## JSON

JSON can hold more complex structures than CSV files which is useful.

However this can also introduce some added complexity during ingestion.

### Datatypes

Data brought from JSON to kdb+ will only ever come as one of:

* String
* Float
* Boolean

This means as well as parsing the data from JSON often we will want to cast to a more suitable datatype.

Take this example converting a long in kfb+ to JSON using .j.j and parsing it back with .j.k

//Roundtrip fails - the input does not equal the output  
6~.j.k .j.j 6

0b

//The problem comes from all numerics in JSON being converted to floats  
.j.k .j.j 6

6f

* <https://code.kx.com/v2/ref/dotj>

### JSON table encoding

//Create a sample table  
tab:([] longCol:1 2;  
 floatCol:4 5f;  
 symbolCol:`b`h;  
 stringCol:("bb";"dd");  
 dateCol:2018.11.23 2018.11.23;  
 timeCol:00:01:00.000 00:01:00.003)  
tab

longCol floatCol symbolCol stringCol dateCol timeCol   
------------------------------------------------------------  
1 4 b "bb" 2018.11.23 00:01:00.000  
2 5 h "dd" 2018.11.23 00:01:00.003

meta tab

c | t f a  
---------| -----  
longCol | j   
floatCol | f   
symbolCol| s   
stringCol| C   
dateCol | d   
timeCol | t

//Round trip to JSON results in many differences  
.j.k .j.j tab  
meta .j.k .j.j tab

longCol floatCol symbolCol stringCol dateCol timeCol   
----------------------------------------------------------------  
1 4 ,"b" "bb" "2018-11-23" "00:01:00.000"  
2 5 ,"h" "dd" "2018-11-23" "00:01:00.003"  
  
  
  
  
  
  
  
c | t f a  
---------| -----  
longCol | f   
floatCol | f   
symbolCol| C   
stringCol| C   
dateCol | C   
timeCol | C

//Use lower case casts on numerics and captial case tok on string type data  
//\* will leave a column untouched  
flip "j\*S\*DT"$flip .j.k .j.j tab  
tab~flip "j\*S\*DT"$flip .j.k .j.j tab

data | "26cd02c57f9db87b1df9f2e7bb20cc7b"  
expiry | 1.527797e+009  
requestID| ,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
result | "success"  
message | "success"  
receipt | 123154 4646646f  
requestID| ,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
receipt | 1.234568e+007 9.875147e+007  
requestID| ,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
data | "26cd02c57f9db87b1df9f2e7bb20cc7b"  
requestID| ,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
  
  
  
  
  
longCol floatCol symbolCol stringCol dateCol timeCol   
------------------------------------------------------------  
1 4 b "bb" 2018.11.23 00:01:00.000  
2 5 h "dd" 2018.11.23 00:01:00.003  
  
  
  
  
receipt | 1.234568e+007 9.875147e+007  
requestID| ,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
listSize| 2f  
list | "lzplogjxokyetaeflilquziatzpjagsginnajfpbkomfancdmhmumxhazblddhcc"  
requestID| "b4a566eb-2529-5cf4-1327-857e3d73653e"  
  
  
  
  
  
1b

Instead of using flip and having to specify \* to leave a column untouched we can write a helper function.

We can pass it a dictionary with the rules we need to perform

helper:{[t;d] ![t;();0b;key[d]!{($;x;y)}'[value d;key d]]}  
  
castRules:`longCol`symbolCol`dateCol`timeCol!"jSDT"  
  
tab~helper[;castRules] .j.k .j.j tab

data | ,"26cd02c57f9db87b1df9f2e7bb20cc7b"  
expiry | 17682D19:58:45.000000000  
requestID| ,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
result | `success  
message | "success"  
receipt | 123154 4646646  
requestID| ,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
receipt | 12345678 98751466  
requestID| ,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
data | "26cd02c57f9db87b1df9f2e7bb20cc7b"   
requestID| "b4a566eb-2529-5cf4-1327-857e3d73653e"  
receipt | 12345678 98751466  
requestID| ,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
listSize| 2  
list | "lzplogjxokyetaeflilquziatzpjagsginnajfpbkomfancdmhmumxhazblddhcc"  
requestID| "b4a566eb-2529-5cf4-1327-857e3d73653e"  
  
  
  
  
  
