it's about time

Developer Brief

Temporal Data: A kdb+ Framework for Corporate Actions

Author:

Sean Rodgers is a kdb+ consultant based in London. He works for a top-tier investment bank on a global tick capture and analytic application for a range of different asset classes.

TABLE OF CONTENTS

1 INTRODUCTION 3
2 CORPORATE ACTIONS 4
3 TEMPORAL DATA 5
3.1 Non-sorted dictionary 5
3.2 Sorted dictionary 5
3.3 Non-sorted keyed table 6
3.4 Sorted keyed table 6
4 CORPORATE ACTION NAME CHANGE 7
4.1 Requirements 7
4.2 Reference Data 7
4.3 Corporate Action Table 7
4.4 Research In Motion 8
4.5 Daily Correction Table 8
5 THE DATA 10
5.1 Corporate Action Adjustment 10
5.2 Get Results 11
6 STOCK SPLIT 13
7 CASH DIVIDEND 16
8 COMBINING ADJUSTMENTS 18
9 CONCLUSION 20

1 INTRODUCTION

kdb+ is leveraged in many financial institutions across the globe and has built a well-earned reputation as a high performance database, appropriate for capturing, storing and analyzing enormous amounts of data. It is essential that any large scale kdb+ system has an efficient design so that time to value is kept to a minimum and the end-users are provided with useful functionality.

This whitepaper examines a framework which can be used to apply corporate action adjustments on the fly to equity tick data.

Corporate actions are a common occurrence that brings about material changes to the underlying securities. We will look into the reasons why a company may choose to apply these actions and what consequences they have on tick data, with a goal to understanding what adjustments are needed and how best to apply them.

It is critical that a kdb+ system can handle these actions in a timely manner and return correct data to the user. Examples of a symbol name change, stock split and cash dividend will be outlined and for the purposes of this paper we will use Reuters cash equities market data.

All tests were run using kdb+ version 3.1 (2014.02.08)

2 CORPORATE ACTIONS

When the board of a company agrees to use a corporate action, there is a resulting effect on the underlying securities of that company and its shareholders. Name changes, stock splits, dividends, rights issues and spin-offs are all examples of corporate actions. However, the purpose of each varies and results in a different effect to the nature and quantity of the securities issued by that company.

Name Change

A company may decide to change its name to reflect a shift in company focus that targets a different core business. Alternatively, it could be to accompany expansion plans in where they require a name that translates across multiple languages. For whatever reason, only in name is the underlying security changed, yet, within a kdb+ system there must be a mapping in place to resolve this action.

Stock Split

If a stock is trading at a very high price it will deter many potential investors. A stock split will increase the number of outstanding shares whilst decreasing the share price accordingly, attracting investors that previously were priced out of the market. In this case size and price adjustments need to be applied to the data.

Cash Dividend

Profits made by companies can be distributed in part to their shareholders in the form of a cash dividend. Some companies, for example start-ups, may not do this to retain any profits as inward investment for growth.

Any investor who purchases a stock before the ex-dividend date (ex-date) is entitled to the dividend, however, beyond this date the dividend belongs to the seller. Therefore, dividends affect the pricing of a stock effective from this date with the number of outstanding shares remaining the same.

Spin-off

As part of a business restructuring, spin-offs can be used to break a company up in order to concentrate on separate core competencies. No creation of shares takes place, only the filtering of existing shares into the separate new companies, each having an adjusted price based on the original stock.

Table 1: Corporate Actions formula for Price and/or Size adjustments

Action	Price Adjustment	Size Adjustment
Name Change	No change	No change
Stock Split	price%adj	size*adj
Cash Dividend	price*adj	No change
Spin-off	price*adj	No Change

The task for a kdb+ developer is how best to apply the adjustments in a consistent and generic manner.

3 TEMPORAL DATA

One option for dealing with corporate actions would be to capture the daily state of each record. However, this would create an unnecessarily large table over time. We are only interested in when a change occurs, marking them 'asof' in a temporal reference table.

In the following sections we look at the behavior of applying the sorted attribute to dictionaries and tables. Its characteristics are important in achieving temporal data to obtain meaningful results when passing any argument within the key range.

Adding the sorted `` `s# `` attribute to a dictionary indicates that the data structure is sorted in ascending order. When kdb+ encounters this, a faster binary search can be used instead of the usual linear search. When applied to a dictionary, an `` `s# `` attribute creates a step function.

