it's about time

Developer Brief

Kdb+ and FIX Messaging

Author:

Damien Barker is a Financial Engineer who has worked as a consultant for some of the world's largest financial institutions. Based in London, Damien is currently working on trading and analytics application at a US investment bank.

Table of Contents

INTRODUCTION

Electronic trading volumes have increased significantly in recent years, prompting financial institutions, both buy and sell side, to invest in increasingly sophisticated Order Management Systems (OMS). OMS's efficiently manage the execution of orders using a set of pre-defined conditions to obtain the best price of execution. OMS's typically use the Financial Information eXchange (FIX) protocol, which has become the industry standard for electronic trade messaging since it was first developed in 1992.

The demand for post trade analytics and compliance requirements (for example proving a client order was filled at the best possible price) provide a need to retain all the FIX messages produced by an OMS. For large volumes of data this can prove extremely challenging; however kdb+ provides an ideal platform to capture and process the FIX messages. It allows efficient querying of large volumes of historical data, and in conjunction with a kdb+ tick set up can produce powerful real time post trade analytics for the front office users.

This paper will introduce the key steps to capture a FIX message feed from an OMS, and understand the data contained within each message. We produce an example that demonstrates a kdb+ set up that captures a FIX feed and produces a final order state table.

All tests were run using kdb+ version 3.1 (2013.12.27)

1. FIX MESSAGES

1.1. FIX message format

FIX messages consist of a series of key-value pairs that contain all the information for a particular state of a transaction. Each tag relates to a field defined in the FIX specification for a given system. In FIX4.4, tags 1-956 are predefined and values for these fields must comply with the values outlined in the FIX protocol. Outside of this range custom fields may be defined; these may be unique to the trading system or firm. Some common tags are given in the table below.

Tag	Field
1	Account
6	AvgPx
8	BeginString
9	BodyLength
10	CheckSum
11	ClOrdID
12	Commission
13	CommType
14	CumQty
15	Currency
17	ExecID
19	ExecRefID
21	HandlInst
29	LastCapacity
30	LastMkt
31	LastPx
32	LastQty
34	MsgSeqNum
35	MsgType
37	OrderID
38	OrderQty
39	OrderStatus
49	SenderCompID
52	SendingTime
56	TargetCompID
151	LeavesQty

Table 1: Some common FIX tags and respective fields.

A FIX message is comprised of a header, body and trailer. All messages must begin with a header consisting of BeginString (8), BodyLength (9), MsgType (35), SenderCompID (49), TargetCompID (56), MsgSeqNum (34) and SendingTime (52) tags. BeginString states the FIX version used, BodyLength is a character count of the message and MsgType gives the type of message, for instance New Order, Execution Report etc. SenderCompID and TargetCompID contain information on the firms sending and receiving the message respectively. The message must finish with tag CheckSum (10); this is the count of all characters from tag 35 onwards including all delimiters. The body of the message consists of all other relevant tags, depending on the message type FIX messages are delimited by ASCII SOH (Start of Heading), however as this in unprintable we will us "|" as a delimiter in this paper. Below is an example of some FIX messages that we will use for this whitepaper.

```
8=FIX.4.4|9=178|35=D|49=A|56=B|1=accountA|6=0|11=00000001|12=0.0002|13=2|14=|15=GBp|17=|1
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=0|11=00000001|12=0.0002|13=2|14=|15=GBp|17=|1
9=|21=|29=|30=|31=|32=|37=|38=10000|39=0|41=|44=|48=VOD.L|50=AB|52=20131218-
09:01:13|54=1|55=VOD|58=|59=1|60=20131218-09:01:13|10=168
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=229.5|11=00000001|12=|13=|14=1500|15=GBp|17=1
00000001|19=|21=1|29=1|30=XLON|31=229.5|32=1500|37=|38=10000|39=1|41=|44=|48=VOD.L|50=AB|
52=20131218-09:02:01|54=1|55=VOD|58=|59=1|60=20131218-09:02:01|10=193
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=229.6125|11=00000001|12=|13=|14=6000|15=GBp|1
7=100000002|19=|21=1|29=1|30=XLON|31=229.65|32=4500|37=|38=10000|39=1|41=|44=|48=VOD.L|50
=AB|52=20131218-09:01:03|54=1|55=VOD|58=|59=1|60=20131218-09:01:03|10=197
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=229.6353846|11=00000001|12=|13=|14=6500|15=GB
p|17=100000003|19=|21=1|29=1|30=XLON|31=229.91|32=500|37=|38=10000|39=1|41=|44=|48=VOD.L|
50=AB|52=20131218-09:01:14|54=1|55=VOD|58=|59=1|60=20131218-09:01:14|10=199
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=229.7496933|11=00000001|12=|13=|14=8150|15=GB
p|17=100000004|19=|21=1|29=1|30=XLON|31=230.2|32=1650|37=|38=10000|39=1|41=|44=|48=VOD.L|
50=AB|52=20131218-09:01:15|54=1|55=VOD|58=|59=1|60=20131218-09:01:15|10=199
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=229.6295|11=00000001|12=|13=|14=10000|15=GBp|
17=100000005|19=|21=1|29=1|30=XLON|31=229.1|32=1850|37=|38=10000|39=2|41=|44=|48=VOD.L|50
=AB|52=20131218-09:01:46|54=1|55=VOD|58=|59=1|60=20131218-09:01:46|10=197
...
```