1b

Rather than force the use of $ we can make a more general helper which can be based a monodic function per column

generalHelper:{[t;d] ![t;();0b;key[d]!{(x;y)}'[value d;key d]]}  
  
castRules:`longCol`symbolCol`dateCol`timeCol!({neg "j"$ x};{`$upper x};"D"$;"T"$)  
  
generalHelper[;castRules] .j.k .j.j tab

longCol floatCol symbolCol stringCol dateCol timeCol   
------------------------------------------------------------  
-1 4 B "bb" 2018.11.23 00:01:00.000  
-2 5 H "dd" 2018.11.23 00:01:00.003

* <https://code.kx.com/v2/ref/cast>
* <https://code.kx.com/v2/ref/tok>

### Field based JSON encoding

One common use of JSON is objects (key/value pairs) which parse in kdb+ as dictionaries.

These are useful for storing sparse datasets which do not make sense to have each key becoming a new column.

\c 25 200  
read0 `:sample.json

"{\"data\":\"26cd02c57f9db87b1df9f2e7bb20cc7b\",\"expiry\":1527796725,\"requestID\":[\"b4a566eb-2529-5cf4-1327-857e3d73653e\"]}"  
"{\"result\":\"success\",\"message\":\"success\",\"receipt\":[123154,4646646],\"requestID\":[\"b4a566eb-2529-5cf4-1327-857e3d73653e\"]}"  
"{\"receipt\":[12345678,98751466],\"requestID\":[\"b4a566eb-2529-5cf4-1327-857e3d73653e\"]}"  
"{\"data\":\"26cd02c57f9db87b1df9f2e7bb20cc7b\",\"requestID\":[\"b4a566eb-2529-5cf4-1327-857e3d73653e\"]}"  
"{\"receipt\":[12345678,98751466],\"requestID\":[\"b4a566eb-2529-5cf4-1327-857e3d73653e\"]}"  
"{\"listSize\":2,\"list\":\"lzplogjxokyetaeflilquziatzpjagsginnajfpbkomfancdmhmumxhazblddhcc\"}"  
"{\"requestID\":[\"b4a566eb-2529-5cf4-1327-857e3d73653e\"]}"

One way to manage these items may be to create a utility which will cast any dictionary using keys to control casting rules.

This allows more complex parsing rules for each field.

//Converts JSON to q with rules per key  
decode:{[j]k:.j.k j;(key k)!j2k[key k]@'value k}  
  
//Converts q to JSON with rules per key  
encode:{[k].j.j (key k)!k2j[key k]@'value k}  
  
//Rules for JSON to q conversion  
j2k:(enlist `)!enlist (::);  
  
j2k[`expiry]:{0D00:00:01\*`long$x};  
j2k[`result]:`$;  
j2k[`receipt]:`long$;  
j2k[`id]:{"G"$first x};  
j2k[`listSize]:`long$;  
j2k[`data]:cut[32];  
j2k[`blockCount]:`long$;  
j2k[`blocks]:raze;  
  
//Rules for q to JSON conversion  
k2j:(enlist `)!enlist (::);  
  
k2j[`expiry]:{`long$%[x;0D00:00:01]};  
k2j[`result]:(::);  
k2j[`receipt]:(::);  
k2j[`id]:enlist;  
k2j[`listSize]:(::);  
k2j[`data]:raze;  
k2j[`blocks]:(::);

//Using default .j.k our structures are not transferred as we wish  
{show .j.k x} each read0 `:sample.json;

//Using decode utility captures complex structures  
{show decode x} each read0 `:sample.json;

//The encode utility allows us to round trip  
{sample~{encode decode x} each sample:read0 x}`:sample.json

1b

### Querying unstructured data

With the release of Anymap in kdb+ 3.6 unstructured data has become much easier to manage in kdb+.

However, some considerations do need to be taken in to account.