3.1 Non-sorted dictionary

When querying a non-sorted dictionary, nulls (`) are returned for values that are not present in the dictionary key.

```
q)d:(100*til 5)!`a`b`c`d`e
q)d
0  | a
100| b
200| c
300| d
400| e

q)d 0 50 150 200 500
`a```c`
```

3.2 Sorted dictionary

Taking the same dictionary and applying the `` `s# `` attribute, instead of nulls the last known value will be returned.

```
q)d:`s#d
q)d
0  | a
100| b
200| c
300| d
400| e

q)d 0 50 150 200 75 500
`a`a`b`c`a`e
```

As a keyed table is a particular case of a dictionary, applying the `` `s# `` attribute has similar effect.

3.3 Non-sorted keyed table

When querying a non-sorted keyed table, nulls (`) will be returned for values that are not present in the table key.

```
q)tab:([date:.Q.addmonths[2013.01.01;]3* til
5];quarter_name:`Q1_2013`Q2_2013`Q3_2013`Q4_2013`Q1_2014)

q)tab
date       | quarter_name
-----------| ------------
2013.01.01| Q1_2013
2013.04.01| Q2_2013
2013.07.01| Q3_2013
2013.10.01| Q4_2013
2014.01.01| Q1_2014

q)tab([] date:2013.01.01 2013.05.05 2013.06.19 2013.08.25 2013.10.01)
quarter_name
------------
Q1_2013



Q4_2013
```

3.4 Sorted keyed table

Running the same query on the sorted version of the table will return more meaningful results.

```
q)tab:`s#tab

q)tab([] date:2013.01.01 2013.05.05 2013.06.19 2013.08.25 2013.10.01)
Quarter_Name
------------
Q1_2013
Q2_2013
Q2_2013
Q3_2013
Q4_2013
```

Setting `s# attribute on a vector has no memory cost and kdb+ will verify the data is in ascending order before applying the attribute.

4 CORPORATE ACTION NAME CHANGE

4.1 Requirements

When kdb+ is the foundation of a tick trade and quote database, its objective is to obtain a complete picture of a security's real time and historical trading activity. Securities from time to time can go through a name change; this is when a company announces that it will be changing its ticker. The following section will present an approach to accessing data for securities that experience this type of corporate action.

4.2 Reference Data

Adequate reference data is paramount to the ability of obtaining consolidated stats. It will play a critical part in forming the correct query to the historical data. Firstly, give each sym a unique identifier (`uid`) that will be constant for the life of a security. The assumption is made that there is a one-to-one correspondence between sym and a security at any given time. One can obtain this `uid` per security from an external reference data provider or can be maintained internally.
Introduced in kdb+ v3.0 GUID is now an option for `uid`.
(http://code.kx.com/wiki/Reference/Datatypes#Guid_.28from_kdb.v3.0.29)

4.3 Corporate Action Table

For ease of understanding, we will be using the sym index to build up this `uid` and the corresponding corporate action temporal data reference table (`cact`). In this example, trade and quote data is loaded from a `hdb_path` directory.

```
q)\l hdb_path/taq

q)cact:update uid:i, date:first date from ([]sym:sym)
sym       uid date
------------------------
AAB.TO     0  2010.10.04
AAV.TO     1  2010.10.04
ABX.TO     2  2010.10.04
ABT.TO     3  2010.10.04
ACC.TO     4  2010.10.04
ABC.TO     5  2010.10.04
..
//first date is used as it is the earliest point in the hdb and
therefore any Corporate Actions before this date are not applicable.

q)save `:/ref_path/cact.csv
`:/ref_path/cact.csv
```

We now have a table in which every distinct sym in the HDB has a `uid` assigned to it. When a security undergoes a name change, this file must reflect it. A daily correction file should be sourced with matching `uid` mapping. If a security goes through one or more name changes we need only map to its `uid` once and use it as a basis to efficiently query a sorted `cact` table obtaining all previous syms for the interested date range. An example will be outlined below.

4.4 Research In Motion

One high profile name change of recent times was that of Research In Motion (RIM) (NASDAQ: RIMM; TSX: RIM). This change was made in order to have a clear global brand, BlackBerry.

This decision was purely a marketing one and did not affect the underlying stock in any way other than to change its name.

The change to the company's ticker was effective from the start of trading on Monday, 04-Feb-2013 trading as "BB" on the Toronto Stock Exchange and "BBRY" on the NASDAQ.

In terms of a Reuters Instrument Code (ric) listed on the Toronto Stock Exchange RIM.TO became BB.TO.

