AquaQ TorQ

email:
support@aquaq.co.uk

web:
www.aquaq.co.uk

AquaQ Analytics

AquaQ

Revision History

Revision	Date	Author(s)	Description
1.0	February 12, 2014	AquaQ	First version released

AquaQ

Copyright ©2013–2014 AquaQ Analytics Limited
Suite 5, Sturgen Building, 9-15 Queen Street, Belfast, BT1 6EA
May be used free of charge. Selling without prior written consent prohibited. Obtain permission before redistributing. In all cases this notice must remain intact.

Contents

AquaQ

AquaQ

 ©AquaQ Analytics Limited 2014

Chapter 1

Company Overview

AquaQ Analytics Limited is a provider of specialist data management, data analytics and data mining services. We also provide strategic advice, training and consulting services in the area of market-data collection to clients predominantly within the capital markets sector. Our domain knowledge, combined with advanced analytical techniques and expertise in best-of-breed technologies, helps our clients get the most out of their data.

The company is currently focussed on four key areas, all of which are conducted either on client site or near-shore:

- Kdb+ Consulting Services: Development, Training and Support. We are an official implementation and training partner of Kx Systems;
- Real Time GUI Development Services;
- SAS Analytics Services;
- Providing IT consultants to investment banks with Java, .NET and Oracle experience.

The company currently has a headcount of 30 consisting of both full time employees and contractors and is actively hiring additional resources. Some of these resources are based full-time on client site while others are involved in remote/near-shore development and support work from our Belfast headquarters. To date we have MSAs in place with 6 major institutions across the UK and the US.

Please feel free to contact us if you feel we may be able to assist you with your kdb+, data or analytics needs.

support@aquaq.co.uk

AquaQ

Chapter 2

Overview

2.1 What is kdb+?

kdb+ is the market leading timeseries database from Kx Systems[1]. kdb+ is used predominently in the Financial Services sector to capture, process and analyse billions of records on a daily basis, with Kx counting almost all of the top tier investment banks as customers. kdb+ incorporates a programming language, q, which is known for its performance and expressive power. Given the unsurpassed data management and analytical capabilities of kdb+, the applicability of kdb+ technology extends beyond the financial domain into any sector where rapid pre-built or adhoc analysis of large datasets is required. Other sectors which have good use cases for kdb+ include utilities, pharmaceuticals, telecoms, manufacturing, retail and any sector utilising telemetry or sensor data.

2.2 What is AquaQ TorQ?

AquaQ TorQ is a framework which forms the basis of a production kdb+ system by implementing some core functionality and utilities on top of kdb+, allowing developers to concentrate on the application business logic. We have tried to incorporate as many best practices as possible, with particular focus on performance, process management, diagnostic information, maintainability and extensibility. We have kept the code as readable as possible using descriptive comments, error messages, function names and variable names. Wherever possible, we have tried to avoid re-inventing the wheel and instead have used contributed code from code.kx.com (either directly or modified). All code sections taken from code.kx.com are referenced in this document.

AquaQ TorQ can be extended or modified as required. We have chosen some default behaviour, but it can all be overridden. The features of AquaQ TorQ are:

- Process Management: Each process is given a type and name. By default these are used to determine the code base it loads, the configuration loaded, log file

[1] `www.kx.com`

AquaQ

naming and how it reports itself to discovery services. Whenever possible we have tried to ensure that all default behaviour can be overridden at the process type level, and further at the process name level.

- Code Management: Processes can optionally load common or process type/name specific code bases. All code loading is error trapped.
- Configuration Management: Configuration scripts can be loaded as standard and with specific process type/name configuration overriding default values. Configuration scripts are loaded in a specific order; default, then process type specific, then process name specific. Values loaded last will override values loaded previously.
- Usage Logging: All process usage is logged to a single text log file and periodically rolled. Logging includes opening/closing of connections, synchronous and asynchronous queries and functions executed on the timer. Logged values include the request, the details of where it came from, the time it was received, the time it took, memory usage before and after, the size of the result set, the status and any error message.
- Incoming and Outgoing Connection Management: Incoming (client) and outgoing (server) connections are stored and their usage monitored through query counts and total data size counts. Connections are stored and retrieved and can be set to automatically be re-opened as required. The password used for outgoing connections can be overridden at default, process type and process name level.
- Access Controls: Basic access controls are provided, and could be extended. These apply restrictions on the IP addresses of remote connections, the users who can access the process, and the functions that each user can execute. A similar hierarchical approach is used for access control management as for configuration management.
- Timer Extensions: Mechanism to allow multiple functions to be added to the timer either on a repeating or one-off basis. Multiple re-scheduling algorithms supplied for repeating timers.
- Standard Out/Error Logging: Functions to print formatted messages to standard out and error. Hooks are provided to extend these as required, e.g. publication to centralised logging database. Standard out and error are redirected to appropriately named, timestamped and aliased log files, which are periodically rolled.
- Error Handling: Different failure options are supplied in case code fails to load; either exit upon failure, stop at the point of failure or trap and continue.
- Documentation and Development Tools: Functionality to document the system is built into AquaQ TorQ, and can be accessed directly from every q session. Developers can extend the documentation as they add new functions. Functionality

 ©AquaQ Analytics Limited 2014

for searching for functions and variables by name and definition is provided, and for ordering variables by memory usage. The standard help.q from code.kx is also included.

- Utilities: We intend to build out and add utilities as we find them to be suitably useful and generic. So far we have:

  1. Async Messaging: allows easy use of advanced async messaging methods such as deferred synchronous communication and async communication using postback functions;
  2. Caching: allows a result set cache to be declared and result sets to be stored and retrieved from the cache. Suitable for functions which may be run multiple times with the same parameters, with the underlying data not changing in a short time frame;
  3. Timezone handling: derived from code.kx, allows conversion between timestamps in different timezones;
  4. Heartbeating: each process can be set to publish heartbeats, and subscribe to and manage heartbeats from other processes in the environment.

AquaQ TorQ will wrap easily around kdb+tick and therefore around any tickerplant, RDB, HDB or real time processing application. We currently have several customised processes of our own:

- Discovery Service: Every process has a type, name and set of available attributes, which are used by other processes to connect to it. The Discovery Service is a central point that can be used to find other available processes. Client processes can subscribe to updates from the discovery service as new processes become available- the discovery service will notify its subscribers, which can then use the supplied hook to implement required behavior e.g. connect to the newly available process;

- Gateway: A fully synchronous and asynchronous gateway is provided. The gateway will connect to a defined list of process types (can be homogenous or heterogeneous processes) and will route queries across them according to the priority of received requests. The routing algorithms can be easily modified e.g. give priority to user X, or only route queries to processes which exist in the same data centre or geographical region to avoid the WAN (this would entail using the process attributes). The gateway can either return the result to the client back down the same handle, or it can wrap it in a callback function to be invoked on the client;

- Tickerplant Log Replay: A process for replaying tickerplant log files to create on-disk data sets. Extended features are provided for only replaying subsets of log files (by message number and/or table name), replaying in chunks, invoking bespoke final behaviour etc.;

 ©AquaQ Analytics Limited 2014

- Monitor: A basic monitoring process which uses the Discovery Service to locate the other processes within the system, listens for heartbeats, and subscribes for log messages. This should be extended as required but provides a basic central point for system health checks;
- Kill: A process used to kill other processes, optionally using the Discovery Service to locate them.

2.3 Do I Really Have to Read This Whole Document?

Hopefully not. The core of AquaQ TorQ is a script called torq.q and we have tried to make it as descriptive as possible, so perhaps that will suffice. The first place to look will be in the config files, the main one being $KDBCONFIG/settings/default.q. This should contain a lot of information on what can be modified. In addition:

- We have added a load of usage information:

```
aquaq$ q torq.q -usage
KDB+ 3.1 2013.10.08 Copyright (C) 1993-2013 Kx Systems

General:
 This script should form the basis of a production kdb+ environment.
 It can be sourced from other files if required, or used as a launch script
     before loading other files/directories using either -load or -loaddir
     flags
... etc ...
```

  If sourcing from another script there are hooks to modify and extend the usage information as required.

- We have some pretty extensive logging:

```
aquaq$ q torq.q -p 9999 -debug
KDB+ 3.1 2013.10.08 Copyright (C) 1993-2013 Kx Systems

2013.11.05D12:22:42.597500000|aquaq|torq.q_3139_9999|INF|init|trap mode (
    initialisation errors will be caught and thrown, rather than causing an
    exit) is set to 0
2013.11.05D12:22:42.597545000|aquaq|torq.q_3139_9999|INF|init|stop mode (
    initialisation errors cause the process loading to stop) is set to 0
2013.11.05D12:22:42.597810000|aquaq|torq.q_3139_9999|INF|init|attempting to
    read required process parameters proctype,procname from file /torqhome/
    config/process.csv
2013.11.05D12:22:42.598081000|aquaq|torq.q_3139_9999|INF|init|read in process
    parameters of proctype=hdb; procname=hdb1
2013.11.05D12:22:42.598950000|aquaq|hdb1|INF|fileload|config file /torqhome/
    config/default.q found
... etc ...
```

- We have added functionality to find functions or variables defined in the session, and also to search function definitions.

```
q).api.f`max
name                | vartype   namespace public descrip              ..
--------------------| ------------------------------------------------..
```

 ©AquaQ Analytics Limited 2014

```
maxs                | function  .q        1      ""                    ..
mmax                | function  .q        1      ""                    ..
.clients.MAXIDLE    | variable  .clients  0      ""                    ..
.access.MAXSIZE     | variable  .access   0      ""                    ..
.cache.maxsize      | variable  .cache    1      "The maximum size in ..
.cache.maxindividual| variable  .cache    1      "The maximum size in ..
max                 | primitive           1      ""                    ..

q)first 0!.api.p`.api
name     | `.api.f
vartype  | `function
namespace| `.api
public   | 1b
descrip  | "Find a function/variable/table/view in the current process"
params   | "[string:search string]"
return   | "table of matching elements"

q).api.p`.api
name         | vartype  namespace public descrip                       ..
-------------| ------------------------------------------------------..
.api.f       | function .api      1      "Find a function/variable/tabl..
.api.p       | function .api      1      "Find a public function/variab..
.api.u       | function .api      1      "Find a non-standard q public ..
.api.s       | function .api      1      "Search all function definitio..
.api.find    | function .api      1      "Generic method for finding fu..
.api.search  | function .api      1      "Generic method for searching ..
.api.add     | function .api      1      "Add a function to the api des..
.api.fullapi| function .api      1      "Return the full function api ..
```

- We have incorporated help.q.

```
q)help`
adverb    | adverbs/operators
attributes| data attributes
cmdline   | command line parameters
data      | data types
define    | assign, define, control and debug
dotz      | .z locale contents
errors    | error messages
save      | save/load tables
syscmd    | system commands
temporal  | temporal - date & time casts
verbs     | verbs/functions
```

- We have separated and commented all of our config:

```
aquaq$ head config/default.q
/- Default configuration - loaded by all processes

/- Process initialisation
\d .proc
loadcommoncode:1b       /- whether to load the common code defined at
                        /- ${KDBCODE}/common
loadprocesscode:0b      /- whether to load the process specific code defined at
                        /- ${KDBCODE}/{process type}
loadnamecode:0b         /- whether to load the name specific code defined at
                        /- ${KDBCODE}/{name of process}
loadhandlers:1b         /- whether to load the message handler code defined at
                        /- ${KDBCODE}/handlers
logroll:1b              /- whether to roll the std out/err logs daily
... etc ...
```

 ©AquaQ Analytics Limited 2014

2.4 Operating System and kdb+ Version

AquaQ TorQ has been built and tested on the linux and OSX operating systems though as far as we are aware there is nothing that would make this incompatible with Solaris or Windows. It has also been tested with kdb+ 3.1 and 2.8. Please report any incompatibilities with other kdb+ versions or operating systems.

