|  |
| --- |
| First Derivatives |
| KDB+ Tick 2.3 |
| A developers guide to KDB+ Tick and a breakdown of all functionality used |

|  |
| --- |
| Ian Wasson, First Derivatives plc  7/17/2008 |

Table of Contents

[What is a Tick? 3](#_Toc204425777)

[Components of Tick: 5](#_Toc204425778)

[Starting a Tickerplant 6](#_Toc204425779)

[Creating a Real Time Database Subscriber 6](#_Toc204425780)

[Creating a Chained Tickerplant 6](#_Toc204425781)

[The Scripts 7](#_Toc204425782)

[tick/u.k 7](#_Toc204425783)

[The functions: 7](#_Toc204425784)

[.u.init (initialize) 7](#_Toc204425785)

[.u.del (delete) 7](#_Toc204425786)

[.u.sel (select) 8](#_Toc204425787)

[.u.pub (publish) 8](#_Toc204425788)

[.u.add (addition) 8](#_Toc204425789)

[.u.sub (subscription) 9](#_Toc204425790)

[.u.end (end of day) 9](#_Toc204425791)

[tick/u.q 9](#_Toc204425792)

[.u.end (end of day – version 2) 9](#_Toc204425793)

[tick.k 10](#_Toc204425794)

[Explanation 10](#_Toc204425795)

[The functions 11](#_Toc204425796)

[.u.ld (log directory) 11](#_Toc204425797)

[.u.tick (tickerplant) 11](#_Toc204425798)

[.u.endofday (local end of day) 12](#_Toc204425799)

[.u.ts (timed function) 12](#_Toc204425800)

[.z.ts (with bulking) 12](#_Toc204425801)

[.u.upd (with bulking) 12](#_Toc204425802)

[.z.ts (without bulking) 13](#_Toc204425803)

[.u.upd (without bulking) 13](#_Toc204425804)

[tick/r.k 14](#_Toc204425805)

[Explanation 14](#_Toc204425806)

[The functions 14](#_Toc204425807)

[.u.end (end of day) 14](#_Toc204425808)

[.u.rep (replay log) 14](#_Toc204425809)

# What is a Tick?

KDB+ Tick is an application built using KDB+ to provide a market data capture system. The architecture of the different components of a Tick system demonstrate a number of different aspects of building KDB+ applications and so severs as a good template for understanding KDB+ architecture. Tick is implemented as a collection of interconnected KDB+ applications which communicate through sockets. The basic architecture consists of 4 KDB+ instances:

* A Feedhandler
* A Tickerplant
* A Real-Time Database
* An Historical Database

Data Feed

Feedhandler

Tickerplant

Log File

Real-Time Database

Historical Database

Chained Tickerplant

Java/C/C#/q Client

Real-Time Subscriber

Real-Time Subscriber

Real-Time Subscriber

## Components of Tick

* Feedhandler
  + subscribes to the feed
  + parses the records as they arrive
  + pushes updates to the tickerplant
* Tickerplant
  + writes updates to log file
  + pushes updates to all subscribing clients
  + discards in-memory data as soon as all clients have received the update
  + sends a notification to all clients that the end-of-day has arrived
* Real-time Database
  + stores intra-day data received from tickerplant
  + writes the day’s data to disk when it receives the end-of-day message
  + in the event of failure, automatically re-syncs itself with the contents of the tickerplant’s log file before receiving further updates
* Historical Database
  + complete on-disk collection of all data stored so far
  + saved by the real-time database on receiving end-of-day message
  + partitioned by date
* Chained Tickerplant
  + receives updates from another tickerplant(or chained tickerplant)
  + can receive only a subset of updates if desired
  + publishes updates to its own list of subscribed clients
* Real-time Subscriber
  + subscribes to updates from tickerplant/chained tickerplant
  + can subscribe to a limited set of tables/symbols
  + any kdb+ process can be a subscriber, simply implement the callback method and call the subscribe function of the tickerplant