1.2. Feed Handler

A feed handler may be used to deliver the messages to kdb+. The feed handler should receive the flow of FIX messages from the OMS, parse the messages to extract the required fields and send them to the kdb+ tickerplant. Feed handlers are generally written in Java or C++ and are widely available, for example from Kx.

For the example provided in this whitepaper we load a file of FIX messages to a q feed handler. Our feed handler reads each FIX message from the file, extracts the tags and casts to the desired q type.

The FIX tag and field names are stored in a FIX specification which should include all possible tags from the OMS including any custom tags unique to our setup. The FIX specification allows us to create reference dictionaries to map the tags to the correct column names.

```
q)fixTagToName
1 | Account
6 | AvgPx
8 | BeginString
11| ClOrdID
12| Commission
13| CommType
14| CumQty
...
```

We include functions to parse the FIX messages and extract the desired tags. These functions can also be included in the RDB to allow us to extract additional information from the raw FIX message for fields not included in our schema.

```
getAllTags:{[msg](!)."S=|"0:msg}
getTag:{[tag;msg](getAllTags[msg])[tag]}
```

We read the file containing the FIX messages, parse each message to extract the information and flip into a table.

```
fixMsgs:read0 hsym `$path,"/fixMsgs.txt";
fixTbl:(uj/){flip fixTagToName[key d]!value enlist each d:getAllTags x} each
fixMsgs;
```

We need to extract the desired fields and cast to the correct type. Functions are used to match the schema of our FIX messages to a pre defined schema in the rdb.

```
colConv:{[intype;outtype]
       $[(intype in ("C";"c"))&(outtype in ("C";"c"));eval';
         (intype in ("C";"c"));upper[outtype]$;
         (outtype in ("C";"c"));string;
         upper[outtype]$string
         ]
       };

matchToSchema:{[t;schema]
       c:inter[cols t;cols schema];
       metsch:exec "C"^first t by c from meta schema;
       mett:exec "C"^first t by c from meta t;
       ?[t;();0b;c!{[y;z;x](colConv[y[x];z[x]];x)}[mett;metsch] each c]
       };
```

We add the full FIX message to the table as a column of strings. This ensures no data is lost from the original message that was received and information can easily be obtained when necessary. The FIX message is then sent to the tickerplant.

```
genFixMsgs:{[]
       //read fix message file
       fixMsgs:read0 hsym `$path,"/fixMsgs.txt";
       // extract each tag, map to name and convert to table
       fixTbl:(uj/){flip fixTagToName[key d]!value enlist each d:getAllTags x}
each fixMsgs;
       // cast fixTbl to correct types
       t:matchToSchema[fixTbl;fixmsgs];
       // Add the original fix message as a column
       t:update FixMessage:fixMsgs from t;
       :t;
       }

runFIXFeed:{[]
       t:genFixMsgs[];
       tick_handle["upd";`fixmsgs;t];
       }
```

1.3. FIX Tags

In this section we look at some of the most important FIX messages.