Table 2: Blackberry Name Change

Effective-Date	Type	Event
04-Oct-2010	RIM.TO	First Date In hdb
04-Feb-2013	BB.TO	Name Change

4.5 Daily Correction Table

A daily correction file will be used to update the `cact` table as per the example below.

```
q)Daily_Cor:([]eff_date:(),2013.02.04;new_ric:`BB.TO;old_ric:`RIM.TO;ui
d:510);

 q)Daily_Cor
 eff_date   new_ric old_ric uid
 -------------------------------
 2013.02.04 BB.TO  RIM.TO   510

 q)cact:`uid`date xkey ("IDS";enlist csv) 0:`:/ref_path/cact.csv

 q)`cact upsert `uid`date xkey select uid:uid, date:eff_date,
sym:new_ric from Daily_Cor
 `cact

 q)cact:`uid`date xasc cact

 q)select from cact where uid=510
 uid date       | sym
 --------------| ------
 510 2010.10.04| RIM.TO
 510 2013.02.04| BB.TO

 q)save `:/ref_path/cact.csv
 `:/ref_path/cact.csv
```

Once this is completed all gateways should be notified to pick up the updated `cact` file and apply the `` `s# `` attribute.

```
 q)cact:`uid xasc `uid`date xkey ("IDS";enlist csv)
0:`:/ref_path/cact.csv

 q)cact:`s#cact;
```

5 THE DATA

Within the majority of kdb+ systems, data is obtained through the use of a gateway process as is discussed in Edition 7 of the Technical Whietpaper series 'Common design principles for kdb+ gateways.'

The gateway acts as an interface between the end user and the underlying databases. We would like to pass many different parameters into the function (`getRes`) that executes the query on the database, and perhaps more than the maximum number allowed in q, which is 8. For this reason we will use a dictionary as the single parameter. A typical parameter dictionary looks like the following:

```
q)params:`symList`startDate`endDate`startTime`endTime`columns`applyCact
!((),`BB.TO`RY.TO;2013.01.31;2013.02.04;14:30t;22:00t;`volume`vwap;`NC)
;
 q)params
 symList   | `BB.TO`RY.TO  /Requested instruments
 startDate| 2013.01.31    /Only take data from startDate
 endDate   | 2013.02.04    /Only take data to endDate
 startTime| 14:30:00.000  /Only take data from startTime
 endTime   | 22:00:00.000  /Only take data to endTime
 columns   | `volume`vwap  /Requested analytics
 applyCact| `NC           /To apply name change adjustments
```

Before this dictionary gets sent to the underlying resources the gateway can enrich the symList with very little expense over the `startDate(sD)` to `endDate(eD)` range to apply any name change corporate actions. This is described in section 5.1.

5.1 Corporate Action Adjustment

The following `cact_adj` utilizes the sorted `cact` table and reverse lookup to first indentify the `uid` for each sym and determine across all dates between the `startDate` to `endDate` range all associated syms.

```
 q)cact_adj:{[symList;sD;eD] days:1+eD-sD;symCount:count symList;t where
differ t:([]OrigSymList:raze days#/:symList)+ cact ([]uid:raze
days#/:((reverse cact)?/:symList)[`uid];date:raze symCount#enlist sD+
til days)};

 q)cact_adj . (`BB.TO`RY.TO;2013.01.31;2013.02.04)
 OrigSymList sym
 ------------------
 BB.TO       RIM.TO
 BB.TO       BB.TO
 RY.TO       RY.TO
```

We can now use this to update the parameters at the gateway level, only executing if the user indicated to apply Corporate Action Adjustment with `applyCact` flag set to `NC.

```
q)if[params[`applyCact]~`NC;params:@[params;`symList`origSymList;:;(cact
_adj . params`symList`startDate`endDate)`sym`OrigSymList]];

q)params
symList     | `RIM.TO`BB.TO`RY.TO
startDate   | 2013.01.31
endDate     | 2013.02.04
startTime   | 14:30:00.000
endTime     | 22:00:00.000
columns     | `volume`vwap
applyCact   | `NC
origSymList| `BB.TO`BB.TO`RY.TO
```

As you can see the `symList` has been updated with the pre and post corporate action syms. This would not have happened if the `` `s# `` attribute had not been applied.