2.5 License

This code is released under the MIT license[2].

AquaQ

[2] http://opensource.org/licenses/MIT

 ©AquaQ Analytics Limited 2014

Chapter 3

Getting Started

kdb+ is very customisable. Customisations are contained in q scripts (.q files), which define functions and variables which modify the behaviour of a process. Every q process can load a single q script, or a directory containing q scripts and/or q data files. Hooks are provided to enable the programmer to apply a custom function to each entry point of the process (.z.p*), to be invoked on the timer (.z.ts) or when a variable in the top level namespace is amended (.z.vs). By default none of these hooks are implemented.

We provide a codebase and a single main script, torq.q. torq.q is essentially a wrapper for bespoke functionality which can load other scripts/directories, or can be sourced from other scripts. Whenever possible, torq.q should be invoked directly and used to load other scripts as required. torq.q will:

- ensure the environment is set up correctly;
- define some common utility functions (such as logging);
- execute process management tasks, such as discovering the name and type of the process, and re-directing output to log files;
- load configuration;
- load the shared code based;
- set up the message handlers;
- load any required bespoke scripts.

The behavior of torq.q is modified by both command line parameters and configuration. We have tried to keep as much as possible in configuration files, but if the parameter either has a global effect on the process or if it is required to be known before the configuration is read, then it is a command line parameter.

AQUAQ

3.1 File Structure

An example file structure with approximate description is displayed below.

```
|-basecode
|---torq.q           <- Main torq script
|---setenv.sh        <- example script to set necessary environment variables
                        (will not work on Windows)
|---code             <- code directory, defined by $KDBCODE
|-----common         <- common code base dir, loaded by all processes
|-------api.q
|-------apidetails.q <- used to augment the api details in the process
|-------async.q
|-------cache.q
|-------heartbeat.q
|-------help.q
|-------order.txt    <- used to define order that scripts are loaded
|-------pubsub.q
|-------timer.q
|-------timezone.q
|-----handlers       <- message handlers, optionally loaded by all processes
|-------apidetails.q
|-------controlaccess.q
|-------dotz.q
|-------logusage.q
|-------order.txt
|-------trackclients.q
|-------trackservers.q
|-------zpsignore.q
|-----hdb            <- example code dir for process with type or name "hdb"
|-----rdb            <- example code dir for process with type or name "rdb"
|-----processes      <- process script directory
|-------discovery.q
|-------gateway.q
|-------kill.q
|-------monitor.q
|-------tickerlogreplay.q
|---config           <- config directory, defined by $KDBCONFIG
|-----process.csv    <- definition of proc type and name for each host:port
|-----tzinfo         <- timezone data file
|-----passwords      <- password files contain passwords for external connections
|-------default.txt  <- default password
|-------gateway.txt  <- password used by proc name/type "gateway"
|-------hdb.txt
|-----permissions    <- permissions files
|-------default_functions.csv        <- default permissions files
|-------default_hosts.csv
|-------default_users.csv
|-------hdb_users.csv                <- users file used by proc name/type "hdb"
|-----settings       <- config scripts
|-------default.q    <- default configuration
|-------discovery.q  <- all other config scripts are loaded by processes
|-------gateway.q       with a matching name or type
|-------hdb.q
|-------hdb1.q
|-------monitor.q
|-------tickerlogreplay.q
|---logs             <- log directory, defined by $KDBLOGS
```

 ©AquaQ Analytics Limited 2014

3.2 Using torq.q

torq.q can be invoked directly from the command line and be set to source a specified file or directory. torq.q requires the 3 environment variables to be set (see section 3.3). If using a unix environment, this can be done with the setenv.sh script. To start a process in the foreground without having to modify any other files (e.g. process.csv) you need to specify the type and name of the process as parameters. An example is below.

```
$ . setenv.sh
$ q torq.q -debug -proctype testproc -procname test1
```

To load a file, do:

```
$ q torq.q -load myfile.q -debug -proctype testproc -procname test1
```

It can also be sourced from another script. If this is the case, some of the variables can be overridden, and the usage information can be modified or extended. Any variable that has a definition like below can be overridden from the loading script.

```
myvar:@[value;`myvar;1 2 3]
```

The available command line parameters are:

Cmd Line Param	Description
-procname x -proctype y	The process name and process type
-procfile x	The name of the file to get the process information from
-load x [y..z]	The files or database directory to load
-loaddir x [y..z]	Load all .q, .k files in specified directories
-trap	Any errors encountered during initialization when loading external files will be caught and logged, processing will continue
-stop	Stop loading the file if an error is encountered but do not exit
-noredirect	Do not redirect std out/std err to a file (useful for debugging)
-noredirectalias	Do not create an alias for the log files (aliases drop any suffix e.g. timestamp suffix)
-noconfig	Do not load configuration
-nopi	Reset the definition of .z.pi to the initial value (useful for debugging)
-debug	Equivalent to [-nopi -noredirect]
-usage	Print usage info and exit

Table 3.1: torq.q Command Line Parameters

In addition any process variable in a namespace (.*.*) can be overridden from the command line. Any value supplied on the command line will take priority over any other predefined value (.e.g. in a configuration or wrapper). Variable names should be supplied with full qualification e.g. -.servers.HOPENTIMEOUT 5000.

 ©AquaQ Analytics Limited 2014

3.3 Environment Variables

Three environment variables are required:

Environment Variable	Description
KDBCONFIG	The base configuration directory
KDBCODE	The base code directory
KDBLOGS	Where standard out/error and usage logs are written

Table 3.2: Required Environment Variables

torq.q will check for these and exit if they are not set. If torq.q is being sourced from another script, the required environment variables can be extended by setting .proc.envvars before loading torq.q.

3.4 Process Identification

At the crux of AquaQ TorQ is how processes identify themselves. This is defined by two variables - .proc.proctype and .proc.procname which are the type and name of the process respectively. These two values determine the code base and configuration loaded, and how they are connected to by other processes.

The most important of these is the proctype. It is up to the user to define at what level to specify a process type. For example, in a production environment it would be valid to specify processes of type "hdb" (historic database) and "rdb" (real time database). It would also be valid to segregate a little more granularly based on approximate functionality, for example "hdbEMEA" and "hdbAmericas". The actual functionality of a process can be defined more specifically, but this will be discussed later. The procname value is used solely for identification purposes. A process can determine its type and name in one of four ways:

1. From the process file in the default location of $KDBCONFIG/process.csv;
2. From the process file defined using the command line parameter -procfile;
3. Using the command line parameters -proctype and -procname;
4. By defining .proc.proctype and .proc.procname in a script which loads torq.q.

For options 3 and 4, both parameters must be defined using that method or neither will be used (the values will be read from the process file). The process file has format as below.

```
aquaq$ cat config/process.csv
host,port,proctype,procname
aquaq,9997,rdb,rdb_europe_1
aquaq,9998,hdb,hdb_europe_1
aquaq,9999,hdb,hdb_europe_2
```

 ©AquaQ Analytics Limited 2014

The process will read the file and try to identify itself based on the host and port it is started on. The host can either be the value returned by .z.h, or the ip address of the server. If the process can not automatically identify itself it will exit.

3.5 Logging

By default, each process will redirect output to a standard out log and a standard error log, and create aliases for them. These will be rolled at midnight on a daily basis. They are all written to the $KDBLOGS directory. The log files created are:

Log File	Description
out_[procname]_[date].log	Timestamped out log
err_[procname]_[date].log	Timestamped error log
out_[procname].log	Alias to current log log
err_[procname].log	Alias to current error log

Table 3.3: Log Files

The date suffix can be overridden by modifying the .proc.logtimestamp function and sourcing torq.q from another script. This could, for example, change the suffixing to a full timestamp.

3.6 Configuration Loading

The process configuration is contained in q scripts, and stored in the $KDBCONFIG directory. Each process tries to load all the configuration it can find. Each process will attempt to load three configuration files in the below order-

- default.q: default configuration loaded by all processes. In a standard installation this should contain the superset of customisable configuration, including comments;
- [proctype].q: configuration for a specific process type;
- [procname].q: configuration for a specific named process.

The only one which should always be present is default.q. Each of the other scripts can contain a subset of the configuration variables, which will override anything loaded previously. Configuration is loaded before code.

3.7 Code Loading

Code is loaded from the $KDBCODE directory. There is also a common codebase, a codebase for each process type, and a code base for each process name, contained in the following directories and loaded in this order:

 ©AquaQ Analytics Limited 2014

- $KDBCODE/common: shared codebase loaded by all processes;
- $KDBCODE/[proctype]: code for a specific process type;
- $KDBCODE/[procname]: code for a specific process name;

For any directory loaded, the load order can be specified by adding order.txt to the directory. order.txt dictates the order that files in the directory are loaded. If a file is not in order.txt, it will still be loaded but after all the files listed in order.txt have been loaded.

Additional directories can be loaded using the -loaddir command line parameter.

3.8 Initialization Errors

Initialization errors can be handled in different ways. The default action is any initialization error causes the process to exit. This is to enable fail-fast type conditions, where it is better for a process to fail entirely and immediately than to start up in an indeterminate state. This can be overridden with the -trap or -stop command line parameters. With -trap, the process will catch the error, log it, and continue. This is useful if, for example, the error is encountered loading a file of stored procedures which may not be invoked and can be reloaded later. With -stop the process will halt at the point of the error but will not exit. Both -stop and -trap are useful for debugging.