1.3.1. MsgType

Msgtype (tag 35) is a required field in the FIX message. It defines the type of message received, for example order, execution, allocation, heartbeat, Indication of Interest etc. For the purpose of this paper we limit ourselves to handling the following msgtypes, which will be most common from an OMS.

```
8 = Execution Report <8>
D = Order - Single <D>
G = Order Cancel/Replace Request <G>
F = Order Cancel Request <F>
```

Every time we receive a new order, the first message should contain MsgType=D. We should only receive one ‘D’ message per order. If this has to be amended at any stage we should receive an order replace request, (MsgType=G), to replace the original order.

As the order executes we will receive Execution Reports (MsgType= 8) for each execution. These are linked back to the original order through one of the ID fields, generally ClOrdID. The execution message contains some important updates to the overall state of the order, particularly CumQty and AvgPx.

If the order is cancelled before the full order quantity is executed, a Cancel Request (MsgType= F) message is sent. This can be rejected with an Order Cancel Reject (MsgType= 9) and the order will continue to execute. It is important to note that this only cancels any outstanding shares not yet executed and not the full order.

1.3.2. OrdStatus

OrdStatus tells us the current state the order is in. It is an important indicator in cases where the order has not been filled, showing if it is still executing, cancelled, done for the day (for multi day orders) etc. The valid values are:

```
0 = New
1 = Partially filled
2 = Filled
3 = Done for day
4 = Canceled
5 = Replaced
6 = Pending Cancel/Replace
7 = Stopped
8 = Rejected
9 = Suspended
A = Pending New
B = Calculated
C = Expired
```

1.3.3. Commission

In FIX there are two fields needed to obtain the correct commission on an order: Commission (12) and CommType (13). Commission and CommType both return a numerical value; the latter a number defined as follows:

```
1 = per unit (implying shares, par, currency, etc)
2 = percentage
3 = absolute (total monetary amount)
4 = (for CIV buy orders) percentage waived - cash discount
5 = (for CIV buy orders) percentage waived - enhanced units
6 = points per bond or contract
```

We will only be concerned with the first three cases (from the list above) for our example in this paper. We define a function to calculate the commission value:

```
calcComm:{[comval;comtyp;px;qty]
        $[comtyp=`1;
          comval*qty;
          comtyp=`2;
          comval*px*qty;
          comtyp=`3;
          comval]
        }
```

1.3.4. LastCapacity

LastCapacity tells us the broker capacity in the execution. It indicates whether a fill on an order was executed as principal or agency. A principal transaction occurs when the broker fills the part of the order from its own inventory while an agency transaction involves the broker filling the order on the market. It is vital in calculating benchmarks or client loss ratios to distinguish between principal and agency flow. The valid values are:

```
1 = Agent
2 = Cross as agent
3 = Cross as principal
4 = Principal
```

2. EXAMPLE –ORDER STATE

2.1. Approach

Our aim is to create a final state table for all orders. In our example the RDB will subscribe to the tickerplant, receive all the messages and generate an order state. For large volumes this could be separated in two processes: the RDB should just capture all messages from the tickerplant and store them in a single table while a separate process can then be set up to subscribe to this table and generate the order and execution tables. This example details an approach to handling the most common messages expected from an OMS. The standard fields expected from an OMS are included, along with some derived fields.