# Starting a Tickerplant

Starting a tickerplant is very simple. An example command line to start a tickerplant would be:

q tick.k trade /d2/tp\_logs/ 17:00:00.000 -p 1234 –t 2000

Explanation:

|  |  |  |  |
| --- | --- | --- | --- |
| Command Line | Description | Required | Required Location |
| q | Standard executable to start q | Yes | 1st |
| tick.k | Location tickerplant load script | Yes | 2nd |
| trade | Name of schema file (with .q extension and must be in tick folder in current or QHOME directory) | Yes | 3rd |
| /d2/tp\_logs/ | Location of tickerplant logs (must have write access) | Yes | 4th |
| 17:00:00.000 | Time to run end of day functionality | Yes | 5th |
| -p 1234 | Port that the process will listen to and be can accessed on | No – defaults to 5010 |  |
| -t 2000 | How long to bulk messages for, before sending (if unsupplied the tickerplant will not bulk) | No |  |

# Creating a Real Time Database Subscriber

To start a realtime database subscriber you need to load the r.k script supplied with KDB+ Tick. An example command line would be:

q tick/r.k :1234 sup-chitkdb01:3456 –p 5678

Explanation:

|  |  |  |  |
| --- | --- | --- | --- |
| Command Line | Description | Required | Required Location |
| q | Standard executable to start q | Yes | 1st |
| tick/r.k | Location of real time database load script | Yes | 2nd |
| :1234 | [host]:port location of the tickerplant | No – defaults to 5010 |  |
| sup-chitkdb01:3456 | [host]:port location of the historical database | No |  |
| -p5678 | Port that the process will listen to and be can accessed on | No – defaults to 5011 |  |

# Creating a Chained Tickerplant

To turn any q process into a chained tickerplant, all you need to do run the following two lines

\l tick/u.k  
.u.init[]

This means that all tables in the home directory cannot be subscribed to, and you can publish new data by explicitly calling the .u.pub function (explained later in this document).

# The Scripts

## tick/u.k

u.k contains all the necessary functionality for a tickerplant or chained tickplant. All functions here are stored in the .u sub-directory in the q instance.

\d .u

init:{w::t!(#t::.q.tables`.)#()}

del:{w[x]\_:w[x;;0]?y};

.z.pc:{del[;x]'t}; /remove all subscriptions on port close

sel:{$[`~y;x;select from x where sym in y]}

pub:{[t;x]{[t;x;w]if[#x:sel[x]w 1;(-\*w)(`upd;t;x)]}[t;x]'w t}

add:{$[(#w x)>i:w[x;;0]?.z.w;.[`.u.w;(x;i;1);?,;y];w[x],:,(.z.w;y)];(x;$[99=@v:. x;sel[v]y;0#v])}

sub:{if[x~`;:sub[;y]'t];if[~x in t;'x];del[x].z.w;add[x;y]}

end:{(-?,/w[;;0])@\:(`.u.end;x)}  
\d .

### The functions:

#### .u.init (initialize)

Input parameters: none

Description: This function must be called to turn a q process into a chained tickerplant. It creates:

* .u.t – a list of all tables that a process can subscibe to, to receive future updates
* .u.w – a dictionary that maintains, per table, the handle of a subscriber to send updates to, and which constaints the subscriber has imposed on the the update (e.g. only symbols `MSFT and `APPL)

Example: .u.init[]

#### .u.del (delete)

Input parameters: table name and local connection handle (if unknown use .z.w through TCP message)

Description: removes a clients subscription from the .u.w dictionary. This function is mainly used in .z.pc - .z.pc is called when a handle is closed to the process. When a handle is closed, we must remove the subsciption from .u.w since handles are recycled and could be assigned to a new process. Also, if the process tries to senda message to this handle, it will cause an error to occur.

Example: .u.del[`trade;5]

#### .u.sel (select)

Input Parameters: actual table (not a symbol) and subscription constraints

Description: This runs a query on the tick received to determine which rows each client wishes to receive based on the user’s subscription constraints. If the constraint=`, then no query is ran and the function just returns the full tick.