AquaQ

 ©AquaQ Analytics Limited 2014

Chapter 4

Message Handlers

There is a separate code directory containing message handler customizations. This is found at $KDBCODE/handlers. Much of the code is derived from Simon Garland's contributions to code.kx[1].

Every external interaction with a process goes through a message handler, and these can be modified to, for example, log or restrict access. Passing through a bespoke function defined in a message handler will add extra processing time and therefore latency to the message. All the customizations we have provided aim to minimise additional latency, but if a bespoke process is latency sensitive then some or all of the customizations could be switched off. We would argue though that generally it is better to switch on all the message handler functions which provide diagnostic information, as for most non-latency sensitive processes (HDBs, Gateways, some RDBs etc.) the extra information upon failure is worth the cost. The message handlers can be globally switched off by setting .proc.loadhandlers to 0b in the configuration file.

[1] http://code.kx.com/wiki/Contrib/UsingDotz

AquaQ

Script	NS	Diag	Function	Modifies
logusage.q	.usage	Y	Log all client interaction to an ascii log file and/or in-memory table. Messages can be logged before and after they are processed. Timer calls are also logged. Exclusion function list can be applied to .z.ps to disable logging of asynchronous real time updates	pw, po, pg, ps, pc, ws, ph, pp, pi, exit, timer
controlaccess.q	.access	N	Restrict access for set of users/user groups to a list of functions, and from a defined set of servers	pw, pg, ps, ws, ph, pp, pi
trackclients.q	.clients	Y	Track client process details including then number of requests and cumulative data size returned	po, pg, ps, ws, pc
trackservers.q	.servers	Y	Discover and track server processes including name, type and attribute information. This also contains the core of the code which can be used in conjunction with the discovery service.	pc, timer
zpsignore.q	.zpsignore	N	Override async message handler based on certain message patterns	ps

Table 4.1: Message Handler Scripts

Each customization can be turned on or off individually from the configuration file(s). Each script can be extensively customised using the configuration file. Example customization for logusage.q, taken from $KDBCONFIG/settings/default.q is below. Please see default.q for the remaining configuration of the other message handler files.

```
/- Configuration used by the usage functions - logging of client interaction
\d .usage
enabled:1b              /- whether the usage logging is enabled
logtodisk:1b            /- whether to log to disk or not
logtomemory:1b          /- write query logs to memory
ignore:1b               /- check the ignore list for functions to ignore
ignorelist:(`upd;"upd") /- the list of functions to ignore in async calls
flushtime:1D00          /- default value for how long to persist the
                        /- in-memory logs. Set to 0D for no flushing
suppressalias:0b        /- whether to suppress the log file alias creation
logtimestamp:{[].z.d}   /- function to generate the log file timestamp suffix
LEVEL:3                 /- log level. 0=none;1=errors;2=errors+complete
                        /- queries;3=errors+before a query+after
logroll:1b              /- Whether or not to roll the log file
                        /- automatically (on a daily schedule)
```

 ©AquaQ Analytics Limited 2014

4.1 logusage.q

logusage.q is probably the most important of the scripts from a diagnostic perspective. It is a modified version of the logusage.q script on code.kx.

In its most verbose mode it will log information to an in-memory table (.usage.usage) and an on-disk ASCII file, both before and after every client interaction and function executed on the timer. These choices were made because:

- logging to memory enables easy interrogation of client interaction;
- logging to disk allows persistence if the process fails or locks up. ASCII text files allow interrogation using OS tools such as vi, grep or tail;
- logging before a query ensures any query that adversely effects the process is definitely captured, as well as capturing some state information before the query execution;
- logging after a query captures the time taken, result set size and resulting state;
- logging timer calls ensures a full history of what the process is actually doing. Also, timer call performance degradation over time is a common source of problems in kdb+ systems.

The following fields are logged in .usage.usage:

Field	Description
time	Time the row was added to the table
id	ID of the query. Normally before and complete rows will be consecutive but it might not be the case if the incoming call invokes further external communication
timer	Execution time. Null for rows with status=b (before)
zcmd	.z handler the query arrived through
status	Query status. One of b, c or e (before, complete, error)
a	Address of sender. .dotz.ipa can be used to convert from the integer format to a hostname
u	Username of sender
w	Handle of sender
cmd	Command sent
mem	Memory statistics
sz	Size of result. Null for rows with status of b or e
error	Error message

Table 4.2: Usage Logging Fields

 ©AquaQ Analytics Limited 2014

4.2 controlaccess.q

controlaccess.q is used to restrict client access to the process. It is modified version of controlaccess.q from code.kx. The script allows control of several aspects:

- the host/ip address of the servers which are allowed to access the process;
- definition of three user groups (default, poweruser and superuser) and the actions each group is allowed to do;
- the group(s) each user is a member of, and any additional actions an individual user is allowed/disallowed outside of the group permissions;
- the maximum size of the result set returned to a client.

The access restrictions are loaded from csv files. The permissions files are stored in $KDBCONFIG/permissions.

File	Description
*_hosts.csv	Contains hostname and ip address (patterns) for servers which are allowed or disallowed access. If a server is not found in the list, it is disallowed
*_users.csv	Contains individual users and the user groups they are are a member of
*_functions.csv	Contains individual functions and whether each user group is allowed to execute them. ; separated user list enables functions to be allowed by individual users

Table 4.3: Permissions Files

The permissions files are loaded using a similar hierarchical approach as for the configuration and code loading. Three files can be provided- default_*.csv, [proctype]_*.csv, and [procname]_*.csv. All of the files will be loaded, but permissions for the same entity (hostpattern, user, or function) defined in [procname]_*.csv will override those in [proctype]_*.csv which will in turn override [procname]_*.csv.

When a client makes a query which is refused by the permissioning layer, an error will be raised and logged in .usuage.usage if it is enabled.

4.3 trackclients.q

trackclients.q is used to track client interaction. It is a slightly modified version of trackclients.q from code.kx, and extends the functionality to handle interaction with the discovery service.

Whenever a client opens a connection to the q process, it will be registered in the .clients.clients table. Various details are logged, but from a diagnostic perspective the

 ©AquaQ Analytics Limited 2014

most important information are the client details, the number of queries it has run, the last time it ran a query, the number of failed queries and the cumulative size of results returned to it.

4.4 trackservers.q

trackservers.q is used to register and maintain handles to external servers. It is a heavily modified version of trackservers.q from code.kx. It is explained more in section 5.

4.5 zpsignore.q

zpsignore.q is used to check incoming async calls for certain patterns and to bypass all further message handler checks for messages matching the pattern. This is useful for handling update messages published to a process from a data source.

4.6 Diagnostic Reporting

The message handler modifications provide a wealth of diagnostic information including:

- the timings and memory usage for every query run on a process;
- failed queries;
- clients trying to do things they are not permissioned for;
- the clients which are querying often and/or regularly extracting large datasets;
- the number of clients currently connected;
- timer calls and how long they take.

Although not currently implemented, it would be straightforward to use this information to implement reports on the behaviour of each process and the overall health of the system. Similarly it would be straightforward to set up periodic publication to a central repository to have a single point for system diagnostic statistics.

AquaQ

 ©AquaQ Analytics Limited 2014

Chapter 5

Connection Management

trackservers.q is used to register and maintain handles to external servers. It is a heavily modified version of trackservers.q from code.kx. All the options are described in the default config file. All connections are tracked in the .servers.SERVERS table. When the handle is used the count and last query time are updated.

```
q).servers.SERVERS
procname     proctype  hpup                              w  hits startp
                           lastp                           endp
                             attributes
-------------------------------------------------------------------------------
discovery1   discovery :aquaq:9996    0                                    2014.01.08
    D11:13:10.583056000                              ()!()
discovery2   discovery :aquaq:9995 6  0    2014.01.07D16:44:47.175757000 2014.01.07
    D16:44:47.174408000                              ()!()
rdb_europe_1 rdb       :aquaq:9998 12 0    2014.01.07D16:46:47.897910000 2014.01.07
    D16:46:47.892901000 2014.01.07D16:46:44.626293000 `datacentre`country!`essex`uk
rdb1         rdb       :aquaq:5011 7  0    2014.01.07D16:44:47.180684000 2014.01.07
    D16:44:47.176994000                              `datacentre`country!`essex`uk
rdb_europe_1 hdb       :aquaq:9997    0                                    2014.01.08
    D11:13:10.757801000                              ()!()
hdb1         hdb       :aquaq:9999    0                                    2014.01.08
    D11:13:10.757801000                              ()!()
hdb2         hdb       :aquaq:5013 8  0    2014.01.07D16:44:47.180684000 2014.01.07
    D16:44:47.176994000                              `datacentre`country!`essex`uk
hdb1         hdb       :aquaq:5012 9  0    2014.01.07D16:44:47.180684000 2014.01.07
    D16:44:47.176994000                              `datacentre`country!`essex`uk

q)last .servers.SERVERS
procname  | `hdb2
proctype  | `hdb
hpup      | `:aquaq:5013
w         | 8i
hits      | 0i
startp    | 2014.01.08D11:51:01.928045000
lastp     | 2014.01.08D11:51:01.925078000
endp      | 0Np
attributes| `datacentre`country!`essex`uk
```

5.1 Connections

Processes locate other processes based on their process type. The location is done either statically using the process.csv file or dynamically using a discovery service. It is recommended to use the discovery service as it allows the process to be notified as new processes become available.

The main configuration variable is .servers.CONNECTIONS, which dictates which process type(s) to create connections to. .servers.startup[] must be called to initialise the connections. When connections are closed, the connection table is automatically updated. The process can be set to periodically retry connections.

5.2 Process Attributes

Each process can report a set of attributes. When process A connects to process B, process A will try to retrieve the attributes of process B. The attributes are defined by the result of the .proc.getattributes function, which is by default an empty dictionary. Attributes are used to retrieve more detail about the capabilities of each process, rather than relying on the broad brush process type and process name categorization. Attributes can be used for intelligent query routing. Potential fields for attributes include:

- range of data contained in the process;
- available tables;
- instrument universe;
- physical location;
- any other fields of relevance.

5.3 Connection Passwords

The password used by a process to connect to external processes is retrieved using the .servers.loadpassword function call. By default, this will read the password from a txt file contained in $KDBCONFIG/passwords. A default password can be used, which is overridden by one for the process type, which is itself overridden by one for the process name. For greater security, the .servers.loadpassword function should be modified.