2.2. Schema

We set up the schema below for the `fixmsgs` table. It contains columns for every tag defined by our FIX spec, and well as a column called FixMessage, which contains the full FIX message as a string, and a column containing the tickerplant time. The FixMessage field is important as any information in the FIX message missing from our schema can still be extracted.

```
fixmsgs:([]
        Account:`$();
        AvgPx:`float$();
        ClOrdID:();
        Commission:`float$();
        CommType:`$();
        CumQty:`float$();
        Currency:`$();
        ExecID:();
        ExecRefID:();
        HandlInst:`$();
        LastCapacity:`$();
        LastMkt:`$();
        LastPx:`float$();
        LastQty:`int$();
        LeavesQty:`float$();
        MsgType:`$();
        OrderID:();
        OrderQty:`int$();
        OrdStatus:`$();
        OrigClOrdID:();
        Price:`float$();
        SecurityID:`$();
        SenderSubID:`$();
        SendingTime:`datetime$();
        Side:`$();
        Symbol:`$();
        Text:();
        TimeInForce:`$();
        TransactTime:`datetime$();
        FixMessage:();
        Time:`datetime$()
        )
```

The `order` schema contains the core fields from the fixmsgs schema as well as derived fields: OrderTime and AmendTime. These fields are not included in the FIX spec but will be required by end users and as such are added in the RDB. The `order` table is keyed on OrderID. In practise a ClOrderID or a combination of ClOrderID and OrigClOrdID may be needed. If an order is cancelled and replaced the OrigClOrdID contains the ClOrderID of the previous version of the order. Only the final version is required in the final state, so we need to track these orders.

```
order:([OrderID:()]
       ClOrdID:();
       OrigClOrdID:();
       SecurityID:`$();
       Symbol:`$();
       Side:`$();
       OrderQty:`int$();
       CumQty:`float$();
       LeavesQty:`float$();
       AvgPx:`float$();
       Currency:`$();
       Commission:`float$();
       CommType:`$();
       CommValue:`float$();
       Account:`$();
       MsgType:`$();
       OrdStatus:`$();
       OrderTime:`datetime$();
       TransactTime:`datetime$();
       AmendTime:`datetime$();
       TimeInForce:`$()
       )
```

2.3. Processing Orders

We define the following `upd` function on the rdb:

```
upd:{[t;x]
       t insert x;
       x:`TransactTime xasc x;
       updNewOrder[`order;select from x where MsgType in `D];
       x:select from x where not MsgType in `D;
       {$[(first x`MsgType)=`8;
              updExecOrder[`order;x];
          (first x`MsgType)=`G;
              updAmendOrder[`order;x];
          (first x`MsgType) in `9`F;
              updCancelOrder[`order;x];
              :()];
       } each (where 0b=(=':)x`MsgType) cut x
       }
```

And a series of functions to handle each MsgType:

```
updNewOrder:{[t;x] ...}

updAmendOrder:{[t;x] ...}

updCancelOrder:{[t;x]  ...}

updExecOrder:{[t;x] ...}
```

We first ensure the messages are ordered correctly, according to TransactTime. This is so the messages are processed in the order they were generated, which is important when looking at the final state of an order.

New orders are processed first since we should only ever receive one ‘D’ message per order.

```
updNewOrder[`order;select from x where MsgType in `D];
```

For all subsequent updates for each order we need to ensure that all amendments, cancellations and executions are handled in the correct order. We separate the remaining messages into chunks of common MsgType and process each chunk sequentially. This is particularly important in the case where we receive an amended order in the middle of a group of executions. This is essential for the final order state to show the correct TransactTime, MsgType and OrdStatus of the final order.

```
{$[(first x`MsgType)=`8;
       updExecOrder[`order;select from x where MsgType in `8];
       updAmendOrder[`order;select from x where MsgType in `G`F]]
}each(where 0b=(=':)x`MsgType) cut x
```

2.4. New Orders

Whenever a new order is received we must ensure it is entered into our final state table. We define the following function:

```
updNewOrder:{[t;x]
       x:update OrderTime:TransactTime from x;
       t insert inter[cols t;cols x]#x;
       }
```

For each order, users will want to know the time the order was received. TransactTime is not sufficient here, since it will be overwritten in the final state table by subsequent updates. We introduce a custom field called OrderTime. This contains the TransactTime of the New Order message and will not be updated by any other messages.