* <https://code.kx.com/v2/releases/ChangesIn3.6/#anymap>

sample:([] data:decode each read0 `:sample.json)  
sample

data   
-----------------------------------------------------------------------------------------------------------------------------  
`data`expiry`requestID!(,"26cd02c57f9db87b1df9f2e7bb20cc7b";17682D19:58:45.000000000;,"b4a566eb-2529-5cf4-1327-857e3d73653e")  
`result`message`receipt`requestID!(`success;"success";123154 4646646;,"b4a566eb-2529-5cf4-1327-857e3d73653e")   
`receipt`requestID!(12345678 98751466;,"b4a566eb-2529-5cf4-1327-857e3d73653e")   
`data`requestID!(,"26cd02c57f9db87b1df9f2e7bb20cc7b";,"b4a566eb-2529-5cf4-1327-857e3d73653e")   
`receipt`requestID!(12345678 98751466;,"b4a566eb-2529-5cf4-1327-857e3d73653e")   
`listSize`list!(2;"lzplogjxokyetaeflilquziatzpjagsginnajfpbkomfancdmhmumxhazblddhcc")   
(,`requestID)!,,"b4a566eb-2529-5cf4-1327-857e3d73653e"

Indexing at depth allows the sparse data within the dictionaries to be queried easily

select data[;`requestID] from sample

x   
---------------------------------------  
,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
,"b4a566eb-2529-5cf4-1327-857e3d73653e"  
0N   
,"b4a566eb-2529-5cf4-1327-857e3d73653e"

When a key is missing from a dictionary kdb+ will return a null value.

The type of this null is determined by the type of the first key within the dictionary.

This poses an issue.

//Many different nulls are returned  
select data[;`expiry] from sample

x   
------------------------  
17682D19:58:45.000000000  
`   
`long$()   
,""   
`long$()   
0N   
,""

//Succeds on first 2 rows as by chance only null returned in a atom null  
select from (2#sample) where null data[;`expiry]  
//Fails once moving to 3 rows as there is an empty list null  
select from (3#sample) where null data[;`expiry]

data   
-------------------------------------------------------------------------------------------------------------  
`result`message`receipt`requestID!(`success;"success";123154 4646646;,"b4a566eb-2529-5cf4-1327-857e3d73653e")  
  
  
  
  
  
evaluation error:  
  
  
type  
  
  
  
 [0] select from (3#sample) where null data[;`expiry]  
 ^

Checking if a given key is in the dictionary will only return rows which do not have the key.

select from sample where `expiry in/:key each data, not null data[;`expiry]

data   
-----------------------------------------------------------------------------------------------------------------------------  
`data`expiry`requestID!(,"26cd02c57f9db87b1df9f2e7bb20cc7b";17682D19:58:45.000000000;,"b4a566eb-2529-5cf4-1327-857e3d73653e")

If we prepend each dictionary with the null symbol key ``and generic null value(::)` we now can query in a more free manner.

update data:(enlist[`]!enlist (::))(,)/:data from `sample;  
sample

data   
---------------------------------------------------------------------------------------------------------------------------------  
``data`expiry`requestID!(::;,"26cd02c57f9db87b1df9f2e7bb20cc7b";17682D19:58:45.000000000;,"b4a566eb-2529-5cf4-1327-857e3d73653e")  
``result`message`receipt`requestID!(::;`success;"success";123154 4646646;,"b4a566eb-2529-5cf4-1327-857e3d73653e")   
``receipt`requestID!(::;12345678 98751466;,"b4a566eb-2529-5cf4-1327-857e3d73653e")   
``data`requestID!(::;,"26cd02c57f9db87b1df9f2e7bb20cc7b";,"b4a566eb-2529-5cf4-1327-857e3d73653e")   
``receipt`requestID!(::;12345678 98751466;,"b4a566eb-2529-5cf4-1327-857e3d73653e")   
``listSize`list!(::;2;"lzplogjxokyetaeflilquziatzpjagsginnajfpbkomfancdmhmumxhazblddhcc")   
``requestID!(::;,"b4a566eb-2529-5cf4-1327-857e3d73653e")

All nulls when a given key is missing are now (::)

select expiry:data[;`expiry] from sample

expiry   
------------------------  
17682D19:58:45.000000000  
::   
::   
::   
::   
::   
::

The previously failing query can now execute as there are no list type nulls

select from sample where not null data[;`expiry]

data   
---------------------------------------------------------------------------------------------------------------------------------  
``data`expiry`requestID!(::;,"26cd02c57f9db87b1df9f2e7bb20cc7b";17682D19:58:45.000000000;,"b4a566eb-2529-5cf4-1327-857e3d73653e")

These (::) can also be replaced with chosen values easily.

Here an infinite value is chosen:

fill:{@[y;where null y;:;x]}  
select expiry:fill[0Wn]data[;`expiry] from sample

expiry   
------------------------  
17682D19:58:45.000000000  
0W   
0W   
0W   
0W   
0W   
0W