5.4 Retrieving and Using Handles

A function .servers.getservers is supplied to return a table of handle information. .servers.getservers takes five parameters:

- type-or-name: whether the lookup is to be done by type or name (can be either proctype or procname);

 ©AquaQ Analytics Limited 2014

- types-or-names: the types or names to retrieve e.g. hdb;
- required-attributes: the dictionary of attributes to match on;
- open-dead-connections: whether to re-open dead connections;
- only-one: whether we only require one handle. So for example if 3 services of the supplied type are registered, and we have an open handle to 1 of them, the open handle will be returned and the others left closed irrespective of the open-dead-connections parameter.

.servers.getservers will compare the required parameters with the available parameters for each handle. The resulting table will have an extra column called attribmatch which can be used to determine how good a match the service is with the required attributes. attribmatch is a dictionary of (required attribute key) ! (Boolean full match; intersection of attributes).

```
q).servers.SERVERS
procname     proctype  hpup                            w hits startp
                           lastp                           endp attributes
-------------------------------------------------------------------------------
discovery1   discovery :aquaq:9996   0                                    2014.01.08D11
    :51:01.922390000      ()!()
discovery2   discovery :aquaq:9995 6 0    2014.01.08D11:51:01.923812000 2014.01.08D11
    :51:01.922390000      ()!()
rdb_europe_1 rdb       :aquaq:9998   0                                    2014.01.08D11
    :51:38.347598000      ()!()
rdb_europe_2 rdb       :aquaq:9997   0                                    2014.01.08D11
    :51:38.347598000      ()!()
rdb1         rdb       :aquaq:5011 7 0    2014.01.08D11:51:01.928045000 2014.01.08D11
    :51:01.925078000      `datacentre`country!`essex`uk
hdb3         hdb       :aquaq:5012 9 0    2014.01.08D11:51:38.349472000 2014.01.08D11
    :51:38.347598000      `datacentre`country!`essex`uk
hdb2         hdb       :aquaq:5013 8 0    2014.01.08D11:51:01.928045000 2014.01.08D11
    :51:01.925078000      `datacentre`country!`essex`uk

/- pull back hdbs.  Leave the attributes empty
q).servers.getservers[`proctype;`hdb;()!();1b;f0b]
procname proctype lastp                          w hpup        attributes
                      attribmatch
-------------------------------------------------------------------------------
hdb3     hdb      2014.01.08D11:51:38.347598000 9 :aquaq:5012 `datacentre`country!`
    essex`uk ()!()
hdb2     hdb      2014.01.08D11:51:01.925078000 8 :aquaq:5013 `datacentre`country!`
    essex`uk ()!()

/- supply some attributes
q).servers.getservers[`proctype;`hdb;(enlist`country)!enlist`uk;1b;0b]
procname proctype lastp                          w hpup        attributes
                      attribmatch
-------------------------------------------------------------------------------
hdb3     hdb      2014.01.08D11:51:38.347598000 9 :aquaq:5012 `datacentre`country!`
    essex`uk (,`country)!,(1b;,`uk)
hdb2     hdb      2014.01.08D11:51:01.925078000 8 :aquaq:5013 `datacentre`country!`
    essex`uk (,`country)!,(1b;,`uk)
q).servers.getservers[`proctype;`hdb;`country`datacentre!`uk`slough;1b;0b]
procname proctype lastp                          w hpup        attributes
                      attribmatch
-------------------------------------------------------------------------------
```

 ©AquaQ Analytics Limited 2014

```
hdb3     hdb      2014.01.08D11:51:38.347598000 9 :aquaq:5012 `datacentre`country!`
    essex`uk `country`datacentre!((1b;,`uk);(0b;`symbol$()))
hdb2     hdb      2014.01.08D11:51:01.925078000 8 :aquaq:5013 `datacentre`country!`
    essex`uk `country`datacentre!((1b;,`uk);(0b;`symbol$()))
```

.servers.getservers will try to automatically re-open connections if required.

```
q).servers.getservers[`proctype;`rdb;()!();1b;0b]
2014.01.08D12:01:06.023146000|aquaq|gateway1|INF|conn|attempting to open handle to :
    aquaq:9998
2014.01.08D12:01:06.023581000|aquaq|gateway1|INF|conn|connection to :aquaq:9998
    failed: hop: Connection refused
2014.01.08D12:01:06.023597000|aquaq|gateway1|INF|conn|attempting to open handle to :
    aquaq:9997
2014.01.08D12:01:06.023872000|aquaq|gateway1|INF|conn|connection to :aquaq:9997
    failed: hop: Connection refused
procname proctype lastp                          w hpup         attributes
                      attribmatch
--------------------------------------------------------------------------------
rdb1     rdb      2014.01.08D11:51:01.925078000 7 :aquaq:5011 `datacentre`country!`
    essex`uk ()!()

/- If we only require one connection, and we have one open,then it doesn't retry
    connections
q).servers.getservers[`proctype;`rdb;()!();1b;1b]
procname proctype lastp                          w hpup        attributes
                      attribmatch
--------------------------------------------------------------------------------
rdb1     rdb      2014.01.08D11:51:01.925078000 7 :aquaq:5011 `datacentre`country!`
    essex`uk ()!()
```

There are two other functions supplied for retrieving server details, both of which are based on .servers.getservers. .servers.gethandlebytype returns a single handle value, .servers.gethpupbytype returns a single host:port value. Both will re-open connections if there are not any valid connections. Both take two parameters:

- types: the type to retrieve e.g. hdb;
- selection-algorithm: can be one of any, last or roundrobin.

5.5 Manually Adding And Using Connections

Connections can also be manually added and used. See .api.p".servers.*" for details.

 ©AquaQ Analytics Limited 2014

Chapter 6

Utilities

We have provided several utility scripts, which either implement developer aids or standard operations which are useful across processes.

6.1 api.q

This provides a mechanism for documenting and publishing function/variable/table or view definitions within the kdb+ process. It provides a search facility both by name and definition (in the case of functions). There is also a function for returning the approximate memory usage of each variable in the process in descending order.

Definitions are added using the .api.add function. A variable can be marked as public or private, and given a description, parameter list and return type. The search functions will return all the values found which match the pattern irrespective of them having a pre-defined definition.

Whether a value is public or private is defined in the definitions table. If not found then by default all values are private, except those which live in the .q or top level namespace.

.api.f is used to find a function, variable, table or view based on a case-insensitive pattern search. If a symbol parameter is supplied, a wildcard search of *[suppliedvalue]* is done. If a string is supplied, the value is used as is, meaning other non-wildcard regex pattern matching can be done.

```
q).api.f`max
name                | vartype   namespace public descrip             ..
--------------------| ----------------------------------------------..
maxs                | function  .q        1      ""                  ..
mmax                | function  .q        1      ""                  ..
.clients.MAXIDLE    | variable  .clients  0      ""                  ..
.access.MAXSIZE     | variable  .access   0      ""                  ..
.cache.maxsize      | variable  .cache    1      "The maximum size in..
.cache.maxindividual| variable  .cache    1      "The maximum size in..
max                 | primitive           1      ""                  ..
q).api.f"max*"
name| vartype   namespace public descrip params return
----| ---------------------------------------------
maxs| function  .q        1      ""      ""     ""
max | primitive           1      ""      ""     ""
```

AquaQ

.api.p is the same as .api.f, but only returns public functions. .api.u is as .api.p, but only includes user defined values i.e. it excludes q primitives and values found in the .q, .Q, .h and .o namespaces. .api.find is a more general version of .api.f which can be used to do case sensitive searches.

.api.s is used to search function definitions for specific values.

```
q).api.s"*max*"
function            definition                                          ..
--------------------------------------------------------------------------..
.Q.w                "k){`used`heap`peak`wmax`mmap`mphy`syms`symw!(.\"..
.clients.cleanup    "{if[count w0:exec w from`.clients.clients where ..
.access.validsize   "{[x;y;z] $[superuser .z.u;x;MAXSIZE>s:-22!x;x;'\..
.servers.getservers "{[nameortype;lookups;req;autoopen;onlyone]\n r:$..
.cache.add          "{[function;id;status]\n \n res:value function;\n..
```

.api.m is used to return the approximate memory usage of variables and views in the process, retrieved using -22!. Views will be re-evaluated if required. Use .api.mem[0b] if you do not want to evaluate and return views.

```
q).api.m[]
variable          size    sizeMB
--------------------------------
.tz.t             1587359 2
.help.TXT         15409   0
.api.detail       10678   0
.proc.usage       3610    0
.proc.configusage 1029    0
..
```

.api.whereami[lambda] can be used to retrieve the name of a function given its definition. This can be useful in debugging.

```
q)g:{x+y}
q)f:{20 + g[x;10]}
q)f[10]
40
q)f[`a]
{x+y}
'type
+
`a
10
q)).api.whereami[.z.s]
`..g
```

6.2 async.q

kdb+ processes can communicate with each using either synchronous or asynchronous calls. Synchronous calls expect a response and so the server must process the request when it is received to generate the result and return it to the waiting client. Asynchronous calls do not expect a response so allow for greater flexibility. The effect of synchronous calls can be replicated with asynchronous calls in one of two ways (further details in section 7.2):

 ©AquaQ Analytics Limited 2014

- deferred synchronous: the client sends an async request, then blocks on the handle waiting for the result. This allows the server more flexibility as to how and when the query is processed;
- asynchronous postback: the client sends an async request which is wrapped in a function to be posted back to the client when the result is ready. This allows the server flexibility as to how and when the query is processed, and allows the client to continue processing while the server is generating the result.

The code for both of these can get a little tricky, largely due to the amount of error trapping required. We have provided two functions to allow these methods to be used more easily. .async.deferred takes a list of handles and a query, and will return a two item list of (success;results).

```
q).async.deferred[3 5;({system"sleep 1";system"p"};())]
1    1
9995 9996
q).async.deferred[3 5;({x+y};1;2)]
1 1
3 3
q).async.deferred[3 5;({x+y};1;`a)]
0                          0
"error: server fail:type" "error: server fail:type"
q).async.deferred[3 5 87;({system"sleep 1";system"p"};())]
1     1     0
9995i 9996i "error: comm fail: failed to send query"
```

.async.postback takes a list of handles, a query, and the name or lambda of the postback function to return the result to. It will immediately return a success vector, and the results will be posted back to the client when ready.

```
q).async.postback[3 5;({system"sleep 1";system"p"};());`showresult]
11b
q)
q)9995i
9996i

q).async.postback[3 5;({x+y};1;2);`showresult]
11b
q)3
3

q).async.postback[3 5;({x+y};1;`a);`showresult]
11b
q)"error: server fail:type"
"error: server fail:type"

q).async.postback[3 5;({x+y};1;`a);showresult]
11b
q)"error: server fail:type"
"error: server fail:type"

q).async.postback[3 5 87;({x+y};1;2);showresult]
110b
q)3
3
```

For more details, see .api.p".async.*".