For a new order message we want to insert all the columns provided in the FIX message. We extract all common columns between our message and the schema. We also note the order table is keyed on OrderID .

```
t insert inter[cols t;cols x]#x;
```

We receive the following new order FIX messages from the OMS

```
8=FIX.4.4|9=178|35=D|49=A|56=B|1=accountA|6=0|11=0000001|12=0.0002|13=2|14=|15=
GBp|17=|19=|21=|29=|30=|31=|32=|151=10000|37=00000001|38=10000|39=|41=|44=|48=V
OD.L|50=AB|52=20131218-09:01:00|54=1|55=VOD|58=|59=1|60=20131218-
09:01:00|10=184
8=FIX.4.4|9=178|35=D|49=A|56=B|1=accountB|6=0|11=0000002|12=0.0002|13=2|14=|15=
GBp|17=|19=|21=|29=|30=|31=|32=|151=4000|37=00000002|38=4000|39=|41=|44=|48=RIO
.L|50=AD|52=20131218-10:24:07|54=2|55=RIO|58=|59=1|60=20131218-10:24:07|10=182
8=FIX.4.4|9=178|35=D|49=A|56=B|1=accountA|6=0|11=0000003|12=0.0002|13=2|14=|15=
GBp|17=|19=|21=|29=|30=|31=|32=|151=20100|37=00000003|38=20100|39=|41=|44=|48=B
ARC.L|50=AR|52=20131218-11:18:22|54=1|55=BARC|58=|59=1|60=20131218-
11:18:22|10=186
8=FIX.4.4|9=178|35=D|49=A|56=B|1=accountC|6=0|11=0000004|12=0.0002|13=2|14=|15=
```

The order state shows a series of unfilled orders. The CumQty and OrdStatus are initially null, as they are not present on the new order message. They will be populated by subsequent execution updates.

```
q)select OrderID, MsgType, OrdStatus, SecurityID, Account, OrderQty, CumQty,
Commission from order
OrderID    MsgType OrdStatus SecurityID Account  OrderQty CumQty Commission
---------------------------------------------------------------------------
"00000001" D                 VOD.L      accountA 10000           0.0002
"00000002" D                 RIO.L      accountB 4000            0.0002
"00000003" D                 BARC.L     accountA 20100           0.0002
"00000004" D                 EDF.PA     accountC 15000           0.0002
"00000005" D                 VOD.L      accountD 3130            0.0002
```

2.5. Amendments and Cancellations

Any value of the order may be amended by sending a message with MsgType=G. This could reflect a correction to commission value, a change in the order quantity etc.

The function to update amendments differs slightly from that for new orders. A field to display the latest amend time is added - this provides the end user with the TransactTime of the last change to the order. Every amend message should have been preceded by a new order message, so the amendment is upserted (rather than inserted) into the `order` state table. A production system could include some sanity checks to ensure we have received an order for any amendment.

```
updAmendOrder:{[t;x]
       x:update AmendTime:TransactTime from x;
       t upsert inter[cols t;cols x]#x;
       }
```

The following example shows an update to the commission value. We have received a new order with commission specified in percent. An update modifies this to an absolute value. The amendment is reflected in the order state and the total value of the commission is extracted using the calcComm function outlined earlier.

```
8=FIX.4.4|9=178|35=G|1=accountA|6=253.8854627|11=0000003|12=700|13=3|14=20100|
15=GBp|17=|19=|21=|29=|30=|31=|32=|151=0|37=00000003|38=20100|39=2|41=|44=|48=
BARC.L|50=AR|52=20131218-16:33:12|54=1|55=BARC|58=|59=1|60=20131218-
16:33:12|10=195
```

```
q)select OrderID,MsgType,Commission,CommType from fixmsgs where OrderID like
"00000003",MsgType in `D`G`F`9
OrderID    MsgType Commission CommType
--------------------------------------
"00000003" D       0.0002     2
"00000003" G       700        3

q)select OrderID, MsgType, CumQty, AvgPx, Commission, CommType,
CommValue:calcComm'[Commission;CommType;AvgPx;CumQty] from order where OrderID
like "00000003"
OrderID    MsgType CumQty AvgPx    Commission CommType CommValue
----------------------------------------------------------------
"00000003" G       20100  253.8855 700        3        700
```

