INTRODUCTION - Tick Tutorial

TASK 1 - Parsing Parameters

OVERVIEW

The aim of this tutorial is to write a basic kdb+ tick setup from scratch, given a code skeleton to work from.

A fairly good understanding of q is required, i.e. you should at least have read "Q for Mortals" and/or have participated in a q training. This tutorial will give you the chance to put your knowledge into practice. You can use any docs you like, but you should try to find solutions to the problems yourself.

For a start, download the tar (Tools->Attachments) into your workspace.

AIM OF TUTORIAL

The aim of this tutorial is to give you exposure to one of the most basic setups of a kdb+ architecture, a tick setup.
You will write a very basic and minimal tick setup from scratch, and you will get to see how much functionality you can build with little code (compared to common OOP languages or procedural languages like C++, Java, C, VBA, etc.).
The aim is NOT to have a production ready tick setup with full resiliency etc., but to get you coding on a "real" world project/problem.

TICK ARCHITECTURE OVERVIEW

In short, a tick setup consists of a tickerplant process (**TP**), which logs incoming data and a real-time database (**RT**), which is a tickerplant subscriber.
The **RT** keeps all data fed into the **TP** in-memory, then saves it down at end-of-day (usually midnight).
A historical database (**HDB**) then makes the database available for historical analytics.
A **TP** can have other subscribers, like calculation engines, java consumers, etc.
Feedhandlers (**FH**) publish into a **TP**; they can be written in Java, C/C++, q, etc. You will write a feedhandler in q off simulated market data.

Architecture setup (data flow): **FH** -> **TP** -> **RT** -> **HDB**

FINAL TECHNICAL SETUP (AIM)

- Start tp on port 5001:

  ```
  q tp.q -p 5000 -tp_path /tmp
  ```

- Start fh on port 4000:

  ```
  q fh.q -p 4000 -tp localhost:5000
  ```

- Start simulator:

  ```
  q simu.q -fh localhost:4000 -data data/msgs
  ```

- Start rt on port 5002:

  ```
  q rt.q -p 5001 -tp localhost:5000 -hdb /tmp/tick
  ```

- Start historical database on port 5003:

  ```
  q hdb.q -p 5002 -db /tmp/tick
  ```

SOME NOTES ON Q PHILOSOPHY

- It's terse and expressive
- It aims for for simplicity (terseness is not seen as complexity - arguably that's right)
- Let the compiler (aka c language) do the tricks; it's a high level language,
  think of IPC

SOME NOTES ON CODING STANDARDS

- Unlike mainstream languages, there are no widely followed coding standards, which can make q a horror to read, especially because of its tersness.
- If you fancy, you can read up on some coding standards here: http://www.nsl.com/papers/style.pdf; however, don't see this as the ultimate reference.
- You will see that at the beginning you prefer verbose code, and the more you get an expert in q, you try to shorten as much as possible.
- My favourite standard is to write readable and concise code, while being sensible at the same time. Someone will have to maintain your code eventually.
- And as with any language, choose a guideline and stick to it.

For this project we will try to stick to a few simple rules that will hopefully make q code readable and maintainable:

- Globals are all CAPITAL_CASE names; underscore can be used to make them more readable
- Function names follow c-style coding, e.g. get_param over getParam (in your own projects you are welcome to use camelCase style of course)
- Local vars should be of short names
- Avoid vars ending with _ (_ is a q function)
- No one-line function; instead a function always start with a signature and end with the closing curly bracket "}" on its own line followed by ";", as in

```
f:{[p]
  p+1
 };
```

is preferred over

```
f:{[p] p+1 };
```

- Define functions to have explicit parameters, don't rely on x,y,z implicit parameters supported by q.
- Don't use tab for identation, instead use 2-3 spaces (or set your vi/editor to small tabs)

USEFUL WIKI PAGES

Below are some wiki pages you may find useful while completing this tutorial.

USEFUL LINKS

CODE TIPS

TASK 1 - Parsing Parameters