Example: .u.sel[trade;`MSFT`APPL] will return all MSFT and APPL rows in the trade table

#### .u.pub (publish)

Input Parameters: The table name and the new table data

Description: .u.pub will:

* Access the .u.w dictionary to determine which clients wish to receive this newest tick;
* Calls the .u.sel function (above) to determine which rows in the current tick each client wishes to receive.
* Sends the tick to call subscribers using TCP, with a message of format (`upd;`tbl\_name;DATA)
* Will NOT send a message with no data in the table (e.g. tick doesn’t contain any of the symbols a particular user has subscribed for)

Example: .u.pub[`trade;select from trade where time>09:00.03.000]

#### .u.add (addition)

Input Parameters: The table name and the list of constraints

Desciption: Called Will add a subscription request to the .u.w dictionary. The output of this function is the a list with the table name and an empty table, which can be used client side to predefine or check the table structure before a tick is received. NOTE: This function uses .z.w to determine the handle to which the subscription messages should be sent to.

Example: .u.add[`trade;`MSFT`APPL]

#### .u.sub (subscription)

Input Parameters: The table name and the list of constraints

Description: If the table name is `, then it recalls itself on every table in the list .u.t, i.e. ` means subscribe to all tables. .u.sub, deletes the client’s previous subscription (calling .u.del) and then adds a new subscription (calling .u.add). This function is used by all clients to create a subsciption.

Example: .u.sub[`trade;`]

#### .u.end (end of day)

Input Parameters: Current date

Description: At the end of the day, this function gets called where it sends to every subscriber, run .u.end now for this date. Typical behaviour would be the RDB would save all tables to disk and clear all in-memory tables. To find all subscribers, .u.end finds all the distinct handles in the .u.w dictionary.

Example: .u.end[2008.07.17]

## tick/u.q

This script is intending to overwrite the .u.end function defined above.

// Updated .u.end function

.u.end:{[dt]

tempTab:() xkey `h xgroup ungroup flip `tab`h!((key .u.w);(value .u.w)[;;0]);

if[(count tempTab)>0;

{(neg x)(`.rdb.endOfDay;y;z)}[;dt;]'[tempTab[`h];tempTab[`tab]];

];

};

#### .u.end (end of day – version 2)

Input Parameters: Current date

Description: Creates a table (called tempTab) which contains a list of all handle and table combinations from the .u.w dictionary. It then sends a message to subscriber of the format  
 (` .rdb.endOfDay;2008.07.17;`tbl\_name)   
with the intent that the function .rdb.endOfDay can perform table specific end of Day functionality. This would be useful if the process is subscribing to more than one tickerplant, where the tickerplants may run end of functionality at different times.

Example: .u.end[2008.07.17]

## tick.k

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

if[~"1"~\*.z.l 4;0N!`license];"kdb+tick 2.3 2006.10.18"

/q tick.k SRC [DST] [-p 5010] [-o h]

."\\l tick/",(src:\*.z.x,,"sym"),".q"

if[~."\\p";."\\p 5010"]

\l tick/u.k

\l tick/u.q // New u.q file to define new end of day function

.u.eodSave:1b;

.u.midNight:\_.z.Z;

\d .u

ld:{if[~@!L::`$(-10\_$L),$x;.[L;();:;()]];i::-11!(-2;L);<L}

tick:{init[];if[~&/(`time`sym~2#!+.:)'t;'`timesym];@[;`sym;`g#]'t;d::\_.z.Z;if[l::#y;L::`$":",y,"/",x,10#".";l::ld d]}

endofday:{end d;d+:1;if[l;>l;l::ld d]}

ts:{

if[(.u.saveTime<"t"$x)&.u.eodSave;

-1 "starting EOD";

.u.eodSave:0b;

endofday[]];

// If next day then update .u.nextDay to be 0b

if[(\_x)>.u.midNight;

-1 "Reseting next day flag";

.u.midNight:\_x;

.u.eodSave:1b;

];

};

if[."\\t"

.z.ts:{t pub'.:'t;@[`.;t;@[;`sym;`g#]0#];ts .z.Z}

upd:{[t;x]

if[#x;

if[~-19=@\*\*x;if[d<\_a:.z.Z;.z.ts[]];a:"t"$a;x:$[0>@\*x;a,x;(,(#\*x)#a),x]]

t insert x;if[l;l@,(`upd;t;x);i+:1];]}]

if[~."\\t";."\\t 1000"

.z.ts:{ts .z.Z}

upd:{[t;x]ts a:.z.Z;

if[#x;

if[~-19=@\*\*x;a:"t"$a;x:$[0>@\*x;a,x;(,(#\*x)#a),x]];

f:!+. t;pub[t;$[0>@\*x;,f!x;+f!x]];if[l;l@,(`upd;t;x);i+:1];]}]

\d .

