# **Advanced Language Interfaces**

## **C API**

**1)**

-the k.h header file must be included in the C source file.

-Depending of the application type, one of the following library should be used: l32/c.o s32/c.o w32/c.dll(c.lib) q.lib

**2)**

-shared libraries (linked with q.lib) which can be dynamically loaded into q servers.

-standalone applications (servers/clients) which communicate with a q process.

**3)**

Both ways are possible: from q we can call functions defined in C and from C we can call q defined functions.

**4)**

-a connection handle to the remote q process should be obtained calling

int c=khp("host",port); (or khpu if credential are required)

-execute queries on the remote process using r=k(c[,"q\_string"[,x[,y[,z]]]],(K)0); where x, y z are optional parameters.

**5)**

Both types of connections are allowed. To specify an asynchronous connection, the negative connection handle must be used when calling a k(-c,…) function. In this way the process won’t wait for the result of remote execution.

**6)**

The connection is broken if k(-c,..) returns zero, otherwise it is a success. Other error types are not checked on this type of connection.

**7)**

r=k(c,..); The response is a K object of K error type (-128). Ks(r) can be used to report the error string.

**8)**

Yes. We can extend the q functions with new ones defined in C and linked as a shared library. During runtime execution, we can map those function in a q process.

**9)**

f:`:path/shared\_lib 2:(`sum;3) will map in q a function “sum” with three parameters which will be reference in the q process as “f”

**10)**

Yes, but we must provide “0” as the connection handle using

k(0,“q\_func”,[optional params,]0)

**11)**

No more than 3 args must be sent to a function. If need more they have to be stored in list(s),

**12)**

All the K types are represented using a unique structure in C. It is defined in k.h as “struct k0” and mainly has a field for K object type (see table below), a reference filed specifying if there are some other K objects using it and a union with fields for value specific to all available K types.

**13)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **q\_id** | **q type** | **type** | **bytes** | **C type** | **creation function** | **vector accessor** |
| 1 | boolean | B | 1 | char | Kb | kG |
| 4 | Byte | G | 1 | char | Kg | kG |
| 5 | Short | H | 2 | short | Kh | kH |
| 6 | Int | I | 4 | int | Ki | kI |
| 7 | Long | J | 8 | int64 | Kj | kJ |
| 8 | Real | E | 4 | float | Ke | kE |
| 9 | Float | F | 8 | double | Kf | kF |
| 10 | Char | C | 1 | char | Kc | kC |
| 11 | Symbol | S | 4 | 8 | char \* | Ks | kS |
| 13 | Month | M | 4 | int | - | kI |
| 14 | Date | D | 4 | int | Kd | kI |
| 15 | datetime | Z | 8 | double | Kz | kF |
| 17 | Minute | U | 4 | int | - | kI |
| 18 | Second | V | 4 | int | - | kI |
| 19 | Time | T | 4 | int | Kt | kI |

**14)**

- kb, kg, kh, … kt: receive a parameter in C type and return an atomic K object of related K type;

- ktn(I t,I n): return a K vector with “n” elements (atoms) of “t” type;

- knk(I n, K...): return a nested list of “n” elements composed by the following K parameters;

- ja(K\*k,V\*a): add an atom “a” to a “k” vector;

- js(K\* k,S s): add a symbol “s” to a “k” object;

- jk(K\*k,K k1): join “k” with “k1”

- k(I c,char\* query,...): execute “query” in the q process specified by the “c” handle

- xT(K k): create a table from the “k” dictionary;

- xD(K k1,K k2): create a dictionary using “k1” symbols and “k2” values

- r1(K k): increase the reference for “k” object. Useful when passing k object defined in C as queries parameters.

- r0(K k): decrease the “k” object reference and release the object if it’s the case;

- I khpu(char\* host,I port,char\*user),khp(char\*host,I port): connect to a q process and return the connection handle;

* ss(S s): create internal string representation (hashing).
* krr(S s): return the “s” error.

**15)**

****