 ©AquaQ Analytics Limited 2014

6.3 timer.q

kdb+ provides a single timer function, .z.ts which is triggered with the frequency specified by -t. We have provided an extension to allow multiple functions to be added to the timer and fired when required. The basic concept is that timer functions are registered in a table, with .z.ts periodically checking the table and running whichever functions are required. This is not a suitable mechanism where very high frequency timers are required (e.g. sub 500ms).

There are two ways a function can be added to a timer- either as a repeating timer, or to fire at a specific time. When a repeating timer is specified, there are three options as to how the timer can be rescheduled. Assuming that a timer function with period P is scheduled to fire at time T0, actually fires at time T1 and finishes at time T2, then

- mode 0 will reschedule for T0+P;
- mode 1 will reschedule for T1+P;
- mode 2 will reschedule for T2+P.

Both mode 0 and mode 1 have the potential for causing the timer to back up if the finish time T2 is after the next schedule time. See .api.p".timer.*"for more details.

6.4 pubsub.q

pubsub.q is essentially a placeholder script to allow publish and subscribe functionality to be implemented. Licenced kdb+tick users can use the publish and subscribe functionality implemented in u.[k|q]. If u.[k|q] is placed in the common code directory and loaded before pubsub.q (make sure u.[k|q] is listed before pubsub.q in order.txt) then publish and subscribe will be implemented. You can also build out this file to add your own publish and subscribe routines as required.

6.5 heartbeat.q

heartbeat.q implements heartbeating, and relies on both timer.q and pubsub.q. A table called heartbeat will be published periodically, allowing downstream processes to detect the availability of upstream components. The heartbeat table contains a heartbeat time and counter. The heartbeat script contains functions to handle and process heartbeats and manage upstream process failures. See .api.p“.hb.*”for details.

6.6 cache.q

cache.q provides a mechanism for storing function results in a cache and returning them from the cache if they are available and non stale. This can greatly boost performance for frequently run queries.

 ©AquaQ Analytics Limited 2014

The result set cache resides in memory and as such takes up space. It is up to the programmer to determine which functions are suitable for caching. Likely candidates are those where some or all of the following conditions hold:

- the function is run multiple times with the same parameters (perhaps different clients all want the same result set);
- the result set changes infrequently or the clients can accept slightly out-of-date values;
- the result set is not too large and/or is relatively expensive to produce. For example, it does not make sense to cache raw data extracts.

The cache has a maximum size and a maximum size for any individual result set, both of which are defined in the configuration file. Size checks are done with -22! which will give an approximation (but underestimate) of the result set size. In the worst case the estimate could be half the size of the actual size.

If a new result set is to be cached, the size is checked. Assuming it does not exceed the maximum individual size then it is placed in the cache. If the new cache size would exceed the maximum allowed space, other result sets are evicted from the cache. The current eviction policy is to remove the least recently accessed result sets until the required space is freed. The cache performance is tracked in a table. Cache adds, hits, fails, reruns and evictions are monitored.

The main function to use the cache is .cache.execute[function; staletime]. If the function has been executed within the last staletime, then the result is returned from the cache. Otherwise the function is executed and placed in the cache.

The function is run and the result placed in the cache:

```
q)\t r:.cache.execute[({system"sleep 2"; x+y};1;2);0D00:01]
2023
q)r
3
```

The second time round, the result set is returned immediately from the cache as we are within the staletime value:

```
q)\t r1:.cache.execute[({system"sleep 2"; x+y};1;2);0D00:01]
0
q)r1
3
```

If the time since the last execution is greater than the required stale time, the function is re-run, the cached result is updated, and the result returned:

```
q)\t r2:.cache.execute[({system"sleep 2"; x+y};1;2);0D00:00]
2008
q)r2
3
```

The cache performance is tracked:

```
q).cache.getperf[]
time                          id status function
-------------------------------------------------------------------
```

 ©AquaQ Analytics Limited 2014

```
2013.11.06D12:41:53.103508000 2  add    {system"sleep 2"; x+y} 1 2
2013.11.06D12:42:01.647731000 2  hit    {system"sleep 2"; x+y} 1 2
2013.11.06D12:42:53.930404000 2  rerun  {system"sleep 2"; x+y} 1 2
```

See .api.p".cache.*"for more details.

6.7 timezone.q

A slightly customised version of the timezone conversion functionality from code.kx[1]. It loads a table of timezone information from $KDBCONFIG. See .api.p".tz.*"for more details.

6.8 help.q

The standard help.q from code.kx provides help utilities in the console. This should be kept up to date with code.kx[2].

```
q)help`
adverb    | adverbs/operators
attributes| data attributes
cmdline   | command line parameters
data      | data types
define    | assign, define, control and debug
dotz      | .z locale contents
errors    | error messages
save      | save/load tables
syscmd    | system commands
temporal  | temporal - date & time casts
verbs     | verbs/functions
```

6.9 Additional Utilities

There are some additional user contributed utility scripts available on code.kx which are good candidates for inclusion. These could either be dropped into the common code directory, or if not globally applicable then in the code directory for either the process type or name.

This is not an exhaustive list. The full set of user contributed code is documented here[3]. Some examples with general or common applicability include:

[1] http://code.kx.com/wiki/Cookbook/Timezones
[2] http://code.kx.com/wsvn/code/kx/kdb+/d/help.q
[3] http://code.kx.com/wiki/Contrib

 ©AquaQ Analytics Limited 2014

Functionality	Location	Description
Debugger	http://code.kx.com/wiki/Contrib/debugQ	A command line debugger for q, with optional web interface
Compression	http://code.kx.com/wsvn/code/contrib/simon/compress	Utilities for compressing databases and retrieving compression statistics
Statistical Functions (QML)	http://althenia.net/qml	A set of useful mathematical functions from the FDLIBM, Cephes, LAPACK and CONMAX libraries
CSV Loading	http://code.kx.com/wsvn/code/contrib/simon/csvguess/	Utilities for reading in CSV files

Table 6.1: Additional Utility Scripts

6.10 Full API

The full public api can be found by running

```
q).api.u`
name              | vartype  namespace public descrip                 ..
------------------| ------------------------------------------------..
.proc.createlog   | function .proc     1      "Create the standard out..
.proc.rolllogauto| function .proc     1      "Roll the standard out/e..
.proc.loadf       | function .proc     1      "Load the specified file..
.proc.loaddir     | function .proc     1      "Load all the .q and .k ..
.lg.o             | function .lg       1      "Log to standard out"   ..
..
```

Combined with the commented configuration file, this should give a good overview of the functionality available. A description of the individual namespaces is below- run .api.u “namespace*”to list the functions.

 ©AquaQ Analytics Limited 2014

Namespace	Description
.proc	Process API
.lg	Standard out/error logging API
.err	Error throwing API
.usage	Usage logging API
.access	Permissions API
.clients	Client tracking API
.servers	Server tracking API
.async	Async communication API
.timer	Timer API
.cache	Caching API
.tz	Timezone conversions API
.ps	Publish and Subscribe API
.hb	Heartbeating API
.api	API management API

Table 6.2: Full API

AquaQ

 ©AquaQ Analytics Limited 2014

Chapter 7

Processes

A set of processes is included. These processes build upon AquaQ TorQ, providing specific functionality. All the process scripts are contained in $KDBCODE/processes. All processes should have an entry in $KDBCONFIG/process.csv. All processes can have any type and name, except for discovery services which must have a process type of "discovery". An example process.csv is:

```
aquaq$ cat config/process.csv
host,port,proctype,procname
aquaq,9998,rdb,rdb_europe_1
aquaq,9997,hdb,rdb_europe_1aquaq,9999,hdb,hdb1
aquaq,9996,discovery,discovery1
aquaq,9995,discovery,discovery2
aquaq,8000,gateway,gateway1
aquaq,5010,tickerplant,tickerplant1
aquaq,5011,rdb,rdb1
aquaq,5012,hdb,hdb1
aquaq,5013,hdb,hdb2
aquaq,9990,tickerlogreplay,tpreplay1
aquaq,20000,kill,killhdbs
aquaq,20001,monitor,monitor1
```

7.1 Discovery Service

7.1.1 Overview

Processes use the discovery service to register their own availability, find other processes (by process type) and subscribe to receive updates for new process availability (by process type). The discovery service does not manage connections- it simply returns tables of registered processes, irrespective of their are current availability. It is up to each individual process to manage its own connections.

The discovery service uses the process.csv file to make connections to processes on start up. After start up it is up to each individual process to attempt connections and register with the discovery service. This is done automatically, depending on the configuration parameters. Multiple discovery services can be run in which case each process will try to register and retrieve process details from each discovery process it

finds in its process.csv file. Discovery services do not replicate between themselves. A discovery process must have its process type listed as discovery.

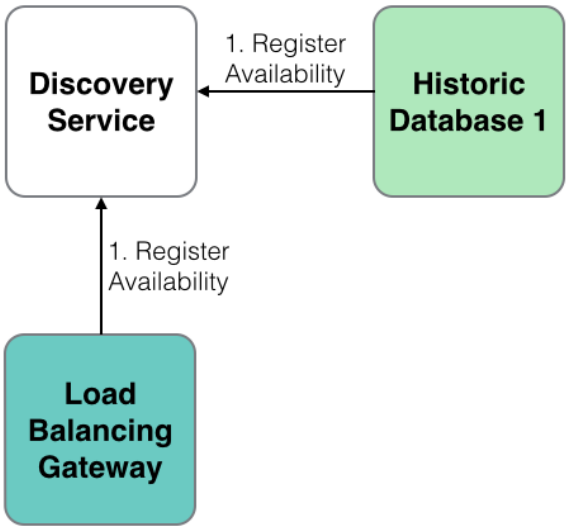
To run the discovery service, use a start line such as:

```
aquaq $ q torq.q -load code/processes/discovery.q -p 9995
```

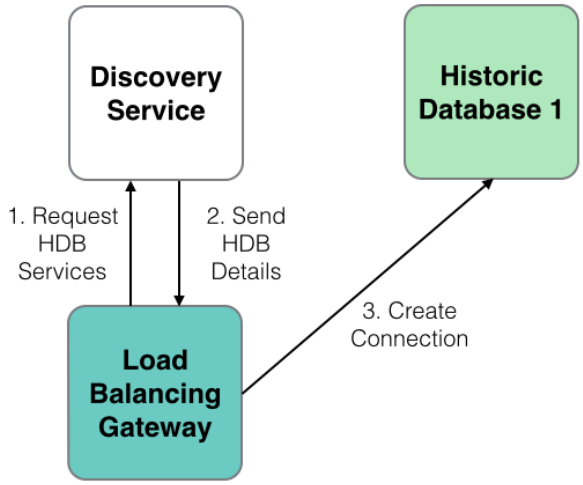
Modify the configuration as required.