An Order Cancel Request (MsgType=F) indicates the cancellation of any outstanding unfilled order quantity. It can be rejected with an Order Cancel Reject (MsgType=9). Along with the order cancel message we should get an Execution Report to confirm the cancellation, with OrdStatus=4 to indicate the order is cancelled. As such this may be sufficient to indicate to end users a cancellation, with the Order Cancel Request and Order Cancel Reject omitted from the Order State logic. For this example we upsert only the MsgType and AmendTime from the cancel messages.

```
updCancelOrder:{[t;x]
       x:update AmendTime:TransactTime from x;
       t upsert `OrderID xkey select OrderID,MsgType,AmendTime from x;
       }
```

When the order is cancelled we receive the following FIX message to request a cancel. The `order` table shows an order that is not fully filled, but cancelled with nothing left to fill.

```
8=FIX.4.4|9=178|35=F|1=accountC|6=25.3156|11=0000004|12=|13=|14=12500|15=EUR|
17=100000018|19=|21=3|29=1|30=XPAR|31=0|32=0|151=2500|37=00000004|38=15000|39
=|41=|44=|48=EDF.PA|50=CD|52=20131218-
13:33:11|54=1|55=EDF|58=|59=1|60=20131218-13:33:11|10=206
```

```
q)select OrderID,MsgType,OrdStatus,OrderQty,CumQty from order where
MsgType=`F
OrderID    MsgType OrdStatus OrderQty CumQty
--------------------------------------------
"00000004" F       1         15000    12500
```

The Execution Report should follow the cancel request to confirm the order has been cancelled and update the status of the order. The confirmation updates the OrdStatus and changes the LeavesQty to reflect the cancellation. We will see how to handle the execution report in the next section.

```
8=FIX.4.4|9=178|35=8|1=accountC|6=25.3156|11=0000004|12=|13=|14=12500|15=EUR|
17=100000018|19=|21=3|29=1|30=XPAR|31=0|32=0|151=2500|37=00000004|38=15000|39
=4|41=|44=|48=EDF.PA|50=CD|52=20131218-
13:33:11|54=1|55=EDF|58=|59=1|60=20131218-13:33:11|151=0|10=210
```

```
q)select OrderID,MsgType,OrdStatus,OrderQty,CumQty,LeavesQty from order where
MsgType=`F
OrderID    MsgType OrdStatus OrderQty CumQty LeavesQty
------------------------------------------------------
"00000004" 8       4         15000    12500  0
```

2.6. Execution Reports

Execution Reports (MsgType=8) are sent every time there is a change in the state of the order. We are only interested in certain fields from execution messages. In our case we want to update OrderID, MsgType, OrdStatus, LastQty, LastPx, AvgPx, CumQty, LeavesQty and LastMkt in the `order` table. AvgPx, CumQty and LeavesQty are derived columns, giving the latest information for the full order. They should be calculated by the OMS and upserted straight into the order state. The LastQty contains the quantity executed on the last fill, and LastPx the price of the last fill. It is important to always take the latest OrdStatus from the execution messages, this ensures the order state always reflects the current state of the order.

```
updExecOrder:{[t;x]
       t upsert select OrderID, MsgType, OrdStatus, LastQty, LastPx, AvgPx,
CumQty, LeavesQty, LastMkt from x;
       }
```