.u.tick[src;.z.x 1];

.u.saveTime:"T"$.z.x 2;

/ WOMBAT calls upd rather than .u.upd

upd:.u.upd

### Explanation

|  |  |
| --- | --- |
| Line 1: | A licence check to prevent tick.k being used on a non-commercial version of q |
| Line 4: | Uses the first input parameter – the schema file name. This line loads the schema from within a tick folder in the current directory or QHOME. |
| Line 6: | If port is defined for this process to listen to, it defaults to port 5010 |
| Line 8: | Load tick/u.k as discussed previously. |
| Line 9: | DRW specific – also load tick/u.q as discussed previously |
| Line 10: | .u.eodSave is used to determine if end of day function has ran yet today. Running .u.end is determined by a regular recurring function that checks if current time has passed the end of day time (see .z.ts function). This .u.eodSave marker exists so as the end of day function can only happen once as soon as we pass the pre-defined end of day run time. |
| Line 11: | Sets .u.midnight as current date, allowing a midnight function to occur as soon as the date changes. |
| Lines 12-45: | All functions in this section are defined in the .u directory unless explicitly declared. |
| Lines 13-29: | Defines various functions – see below for a description of each. |
| Lines 31-43: | Check to see if –t was supplied in the command line.  If is it, then the .u.upd function does not publish when a tick is received, rather it inserts into a local table. .z.ts will publish at regular intervals, and clear out these local tables.  If t is not defined, then .u.upd will publish the tick the moment it is received, and not insert it into the table. |
| Line 46: | Initializes the tickerplant |
| Line 47: | Defines the varaiable .u.saveTime based on the command line input |
| Line 49: | Sets upd=.u.upd for WOMBAT |

### The functions

#### .u.ld (log directory)

Input Parameters: The Date partition for the log file

Description: Creates the log file, defines .u.L – the location of the log file, and defines .u.i – the number of rows in the log file.

Example: .u.ld[`:./logs]

#### .u.tick (tickerplant)

Input Parameters: The name of the schema script and the log file location

Description: Performs the following steps:

1. Runs .u.init[] to create .u.t and .u.w
2. Checks first two columns of every table for time and then sym
3. Applies the g attribute to the sym column of each table (to speed up the .u.sel query)
4. Sets .u.d as current date
5. Checks for a log file, in which cases creates a new one if it doesn’t exist using .u.ld.

Example: .u.tick[`trade;“./logs”]

#### .u.endofday (local end of day)

Input Parameters: None

Description: This function runs the .u.end function defined previously to alert clients that end of day has been reached, then it increases .u.d to the next date and creates a new log file.

Example: .u.endofday[]

#### .u.ts (timed function)

Input Parameters: Current Time

Description: Checks current time against flags to identify if the end of day function or the Midnight function need to be executed.

Example: .u.ts[17:00:00.001]

#### .z.ts (with bulking)

Input Parameters: None

Description: Gets called every \t milliseconds by q. With bulking, it is defined as:

.z.ts:{t pub'.:'t;@[`.;t;@[;`sym;`g#]0#];ts .z.Z}

Where it publishes out every table, deletes from every table and reapplies the g attribute. Finally it runs the checking function .u.ts

#### .u.upd (with bulking)

Input Parameters: Table Name and list of lists representing the columns in the tick received.

Description: Gets called on every tick received. With bulking, it is defined as:

.u.upd:{[t;x]

if[#x;

if[~-19=@\*\*x;if[d<\_a:.z.Z;.z.ts[]];a:"t"$a;x:$[0>@\*x;a,x;(,(#\*x)#a),x]]

t insert x;if[l;l@,(`upd;t;x);i+:1];]}]

The function checks that x contains data, prepends a column with current time, if time has not been supplied, inserts the new data into the tickerplant tables and writes the message to the log file.