7.1.2 Operation

1. Processes register with the discovery service.

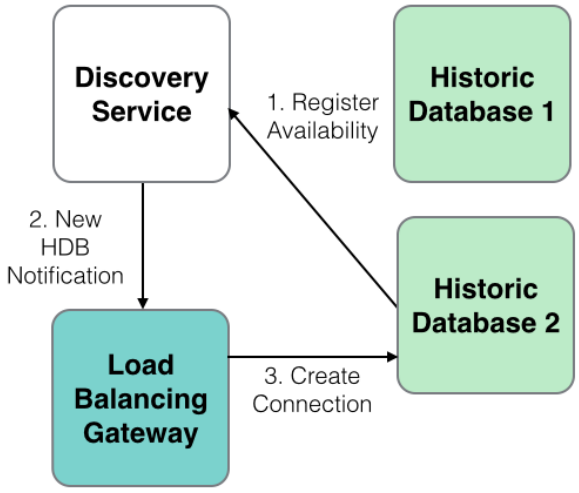


2. Processes use the discovery service to locate other processes.



 ©AquaQ Analytics Limited 2014

3. When new services register, any processes which have registered an interest in that process type are notified.



7.1.3 Available Processes

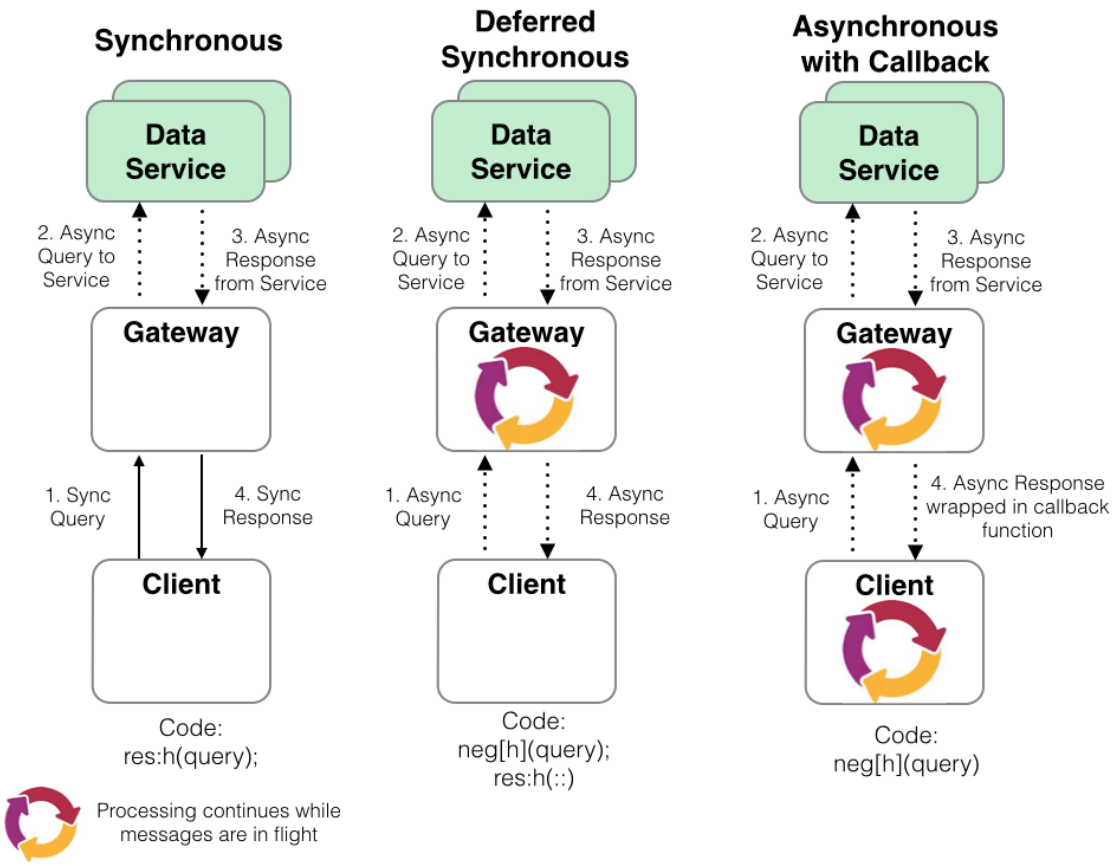
The list of available processes can be found in the .servers.SERVERS table.

```
q).servers.SERVERS
procname     proctype        hpup            w  hits startp
    lastp                          endp attributes
-------------------------------------------------------------------------------------------
discovery1   discovery       :aquaq:9995     0
    2014.01.22D17:00:40.947470000      ()!()
discovery2   discovery       :aquaq:9996     0
    2014.01.22D17:00:40.947517000      ()!()
hdb2         hdb             :aquaq:5013     0
    2014.01.22D17:00:40.947602000      ()!()
killtick     kill            :aquaq:20000    0
    2014.01.22D17:00:40.947602000      ()!()
tpreplay1    tickerlogreplay :aquaq:20002    0
    2014.01.22D17:00:40.947602000      ()!()
tickerplant1 tickerplant     :aquaq:5010  6  0    2014.01.22D17:00:40.967699000
    2014.01.22D17:00:40.967698000      ()!()
monitor1     monitor         :aquaq:20001 9  0    2014.01.22D17:00:40.971344000
    2014.01.22D17:00:40.971344000      ()!()
rdb1         rdb             :aquaq:5011  7  0    2014.01.22D17:06:13.032883000
    2014.01.22D17:06:13.032883000      `date`tables!(,2014.01.22;`fxquotes`heartbeat`
    logmsg`quotes`trades)
hdb3         hdb             :aquaq:5012  8  0    2014.01.22D17:06:18.647349000
    2014.01.22D17:06:18.647349000      `date`tables!(2014.01.13 2014.01.14;`fxquotes`
    heartbeat`logmsg`quotes`trades)
gateway1     gateway         :aquaq:5020  10 0    2014.01.22D17:06:32.152836000
    2014.01.22D17:06:32.152836000      ()!()
```

 ©AquaQ Analytics Limited 2014

7.2 Gateway

A synchronous and asynchronous gateway is provided. The gateway can be used for load balancing and/or to join the results of queries across heterogeneous servers (e.g. an RDB and HDB). Ideally the gateway should only be used with asynchronous calls. Synchronous calls cause the gateway to block so limits the gateway to serving one query at a time (although if querying across multiple backend servers the backend queries will be run in parallel). When using asynchronous calls the client can either block and wait for the result (deferred synchronous) or post a call back function which the gateway will call back to the client with. With both asynchronous and synchronous queries the backend servers to execute queries against are selected using process type. The gateway API can be seen by querying .api.p".gw.*" within a gateway process.



7.2.1 Asynchronous Behaviour

Asynchronous queries allow much greater flexibility. They allow multiple queries to be serviced at once, prioritisation, and queries to be timed out. When an asynchronous query is received the following happens:

 ©AquaQ Analytics Limited 2014

- the query is placed in a queue;
- the list of available servers is retrieved;
- the queue is prioritised, so those queries with higher priority are serviced first;
- queries are sent to back end servers as they become available. Once the backend server returns its result, it is given another query;
- when all the partial results from the query are returned the results are aggregated and returned to the client. They are either returned directly, or wrapped in a callback and posted back asynchronously to the client.

The two main customisable features of the gateway are the selection of available servers (.gw.availableservers) and the queue prioritisation (.gw.getnextqueryid). With default configuration, the available servers are those servers which are not currently servicing a query from the gateway, and the queue priority is a simple FIFO queue. The available servers could be extended to handle process attributes, such as the available datasets or the location of the process, and the queue prioritisation could be modified to anything required e.g. based on the query itself, the username, host of the client etc.

An asynchronous query can be timed out using a timeout defined by the client. The gateway will periodically check if any client queries have not completed in the alotted time, and return a timeout error to the client. If the query is already running on any backend servers then they cannot be timed out other than by using the standard -T flag.

7.2.2 Synchronous Behaviour

When using synchronous queries the gateway can only handle one query at a time and cannot timeout queries other than with the standard -T flag. All synchronous queries will be immediately dispatched to the back end processes. They will be dispatched using an asyhcnronous call, allowing them to run in parallel rather than serially. When the results are received they are aggregated and returned to the client.

7.2.3 Process Discovery

The gateway uses the discovery service to locate processes to query across. The discovery service will notify the gateway when new processes become available and the gateway will automatically connect and start using them. The gateway can also use the static information in process.csv, but this limits the gateway to a predefined list of processes rather than allowing new services to come online as demand requires.

7.2.4 Error Handling

When synchronous calls are used, q errors are returned to clients as they are encountered. When using asynchronous calls there is no way to return actual errors and appropriately prefixed strings must be used instead. It is up to the client to check the

 ©AquaQ Analytics Limited 2014

type of the received result and if it is a string then whether it contains the error prefix. The error prefix can be changed, but the default is "error: ". Errors will be returned when:

- the client requests a query against a server type which the gateway does not currently have any active instances of (this error is returned immediately);
- the query is timed out;
- a back end server returns an error;
- a back end server fails;
- the join function fails.

If postback functions are used, the error string will be posted back within the postback function (i.e. it will be packed the same way as a valid result).

7.2.5 Client Calls

There are four main client calls. The .gw.sync* methods should only be invoked synchronously, and the .gw.async* methods should only be invoked asynchronously. Each of these are documented more extensively in the gateway api. Use .api.p".gw.*" for more details.

Function	Description
.gw.syncexec[query; servertypes]	Execute the specified query synchronously against the required list of servers. If more than one server, the results will be razed.
.gw.syncexecj[query; servertypes; joinfunction]	Execute the specified query against the required list of servers. Use the specified join function to aggregate the results.
.gw.asyncexec[query; servertypes]	Execute the specified query against the required list of servers. If more than one server, the results will be razed. The client must block and wait for the results.
.gw.asyncexecjpt[query; servertypes; joinfunction; postback; timeout]	Execute the specified query against the required list of servers. Use the specified join function to aggregate the results. If the postback function is not set, the client must block and wait for the results. If it is set, the result will be wrapped in the specified postback function and returned asynchronously to the client. The query will be timed out if the timeout value is exceeded.