5.2 Get Results

The new enriched `params` will then be sent to the hdb to obtain the result set by calling the `getRes` function.

```
q)getRes:{[params]:0!select vwap:wavg[size;price],volume:sum[size] by
sym from trade where date within params`startDate`endDate, sym in
params[`symList]}

q)res:getRes[params]
q)res
sym    vwap     volume
------------------------
BB.TO  14.31078 10890299
RIM.TO 12.91377 19889196
RY.TO  62.23244 6057164
```

Once the query is finished the result-set is sent back to the gateway for post processing. First, add the original symList `(origSymList)` passed by the user with a left-join.

```
q)res:(flip select sym:symList,origSymList from params) lj `sym xkey
res
q)res
sym    origSymList vwap     volume
-------------------------------------
RIM.TO BB.TO       12.91377 19889196
BB.TO  BB.TO       14.31078 10890299
RY.TO  RY.TO       62.23244 6057164
```

All that is left to do is to aggregate the data by the `origSymList`. Use of a functional select here has the power to do this.

http://code.kx.com/wiki/JB:QforMortals2/queries_q_sql#Functional_select

```
/aggregate by
q)b:(enlist `sym)!enlist `origSymList

/aggregate clauses
q)a:`volume`vwap!((sum;`volume);(wavg;`volume;`vwap))

q)res: 0!?[res;();b;a]
q)res
sym   volume   vwap
----------------------
BB.TO 30779495 13.40805
RY.TO 6057164  62.23244
```

The final step is to update the consolidated analytics with parameters that the user will find useful.

```
q)res:![res;();0b;`startDate`endDate`startTime`endTime#params]
```

The final result that is returned to the user is:

```
q)res
sym   volume   vwap     startDate  endDate    startTime    endTime
-----------------------------------------------------------------------
BB.TO 30779495 13.40805 2013.01.31 2013.02.04 14:30:00.000 22:00:00.000
RY.TO 6057164  62.23244 2013.01.31 2013.02.04 14:30:00.000 22:00:00.000
```

6 STOCK SPLIT

When a company decides to divide their common shares into a larger number of shares this is known as a stock split.

If a company proceeds with a five-for-one-split, all number of units held by shareholders would increase by 5 times, however, their equity will remain constant as share price changes accordingly. For example, if a shareholder held 1000 shares before the split each priced at £10, they would own 5000 shares after the split at a new price of £2.

This leads to a challenge for a kdb+ developer to return historical stats in terms of today's stock structure.

Table 3: Stock Split formula for Price and Size adjustments

Action	Price Adjustment	Size Adjustment
Stock Split	price%adj	size*adj

Imagine a stock (XYZ.L) that has gone through 2 stock splits in its lifetime. First a ten-for-one split effective from 01-Oct-2010. Then again on the 16-Feb-2012 a further two-for-one split took effect.

Table 4: XYZ.L Stock Split history

Effective-Date	Type	Event
01-Oct-2010	Stock Split	10 for 1 (XYZ.L)
16-Feb-2012	Stock Split	2 for 1 (XYZ.L)

Typical source data:

```
q)scrTbl:([]sym:`XYZ.L;date:2010.10.01
2012.02.16;action:`SS;adj:`float$10 2);
q)scrTbl
sym   date       action adj
---------------------------
XYZ.L 2010.10.01 SS     10
XYZ.L 2012.02.16 SS     2
```

Table 1 showed that there are inconsistencies in how typical source data are applied for different types of actions. For example, price is divided by the adjustment for stock split while for cash dividend it is multiplied. In the following section an adjust source `(adjscr)` function is defined that addresses this and will produce a consistent `scrTbl` table for any corporate action. It will provide adjustments for both size (`sadj`) and price (`padj`) and will also ensure that these adjustments will always need to be multiplied. This will become important when adjusting for more than one type of corporate action at a time.

```
//Adjust scrTbl function, to be consistent for any action
q)adjscr:{[scrTbl] scrTbl:`sym`date`action`padj xcol update sadj:1%adj
from scrTbl where action in `SS; scrTbl:update padj:1%padj from scrTbl
where not action in `SS;update sadj:1^sadj from scrTbl};

q)scrTbl:adjscr[scrTbl]
q)scrTbl
sym   date       action padj sadj
---------------------------------
XYZ.L 2010.10.01 SS     10   0.1
XYZ.L 2012.02.16 SS     2    0.5
```

Again, we are only interested in storing data points of when changes took place. Therefore in a temporal table we need:

Table 5: XYZ.L temporal table for Size adjustments

Effective-Date	Type	Price_Adjustment	Size_Adjustment
16-Feb-2012	asof	1	1
01-Oct-2010	asof	0.5	2
01-Oct-2010	before	0.05	20

