focuses on a back-testing style framework that recreates market conditions to see whether certain strategies are suited to different ECNs. In our case, the first strategy is more suited to this environment.

Ultimately, trading strategies will need to be more adaptable than what has been displayed here. Markets move throughout the day. Spreads of currency pairs change depending on market activity and the parameters of a strategy will need to move with these changes. For the purpose of illustration, the data set was kept as straightforward as possible. In a real market scenario, there are a number of key components that need to also be considered. These fall outside the scope of this whitepaper but we describe some of them below.

The dataset provided contained 2 levels of depth. There is always a lot of information to be gained by analyzing the full order book.

- Different strategies will be interested in different parts of market data.
- One strategy may be looking for a tight spread at the top level price in which to take.
- Others may be looking for a large gap in either the price or size between different levels of an order book.

As well as changing market conditions, the rules of different ECNs need to be considered when developing and testing a strategy. One of the major factors that a strategy must consider is where it gets market data from and how this market data is published by its respective providers.

- Thomson Reuters, who stream snapshots every 250ms, has one sole data centre in London.
- EBS provides market data from three centers: New York, London and Tokyo.

You have to be conscious of the time taken to get an order to an ECN. For example, if the strategy is relying on market data from London, to place an order in New York, it needs to consider the round trip transit time to get the market data to New York.

Trying to come up with a one-size-fits-all solution is difficult and will not always provide the best results. Different strategies will be designed for different ECNs. It is only by analyzing the performance of different strategies in different ECNs that information can be gathered as to what strategy is suited for what ECNs.

All tests were run using kdb+ version 3.2 (2015.01.14).