Table 7.1: Gateway API

For the purposes of demonstration, assume that the queries must be run across an RDB and HDB process, and the gateway has one RDB and two HDB processes available to it.

 ©AquaQ Analytics Limited 2014

```
q).gw.servers
handle| servertype inuse active querycount lastquery                    usage
                  attributes
------| ---------------------------------------------------------------
7     | rdb        0     1      17         2014.01.07D17:05:03.113927000 0D00
    :00:52.149069000 `datacentre`country!`essex`uk
8     | hdb        0     1      17         2014.01.07D17:05:03.113927000 0D00
    :01:26.143564000 `datacentre`country!`essex`uk
9     | hdb        0     1      2          2014.01.07D16:47:33.615538000 0D00
    :00:08.019862000 `datacentre`country!`essex`uk
12    | rdb        0     1      2          2014.01.07D16:47:33.615538000 0D00
    :00:04.018349000 `datacentre`country!`essex`uk
```

Both the RDB and HDB processes have a function f and table t defined. f will run for 2 seconds longer on the HDB processes then it will the RDB.

```
q)f
{system"sleep ",string x+$[`hdb=.proc.proctype;2;0]; t}
q)t
a
----
5013
5014
5015
5016
5017
```

Run the gateway. The main parameter which should be set is the .servers.CONNECTIONS parameter, which dictates the process types the gateway queries across. Also, we need to explicitly allow sync calls. We can do this from the config or from the command line.

```
q torq.q -load code/processes/gateway.q -p 8000 -.gw.synccallsallowed 1 -.servers.
    CONNECTIONS hdb rdb
```

Start a client and connect to the gateway. Start with a sync query. The HDB query should take 4 seconds and the RDB query should take 2 seconds. If the queries run in parallel, the total query time should be 4 seconds.

```
q)h:hopen 8000
q)h(`.gw.syncexec;(`f;2);`hdb`rdb)
a
----
5014
5015
5016
5017
5018
5012
5013
5014
5015
5016
q)\t h(`.gw.syncexec;(`f;2);`hdb`rdb)
4009
```

If a query is done for a server type which is not registered, an error is returned:

```
q)\t h(`.gw.syncexec;(`f;2);`hdb`rdb`other)
'not all of the requested server types are available; missing other
```

Custom join functions can be specified:

 ©AquaQ Analytics Limited 2014

```
q)h(`.gw.syncexecj;(`f;2);`hdb`rdb;{sum{select count i by a from x} each x})
a   | x
----| -
5014| 2
5015| 2
5016| 2
5017| 1
5018| 1
5012| 1
5013| 1
```

Custom joins can fail with appropriate errors:

```
q)h(`.gw.syncexecj;(`f;2);`hdb`rdb;{sum{select count i by b from x} each x})
'failed to apply supplied join function to results: b
```

Asynchronous queries must be sent in async and blocked:

```
q)(neg h)(`.gw.asyncexec;(`f;2);`hdb`rdb); r:h(::)
        /- This white space is from pressing return
        /- the client is blocked and unresponsive

q)q)q)
q)
q)r
a
----
5014
5015
5016
5017
5018
5012
5013
5014
5015
5016
q)
```

We can send multiple async queries at once. Given the gateway has two RDBs and two HDBs avaialble to it, it should be possible to service two of these queries at the same time.

```
q)h:hopen each 8000 8000
q)\t (neg h)@\:(`.gw.asyncexec;(`f;2);`hdb`rdb); (neg h)@\:(::); r:h@\:(::)
4012
q)r
+(,`a)!,5014 5015 5016 5017 5018 5012 5013 5014 5015 5016
+(,`a)!,5013 5014 5015 5016 5017 9999 10000 10001 10002 10003
```

Alternatively async queries can specify a postback so the client does not have to block and wait for the result. The postback function must take two parameters- the first is the function that was sent up, the second is the results. The postback can either be a lambda, or the name of a function.

```
q)h:hopen 8000
q)handleresults:{-1(string .z.z)," got results"; -3!x; show y}
q)(neg h)(`.gw.asyncexecjpt;(`f;2);`hdb`rdb;raze;handleresults;0Wn)
q)
q)      /- These q prompts are from pressing enter
q)      /- The q client is not blocked, unlike the previous example
q)
```

 ©AquaQ Analytics Limited 2014

```
q)2014.01.07T16:53:42.481 got results
a
----
5014
5015
5016
5017
5018
5012
5013
5014
5015
5016

/- Can also use a named function rather than a lambda
q)(neg h)(`.gw.asyncexecjpt;(`f;2);`hdb`rdb;raze;`handleresults;0Wn)
q)
q)
q)2014.01.07T16:55:12.235 got results
a
----
5014
5015
5016
5017
5018
5012
5013
5014
5015
5016
```

Asynchronous queries can also be timed out. This query will run for 22 seconds, but should be timed out after 5 seconds. There is a tolerance of +5 seconds on the timeout value, as that is how often the query list is checked. This can be reduced as required.

```
q)(neg h)(`.gw.asyncexecjpt;(`f;20);`hdb`rdb;raze;();0D00:00:05); r:h(::)


q)q)q)r
"error: query has exceeded specified timeout value"
q)\t (neg h)(`.gw.asyncexecjpt;(`f;20);`hdb`rdb;raze;();0D00:00:05); r:h(::)
6550
```

7.2.6 Non kdb+ Clients

All the examples in the previous section are from clients written in q. However it should be possible to do most of the above from non kdb+ clients. The officially supported APIs for Java, C# and C allow the asynchronous methods above. For example, we can modify the try block in the main function of the Java Grid Viewer[1]:

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.io.IOException;
import java.lang.reflect.Array;
import java.util.logging.Level;
```

[1] http://code.kx.com/wiki/Cookbook/InterfacingWithJava

 ©AquaQ Analytics Limited 2014

```
import java.util.logging.Logger;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
import kx.c;

public class Main {
    public static class KxTableModel extends AbstractTableModel {
        private c.Flip flip;
        public void setFlip(c.Flip data) {
            this.flip = data;
        }

        public int getRowCount() {
            return Array.getLength(flip.y[0]);
        }

        public int getColumnCount() {
            return flip.y.length;
        }

        public Object getValueAt(int rowIndex, int columnIndex) {
            return c.at(flip.y[columnIndex], rowIndex);
        }

        public String getColumnName(int columnIndex) {
            return flip.x[columnIndex];
        }
    };

    public static void main(String[] args) {
        KxTableModel model = new KxTableModel();
        c c = null;
        try {
            c = new c("localhost", 8000,"username:password");
            // Create the query to send
            String query=".gw.asyncexec[(`f;2);`hdb`rdb]";
            // Send the query
            c.ks(query);
            // Block on the socket and wait for the result
            model.setFlip((c.Flip) c.k());
        } catch (Exception ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            if (c != null) {try{c.close();} catch (IOException ex) {}
          }
        }
        JTable table = new JTable(model);
        table.setGridColor(Color.BLACK);
        String title = "kdb+ Example - "+model.getRowCount()+" Rows";
        JFrame frame = new JFrame(title);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new JScrollPane(table), BorderLayout.CENTER);
        frame.setSize(300, 300);
        frame.setVisible(true);
    }
}
```

Some of the unofficially supported APIs may only allow synchronous calls to be made.

 ©AquaQ Analytics Limited 2014

7.3 Tickerplant Log Replay

The Tickerplant Log Replay script is for replaying tickerplant logs. This is useful for:

1. handling end of day save down failures;
2. handling large volumes of data (larger than can fit into RAM).

The process takes as the main input either an individual log file to replay, or a directory containing a set of log files. Amongst other functionality, the process can:

- replay specific message ranges;
- replay in manageable message chunks;
- ignore specific tables;
- modify the tables before or after they are saved;
- apply sorting and parting after all the data is written out.

The process must have some variables set (the tickerplant log file or directory, the schema file, and the on-disk database directory to write to) or it will fail on startup. These can either be set in the config file, or overridden from the command line in the usual way. An example start line would be:

```
q torq.q -debug -load code/processes/tickerlogreplay.q -p 9990 -.replay.tplogfile ../
    test/tplogs/marketdata2013.12.17 -.replay.schemafile ../test/marketdata.q -.
    replay.hdbdir ../test/hdb1
```

The tickerplant log replay script has extended usage information which can be accessed with -.replay.usage.

```
q torq.q -debug -load code/processes/tickerlogreplay.q -p 9990 -.replay.usage
```

7.4 Monitor

The Monitor process is a simple process to monitor the health of the other processes in the system. It connects to each process that it finds (by default using the discovery service, though can use the static file as well) and subscribes to both heartbeats and log messages. It maintains a keyed table of heartbeats, and a table of all log messages received.

Run it with:

```
aquaq $ q torq.q -load code/processes/monitor.q -p 20001
```

It is probably advisable to run the monitor process with the -trap flag, as there may be some start up errors if the processes it is connecting to do not have the necessary heartbeating or publish/subscribe code loaded.

```
aquaq $ q torq.q -load code/processes/monitor.q -p 20001 -trap
```

 ©AquaQ Analytics Limited 2014

The current heartbeat statuses are tracked in .hb.hb, and the log messages in logmsg

```
q)show .hb.hb
sym       procname    | time                          counter warning error
----------------------| ----------------------------------------------------
discovery discovery2  | 2014.01.07D13:24:31.848257000 893     0       0
hdb       hdb1        | 2014.01.07D13:24:31.866459000 955     0       0
rdb       rdb_europe_1| 2014.01.07D13:23:31.507203000 901     1       0
rdb       rdb1        | 2014.01.07D13:24:31.848259000 34      0       0

q)show select from logmsg where loglevel=`ERR
time                          sym  host  loglevel id      message
-------------------------------------------------------------------------------
2014.01.07D12:25:17.457535000 hdb1 aquaq ERR      reload  "failed to reload database"
2014.01.07D13:29:28.784333000 rdb1 aquaq ERR      eodsave "failed to save tables :
    trade, quote"
```

7.5 Kill

The kill process is used to connect to and terminate currently running processes. It kills the process by sending the exit command therefore the kill process must have appropriate permissions to do send the command, and it must be able to create a connection (i.e. it will not be able to kill a blocked process in the same way that the unix command kill -9 would). By default, the kill process will connect to the discovery service(s), and kill the processes of the specified types. The kill process can be modified to not use the discovery service and instead use the process.csv file via the configuration in the standard way.

If run without any command line parameters, kill.q will try to kill each process it finds with type defined by its .servers.CONNECTIONS variable.