Transforming the source data table can be done in the following way

```
//calculating adjustment factors
q)afact:{reverse reciprocal prds 1,reverse x}
```

```
q)ca:{[cact]`s#2!ungroup update date:(0Nd,'date),padj:afact each
padj,sadj:afact each sadj from `sym xgroup `sym`date xasc ``action _
select from scrTbl where date<=.z.d, action in cact}

q)adjTbl:ca[`SS]
sym   date       | padj sadj
---------------- | ---------
XYZ.L            | 0.05 20
XYZ.L 2010.10.01 | 0.5  2
XYZ.L 2012.02.16 | 1    1
```

Raw without stock split adjustment:

```
q).Q.view 2010.06.24 2011.07.12 2014.01.10
q)select sum size,avg price by sym,date from trade where sym=`XYZ.L
sym   date       | size  price
---------------| --------------
XYZ.L 2010.06.24| 1838  293.3333
XYZ.L 2011.07.12| 2911  553.8033
XYZ.L 2014.01.10| 27159 1478.329
```

Enriched data with stock split adjustments:

```
//adjscr allows us to have adjAgg constant for all actions
q)adjAgg:`size`price!((*;`size;`sadj);(*;`price;`padj));
q)adjAgg
size | * `size  `sadj
price| * `price `padj

q)adj:{[cact;res] res:update padj:1^padj,sadj:1^sadj from
aj[`sym`date;res;$[not count  adjTbl:ca[cact];:res;adjTbl]];:`padj`sadj
_ 0!![res;();0b;](c where (c:cols res) in key adjAgg)#adjAgg}

q)select sum size,avg price by sym,date from adj[`SS;] select from
trade where sym in `XYZ.L
sym   date       | size  price
---------------| --------------
XYZ.L 2010.06.24| 36760 14.66667
XYZ.L 2011.07.12| 5822  276.9017
XYZ.L 2014.01.10| 27159 1478.329
```

One can see that, for trades occurring after the latest stock split, size remains the same. Trades on 12-Jul-2011 were before the last stock split but after the first, therefore, trade sizes have increased by a factor of 2, as 1 share then represents 2 shares today. Likewise 24-Jun-2010 was before any splits in the stock and size adjustment is by a factor of 20, as 1 share then represents 20 shares at present. Price adjustments also appear to ensure trade value remains constant.

7 CASH DIVIDEND

Say a stock that has decided to pay a £0.05 dividend per share is trading at £7.00 prior to its ex-dividend date (Ex-date).

A shareholder with 10,000 shares has a total value prior to the ex-date of 10,000*£7.00=£70,000. After the ex-date, the price should theoretically drop to £6.95. Yet, the investor's total value is maintained as 10,000*£6.95=£69,500 + £500 cash.

The adjustment factor is determined by:

```
q)cd_padj:{[P;X] (P-X)%P}
q)cd_padj[7.00;0.05]
0.9928571

// cross check of calculation
q)7.00*0.9928571
6.95
```

Users may request that historical price values be adjusted. However, size remains the same.

Table 6: Cash Dividend formula for Price and Size adjustments

Action	Price Adjustment	Size Adjustment
Cash Dividend	price*adj	(no change)

Let's take a look at a real world example for BP.L;

Table 7: BP.L Cash Dividend history

Date	Type	Event
04-Feb-2014	Results: 4Q 2013 results and dividend announcement	Dividend of 5.7065 pence per share
11-Feb-2014	Close Price	491.75
12-Feb-2014	Ex-date	Fourth Quarter ex-dividend
12-Feb-2014	Close Price	487.05
28-Mar-2014	Dividend	Fourth Quarter payment date

Therefore the corresponding price adjustment is as follows:

```
q)cd_padj[491.75;5.7065]
0.9883955
```

Similar to the above stock split, typical source data is provided and can be transformed to be a temporal `adjTbl` with the `adjscr`, `ca` and `afact` functions.

```
q)scrTbl:([] sym:(),`BP.L;date:2014.02.12;action:`CD;adj:0.9883955)
q)scrTbl
sym  date       action adj
--------------------------------
BP.L 2014.02.12 CD     0.9883955

q)scrTbl: adjscr[scrTbl]
q)scrTbl
sym  date       action padj     sadj
-----------------------------------
BP.L 2014.02.12 CD     1.011741 1

q)adjTbl:ca[`CD]
sym  date      | padj      sadj
---------------| --------------
BP.L           | 0.9883955 1
BP.L 2014.02.12| 1         1
```