The following messages show all the execution reports received for one order. The first message is a confirmation of the new order and sets the OrdStatus to 0. The subsequent messages show each fill on the order. The OrdStatus is set to 1 for each fill until order is complete, when we receive an OrdStatus of 2.

```
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=0|11=0000001|12=0.0002|13=2|14=|15=
GBp|17=|19=|21=|29=|30=|31=|32=|151=10000|37=00000001|38=10000|39=0|41=|44=|48=
VOD.L|50=AB|52=20131218-09:01:00|54=1|55=VOD|58=|59=1|60=20131218-
09:01:00|10=185
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=0|11=0000001|12=0.0002|13=2|14=|15=
GBp|17=|19=|21=|29=|30=|31=|32=|151=10000|37=00000001|38=10000|39=0|41=|44=|48=
VOD.L|50=AB|52=20131218-09:01:03|54=1|55=VOD|58=|59=1|60=20131218-
09:01:03|10=185
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=229.5|11=0000001|12=|13=|14=1500|15
=GBp|17=10000001|19=|21=1|29=1|30=XLON|31=229.5|32=1500|151=8500|37=0000001|3
8=10000|39=1|41=|44=|48=VOD.L|50=AB|52=20131218-
09:01:11|54=1|55=VOD|58=|59=1|60=20131218-09:01:11|10=209
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=229.6125|11=0000001|12=|13=|14=6000
|15=GBp|17=100000002|19=|21=1|29=1|30=XLON|31=229.65|32=4500|151=4000|37=000000
01|38=10000|39=1|41=|44=|48=VOD.L|50=AB|52=20131218-
09:01:13|54=1|55=VOD|58=|59=1|60=20131218-09:01:13|10=213
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=229.6353846|11=0000001|12=|13=|14=6
500|15=GBp|17=100000003|19=|21=1|29=1|30=XLON|31=229.91|32=500|151=3500|37=0000
0001|38=10000|39=1|41=|44=|48=VOD.L|50=AB|52=20131218-
09:01:14|54=1|55=VOD|58=|59=1|60=20131218-09:01:14|10=215
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=229.7496933|11=0000001|12=|13=|14=8
150|15=GBp|17=100000004|19=|21=1|29=1|30=XLON|31=230.2|32=1650|151=1850|37=0000
0001|38=10000|39=1|41=|44=|48=VOD.L|50=AB|52=20131218-
09:01:15|54=1|55=VOD|58=|59=1|60=20131218-09:01:15|10=215
8=FIX.4.4|9=178|35=8|49=A|56=B|1=accountA|6=229.6295|11=0000001|12=|13=|14=1000
0|15=GBp|17=100000005|19=|21=1|29=1|30=XLON|31=229.1|32=1850|151=0|37=00000001|
38=10000|39=2|41=|44=|48=VOD.L|50=AB|52=20131218-
09:01:46|54=1|55=VOD|58=|59=1|60=20131218-09:01:46|10=210
```

The final table shows this order (OrderID = “00000001”) as fully filled. We can also see the cancelled order (“00000004”) reflected with OrdStatus=4. The order we amended (“00000003”) shows an amended commission value of 700.

```
q)select OrderID, SecurityID, Side, MsgType, OrdStatus, OrderQty, CumQty,
AvgPx, CommValue:calcComm'[Commission;CommType;AvgPx;CumQty] from order
OrderID    SecurityID Side MsgType OrdStatus OrderQty CumQty AvgPx    CommValue
-------------------------------------------------------------------------------
"00000001" VOD.L      1    8       2         10000    10000  229.6295 459.259
"00000002" RIO.L      2    8       2         4000     400    3253.537 260.283
"00000003" BARC.L     1    G       2         20100    20100  253.8855 700
"00000004" EDF.PA     1    8       4         15000    12500  25.3156  63.289
"00000005" VOD.L      2    8       2         3130     3130   229.7559 143.8272
```

3. CONCLUSION

This paper has provided a guide to working with FIX messages in kdb+, focusing primarily on capturing messages from an OMS. We focused on some key FIX fields to provide an understanding of the valid entries and an insight into how they should be handled. A comprehensive list of all fields for each FIX version can be found at http://fixwiki.org/fixwiki/FIXwiki.

It is an essential requirement to be able to view the current and final state of each order received from the OMS. We provided an example to show how to generate an order state. This process can be extended to also derive the latest execution state. Additional fields need to be extracted to identity the execution type, amendments, cancellations etc, similar to the order state. An execution state enables complex analytics, including risk benchmarking and transaction cost analysis to be computed efficiently on every execution.

All tests were run using kdb+ version 3.1 (2013.12.27)