```
q torq.q -load code/processes/kill.q -p 20000
```

.servers.CONNECTIONS can optionally be overridden from the command line (as can any other process variable):

```
q torq.q -load code/processes/kill.q -p 20000 -.servers.CONNECTIONS rdb tickerplant
```

The kill process can also be used to kill only specific named processes within the process types:

```
q torq.q -load code/processes/kill.q -p 20000 -killnames hdb1 hdb2
```

 ©AquaQ Analytics Limited 2014

Chapter 8

Integration with kdb+tick

AquaQ TorQ can be fully integrated with kdb+tick. It can be used to extend and enhance the functionality of kdb+tick, but for the purposes of demonstration we will use un-modified kdb+tick (2.7) scripts. We will use as a basis the example start lines from the bottom of tick.q.

```
>q tick.q sym  .  -p 5010       /tick
>q tick/r.q :5010 -p 5011       /rdb
>q sym            -p 5012       /hdb
```

There are some pre-built scripts provided for kdb+tick. kdb+tick should be unzipped and placed in the root directory i.e. in the same location as torq.q. The root directory should contain torq.q, tickerplant.q, tick.q and the tick directory. u.q should also be copied to the the common code directory to enable all processes to implement publish and subscribe.

All the process start lines in this section are applicable to unix based systems. For Windows systems the redirection lines should be modified.

8.1 process.csv

To start, we need to add each of the above processes to the process.csv file, giving them an appropriate type and name.

```
aquaq$ cat config/process.csv
host,port,proctype,procname
aquaq,5010,tickerplant,tickerplant1
aquaq,5011,rdb,rdb1
aquaq,5012,hdb,hdb1
aquaq,5013,hdb,hdb2
aquaq,5020,gateway,gw1
aquaq,9995,discovery,discovery1
aquaq,9996,discovery,discovery2
aquaq,20000,kill,killtick
```

AQUAQ

8.2 Tickerplant

The tickerplant is a latency sensitive application. As such we should be very careful with message handlers. It does not do any harm to load extra code into the tickerplant as long as it not invoked, but for the sake of the example we will only load the minimum code required. We actually do not want AquaQ TorQ to do too much modification-really we just want it for log file redirection, and to register client connections.

We can do all this in two steps. The first is create a wrapper script to load the tickerplant followed by torq.q. torq.q must be loaded second, as otherwise the .z.pc definition is modified by tick.q.

```
aquaq$ cat tickerplant.q
\l tick.q
\l torq.q
```

The second is to create some bespoke config for processes of type tickerplant. We do this by adding $KDBCONFIG/settings/tickerplant.q, and overriding some of the variables defined in $KDBCONFIG/settings/default.q. Specifically, we do not want to load any extra code, and the only message handlers we want to load are the client tracking ones, and only invoke them when connections are opened or closed. If required, it would be feasible to also add access controls but only check them when a connection is opened.

```
aquaq$ cat ../config/tickerplant.q
/- tickerplant configuration

/- Process initialisation
\d .proc
loadcommoncode:0b               /- do not load common code
loadprocesscode:0b              /- do not load process code
loadnamecode:0b                 /- do not load name code
loadhandlers:1b                 /- load the message handles (but switch most off)
logroll:0b                      /- do not roll logs

/- Configuration used by the usage functions - logging of client interaction
\d .usage
enabled:0b                      /- switch off the usage logging

/- Client tracking configuration
/- This is the only thing we want to do
/- and only for connections being opened and closed
\d .clients
enabled:1b                      /- whether client tracking is enabled
opencloseonly:1b                /- only log open and closing of connections
INTRUSIVE:0b                    /- do not interrogate clients
AUTOCLEAN:1b                    /- clean out old records when handling a close
RETAIN:`long$0D02               /- length of time to retain client information
MAXIDLE:`long$0D                /- no closing of idle connections

/- Access controls
\d .access
enabled:0b                      /- disable access controls

/- Server connection details
\d .servers
enabled:0b                      /- disable server tracking
```

 ©AquaQ Analytics Limited 2014

```
\d .timer
enabled:0b                      /- disable the timer

\d .hb
enabled:0b                      /- disable heartbeating

\d .zpsignore
enabled:0b                      /- disable zpsignore - .z.ps should be empty
```

The standard tickerplant set up requires a schema file to be placed in the tick directory:

```
aquaq$ cat tick/equity.q
trade:([]time:`timestamp$();sym:`g#`symbol$();price:`float$())
```

Run the tickerplant. The parameters are the same parameters to the tick.q- the schema file and the log directory.

```
aquaq$q tickerplant.q equity hdb -p 5010 </dev/null >$KDBLOG/torqtp.txt 2>&1 &
```

The initial log messages are written to $KDBLOG/torqtp.txt. Once torq.q is initialised, appropriate log files and aliases are created.

```
aquaq$ls -lrt $KDBLOG/*tickerplant*
-rw-r--r--  1 aquaq staff    126  6 Nov 14:46 /torqhome/logs/err_tickerplant1_2013
    .11.06.log
lrwxr-xr-x  1 aquaq  staff     31  6 Nov 15:29 /torqhome/logs/out_tickerplant1.log ->
     out_tickerplant1_2013.11.06.log
lrwxr-xr-x  1 aquaq  staff     31  6 Nov 15:29 /torqhome/logs/err_tickerplant1.log ->
     err_tickerplant1_2013.11.06.log
-rw-r--r--  1 aquaq  staff  14831  6 Nov 15:29 /torqhome/logs/out_tickerplant1_2013
    .11.06.log
```

8.3 RDB

For the purposes of this example, we are going to assume that the RDB is not as latency sensitive and as such we should be able to use the whole framework, with default values. The RDB still requires command line parameters in a specific order, but fortunately we can load it directly, as long as we put the tickerplant port and hdb port parameters in the correct place. If the RDB is latency senstive, the configuration can be modified in a similar way to the tickerplant.

To run an RDB within AquaQ TorQ:

```
aquaq$ q torq.q :5010 :5012 -load tick/r.q -p 5011 </dev/null >$KDBLOG/torqrdb.txt
    2>&1 &
```

All the log files and aliases should have been created including usage logs:

```
aquaq$ ls -lrt /torqhome/logs/*rdb*
lrwxr-xr-x  1 aquaq  staff    25  6 Nov 15:52 /torqhome/logs/usage_rdb1.log ->
    usage_rdb1_2013.11.06.log
lrwxr-xr-x  1 aquaq  staff    23  6 Nov 15:52 /torqhome/logs/out_rdb1.log ->
    out_rdb1_2013.11.06.log
-rw-r--r--  1 aquaq  staff  1453  6 Nov 15:52 /torqhome/logs/torqrdb.txt
-rw-r--r--  1 aquaq  staff     0  6 Nov 15:52 /torqhome/logs/err_rdb1_2013.11.06.log
lrwxr-xr-x  1 aquaq  staff    23  6 Nov 15:52 /torqhome/logs/err_rdb1.log ->
    err_rdb1_2013.11.06.log
-rw-r--r--  1 aquaq  staff   130  6 Nov 15:52 /torqhome/logs/usage_rdb1_2013.11.06.
    log
-rw-r--r--  1 aquaq  staff  7954  6 Nov 15:52 /torqhome/logs/out_rdb1_2013.11.06.log
```

 ©AquaQ Analytics Limited 2014

8.4 HDB

The HDB is exactly the same as the RDB. It can be invoked directly from torq.q and we can use the full framework. However, we do not need to specify any parameters, just the HDB directory.

```
aquaq$q torq.q -load hdb/equity -p 5012 </dev/null >$KDBLOG/torqhdb.txt 2>&1 &
```

8.5 Discovery Service

A discovery service can be run to allow other processes to locate the tickerplant(s), rdb(s) and hdb(s). Remember to add the discovery service to process.csv.

```
q torq.q -load code/processes/discovery.q -p 9995 </dev/null >$KDBLOG/torqdiscovery.
    txt 2>&1 &
```

8.6 Gateway

A gateway can be added. In this example we are adding a gateway to query across the HDB and RDB. Multiple gateways with different characteristics can be added as required.

```
q torq.q -load code/processes/gateway.q -p 5020 -.servers.CONNECTIONS hdb rdb </dev/
    null >$KDBLOG/torqgw.txt 2>&1 &
```

8.7 Kill the System

The system can be killed using the kill process, assuming the discovery service is running e.g.

```
q torq.q -load code/processes/kill.q -p 20000 -.servers.CONNECTIONS rdb tickerplant
    hdb </dev/null >$KDBLOG/torqkill.txt 2>&1 &
```

8.8 Debugging

To debug any of the processes in the foreground, simply drop the redirection lines and add the debug flag e.g.

```
aquaq$q torq.q -load hdb/equity -p 5012 -debug
```

8.9 Tick Modifications

Some modifications are advisable to ensure smooth running of the system:

 ©AquaQ Analytics Limited 2014

- If using heartbeating and/or log message publication, the RDB end-of-day function should be modified to ensure the heartbeat and logmsg table are not saved down at end-of-day. If they are saved, then the heartbeating and log publishing functionality of the HDB may fail to work as the HDB will regard these tables as splayed and will not be able to publish them using the standard pub/sub functionality. A function, .rdb.moveandclear, is provided to aid this. It should be invoked prior to the end-of-day job to move these two tables to a different namespace, and then move them back when complete;
- The RDB can be modified to reload multiple HDBs (i.e. multiple HDB processes accessing the same data) at day end. The RDB can use the discovery service to find all the registered HDB processes, and reload them all.

AquaQ

 ©AquaQ Analytics Limited 2014

Chapter 9

What Can We Do For You?

AquaQ are a leading provider of kdb+ support, training and consultancy. Our staff have many years of experience architecting and implementing kdb+ systems. We would be happy to engage with you either implementing and customizing AquaQ TorQ, or in bespoke development and support of incumbent systems. Areas that we can assist include:

- Schema Design: deciding the best schema to capture, store and analyse your data;
- Real Time Data Processing: process and act on live data as fast as possible;
- Gateway Design: transparent access across heterogenous data sources e.g. real-time databases and historic database. Load balancing across homogeneous resources;
- Resilience: no single points of failure. Disaster recovery strategies;
- Massive Data Management: strategies to minimise the system memory footprint whilst maintaining access to the data;
- System Stabilisation: ensuring system stability, resolving system bottlenecks.
- Quantitative Analysis: helping you get the most from your data.

Our experience to date is predominantly in the Capital Markets industry. However, our expertise in system architecture and data analysis techniques will extend across domains into other sectors. Please contact us to talk to one of our experts.

9.1 Feedback

Please submit suggestions, improvements, bug reports and support queries to

`http://support.aquaq.co.uk`

AquaQ