A typical query for last price without adjustment applied

```
q).Q.view 2014.02.11 2014.02.12
q)0!select last price,last size by sym,date from trade where
sym=`BP_.L
sym   date       price  size
------------------------------
BP.L 2014.02.11 491.75 6432023
BP.L 2014.02.12 487.05 6852708
```

Same query but now with Cash Dividend adjustments

```
q).Q.view 2014.02.11 2014.02.12
q)adj[`CD;] select last price,last size by sym,date from trade where
sym=`BP.L
sym   date       price    size
--------------------------------
BP.L 2014.02.11 486.0435 6432023
BP.L 2014.02.12 487.05   6852708

// cross check of calculation
q)491.75*0.9883955
486.0435
```

The correct price adjustment has been applied for date prior to the ex-dividend date.

8 COMBINING ADJUSTMENTS

The framework outlined in this paper gives the Users an option of which corporate action adjustments, if any, to apply. In the following example a test trade table is created to aid the example.

```
q)trade:([]date:2013.01.01 2013.04.01 2013.07.01
2014.01.01;sym:4#`VOD.L;price:4#10;size:4#1000)
q)trade
date       sym   price size
--------------------------
2013.01.01 VOD.L 10    1000
2013.04.01 VOD.L 10    1000
2013.07.01 VOD.L 10    1000
2014.01.01 VOD.L 10    1000
```

Sample source data

```
q)scrTbl:([] sym:`VOD.L;date:2012.05.01 2013.02.01 2013.07.01
2013.11.01 2014.06.01;action:`SS`CD`CD`SS`CD;adj:`float$2 0.95 0.97 10
0.96)
q)scrTbl
sym   date       action adj
--------------------------
VOD.L 2012.05.01 SS     2
VOD.L 2013.02.01 CD     0.95
VOD.L 2013.07.01 CD     0.97
VOD.L 2013.11.01 SS     10
VOD.L 2014.06.01 CD     0.96

q)scrTbl:adjscr[scrTbl]
sym   date       action padj     sadj
-----------------------------------
VOD.L 2012.05.01 SS     2        0.5
VOD.L 2013.02.01 CD     1.052632 1
VOD.L 2013.07.01 CD     1.030928 1
VOD.L 2013.11.01 SS     10       0.1
VOD.L 2014.06.01 CD     1.041667 1
```

```
 //No adjustments applied
q)adj[`;]select from trade
date       sym   price size
--------------------------
2013.01.01 VOD.L 10    1000
2013.04.01 VOD.L 10    1000
2013.07.01 VOD.L 10    1000
2014.01.01 VOD.L 10    1000
```

```
//Stock Splits Only
q)adj[`SS;]select from trade
date       sym   price size
---------------------------
2013.01.01 VOD.L 1     10000
2013.04.01 VOD.L 1     10000
2013.07.01 VOD.L 1     10000
2014.01.01 VOD.L 10    1000

//Cash Dividend Only
q)adj[`CD;]select from trade
date       sym   price size
---------------------------
2013.01.01 VOD.L 9.215 1000
2013.04.01 VOD.L 9.7   1000
2013.07.01 VOD.L 10    1000
2014.01.01 VOD.L 10    1000

//Stock Split and Cash Dividend combined
q)adj[`SS`CD;]select from trade
date       sym   price  size
----------------------------
2013.01.01 VOD.L 0.9215 10000
2013.04.01 VOD.L 0.97   10000
2013.07.01 VOD.L 1      10000
2014.01.01 VOD.L 10     1000
```

From adjusting the standard source data in `adjscr` function we can see that adjustment factors for any action are simply multiplied together to give the combined adjustment factor.

9 CONCLUSION

This whitepaper introduced a method for applying corporate action adjustments to equity tick data on the fly. The basic use of temporal data was outlined, highlighting the power of the `s# attribute. After this, we explained the role of reference data and its importance in a kdb+ system. With this knowledge we laid out an example of a simple gateway request to show how we could aggregate tick data across a date range in which a name change had taken place. Later in the paper, stock splits and cash dividends were also covered.

Overall, this paper provides an insight into the capabilities of kdb+ regarding varies types of corporate actions. It may be used as a framework for firstly dealing with name changes at a gateway level and secondly for handling stock splits and cash dividends at a database level. However it is not limited by these examples, and can also be used for other actions such as stock dividends, rights issues and spin-offs.

All tests were run using kdb+ version 3.1 (2014.02.08)