Example: .u.upd[`trade;( (09:00:00.000 09:00:00.001 09:00:00.003 09:00:00.040);  
 (`MSFT `YHOO`IBM`MSFT);  
 (200 300 400 250);  
 (1 3.4 5.5 4f))]

#### .z.ts (without bulking)

Input Parameters: None

Description: Gets called every \t milliseconds by q. Without bulking, it is defined as:

.z.ts:{ts .z.Z}

Now it only calls the .u.ts function at regular intervals.

#### .u.upd (without bulking)

Input Parameters: Table Name and list of lists representing the columns in the tick received.

Description: Gets called on every tick received. Without bulking, it is defined as:

.u.upd:{[t;x]ts a:.z.Z;

if[#x;

if[~-19=@\*\*x;a:"t"$a;x:$[0>@\*x;a,x;(,(#\*x)#a),x]];

f:!+. t;pub[t;$[0>@\*x;,f!x;+f!x]];if[l;l@,(`upd;t;x);i+:1];]}]

The function checks that x contains data, prepends a column with current time, if time has not been supplied, immediately publishes the data to the subscribers and writes the message to the log file.

Example: .u.upd[`trade;( (09:00:00.000 09:00:00.001 09:00:00.003 09:00:00.040);  
 (`MSFT `YHOO`IBM`MSFT);  
 (200 300 400 250);  
 (1 3.4 5.5 4f))]

## tick/r.k

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

/q tick/r.k [host]:port[:usr:pwd] [host]:port[:usr:pwd]

/[…] shows optional variables

if[~"w"=\*$.z.o;."\\sleep 1"]

upd:insert

/ get the ticker plant and history ports, defaults are 5010,5012

.u.x:.z.x,(#.z.x)\_(":5010";":5012")

/ end of day: save, clear, hdb reload

.u.end:{t:.q.tables`.;t@:&`g=-2!'t@\:`sym;.Q.hdpf[`$":",.u.x 1;`:.;x;`sym];@[;`sym;`g#]'t;}

/ init schema and sync up from log file;cd to hdb(so client save can run)

.u.rep:{(.[;();:;].)'x;if[~@\*y;:()];-11!y;."\\cd ",1\_-10\_$\*|y}

/ connect to ticker plant for (schema;(logcount;log))

.u.rep .(<`$":",.u.x 0)"(.u.sub[`;`];`.u `i`L)"

### Explanation

|  |  |
| --- | --- |
| Line 3: | If starting using a Window’s version of q, it must wait a second before connecting to the tickerplant |
| Line 4: | Standard upd function to deal with tick updates for an unkeyed table |
| Line 6: | If locations of the tickerplant and historical database are not supplied, default values are used |
| Line 9: | The standard end of day save down function |
| Line 12: | Log Replay function (see below) |
| Line 15: | Calls the Log Replay function, and also opens a handle to the tickerplant and subscribes to every table. |

### The functions

#### .u.end (end of day)

Input Parameters: The Date partition for the historical table

Description: Creates a splayed database for all the tables in the home directory of the real time database

Example: .u.end[2008.07.17]

#### .u.rep (replay log)

Input Parameters: An empty version of the tables and a 2 item list of number of rows and location of the log file.

Description: .u.rep will:

* Use the empty tables (from the tickerplant) to replicate the same table schemas
* Replay the log file using the -11! Function
* Use the cd command to move the shell to the location of the tickerplant log.

Example: .u.rep[enlist ([]time:`time$();sym:`symbol$();price:`float$());(4;/d2/log\_files/trade2008.05.17)]

# INDEX

.

.u.add 5

.u.d 9

.u.del 4

.u.end 6, 9, 11

.u.eodSave 8

.u.i 8

.u.init 4

.u.k 4

.u.L 8

.u.ld 8

.u.midnight 8

.u.pub 5

.u.rep 11

.u.sel 5

.u.sub 6

.u.t 4

.u.tick 8

.u.ts 9

.u.upd 9, 10

.u.w 4, 5

.z.pc 4

.z.ts 9, 10

.z.w 5

B

bulking 3, 9

C

Chained Tickerplant 3

Command Line 3

E

end of day 11

L

licence 7

log file 8, 11

log replay 11

R

r.k 11

Real Time Database 3

T

tick.k 7

U

u.q 6

upd 5