QDF Background

Ramesh Subramonian

February 11, 2026

1 QDF Data Layout

This section defines the layout of data in a QDF

1.1 qtype

The enum `qtype_t` is defined as

```
typedef enum {
  Qerr = 0,
  Q0,
  B1, // boolean stored as bit
  BL, // boolean stored as bool
  I8, // signed 1-byte intege
  I16, // signed 2-byte intege
  I32, // signed 4-byte intege
  I64, // signed 8-byte intege
  BF16, // bfloat16 floating point
  FP32, // single precision floating point
  FP64, // double prevision floating point
  UI8, // unsigned 1-byte integer
  UI16, // unsigned 2-byte integer
  UI32, // unsigned 4-byte integer
  UI64, // unsigned 8-byte integer
  SC,  // consatnt length string
  TM, // struct tm as defined in time.h
} qtype_t;
```

1.2 q2c

The Lua table q2c is a mapping between a value of type `qtype_t` and a C type.

I32	`int32_t`
TM	`struct tm`
FP32	`float`
Q0	`void`

Table 1: Qtypes to C Types

1.3 jtype

The enum `jtype_t` is defined as follows

```
typedef enum {
  j_error = 0,
  j_undef,
  j_nil,
  j_bool,
  j_string,
  j_number,
  j_date,
  j_array,
  j_object,
  // j_hashtable
} jtype_t;
```

1.4 QDF struct

The struct `qdf_rec_type`is defined as follows

```
typedef struct _qdf_rec_type {
  void *data;
  uint32_t size; // must be a multiple of 8
  bool is_mmap; // true => we have mmapped data not malloc'd it
  bool is_foreign; // true => do not free() or munmap()
  bool is_read_only; // true => don't modify
} QDF_REC_TYPE;
```

2 Read only helper functions

Table 2 lists helper functions that have been provided to you. They take as an argument an immutable pointer to `qdf_rec_type` and do **not** modify the location pointed to.

Return	Name	Args
bool	chk_qdf()	(const qdf_rec_type * const x)
qtype_t	get_qtype()	(const qdf_rec_type * const x)
jtype_t	get_jtype()	(const qdf_rec_type * const x)
UI4	get_length()	(const qdf_rec_type * const x)
const char *	get_read_arr_ptr()	(const qdf_rec_type * const x)
const bool *	get_nulls()	(const qdf_rec_type * const x)
UI4	get_num_keys()	(const qdf_rec_type * const x)

Table 2: List of read only helper functions

2.1 chk_qdf()

Returns true if x is syntactically valid; false, otherwise

2.2 get_qtype()

Returns qtype_t of x

2.3 get_jtype()

Returns jtype_t of x

2.4 get_length()

If jtype_t of x is j_array, then returns length of array; else, returns 0.

2.5 get_read_arr_ptr()

If jtype_t of x is j_array, then returns NULL. If qtype_t of x is Qerr or Q0, then returns NULL. Else, returns a pointer to the 0^{th} element of the array. The data pointed to cannot be modified.

2.6 get_nulls()

If jtype_t of x is j_array, then returns NULL. If qtype_t of x is Qerr or Q0, then returns NULL. If x does not have null values, then return NULL. Else, returns a pointer to the 0^{th} element of the bool array, nnx, such that

$$nnx[i] \Rightarrow x[i] \neq \perp \neg nnx[i] \Rightarrow x[i] = \perp$$

The data pointed to cannot be modified.

2.7 get_num_keys()

If jtype_t x is j_object, then returns number of keys; else, returns 0.

3 Coalesce

- Let x be a read only immutable pointer to qdf_rec_type.
- Let y be a read only immutable pointer to qdf_rec_type.
- Let z be a pointer to qdf_rec_type.
- qx := get_qtype()(x)
- qy := get_qtype()(y)
- cqx := q2c[qx] C type corresponding to qx as defined in Lua table q2c
- cqy := q2c[qy] C type corresponding to qy as defined in Lua table q2c
- jx := get_jtype()(X)
- jy := get_jtype()(Y)
- nx := get_length()(X)
- ny := get_length()(Y)
- const cqx * const vx = get_read_arr_ptr()(x)
- const cqy * const vy = get_read_arr_ptr()(y)
- const bool * const nnx = get_nulls()(x)
- const bool * const nny = get_nulls()(y)

```
T = { "coalesce", "_", qx, "_", qy }
func = string.concat(T)
```

Function signature is

1. name is func, specified above.
2. Return value is of type integer. Return value is 0 on success, -1 on failure.
3. Arguments are x, y, z specified above

Function logic is as follows.
Function fails under the conditions listed below

1. Fails if x == NULL
2. Fails if y == NULL
3. Fails if x == y
4. Fails if qx ≠ qy
5. Fails if jx ≠ `j_array`
6. Fails if jy ≠ `j_array`
7. Fails if nx ≠ ny
8. Fails if nx == 0

Let nz := nx Let jz := jx Let qz := qx Create vz as an array of type cqx and length nz Create nnz as an array of type bool and length nz

Execute the following for all values of i in [0 .. nz-1]

if (not nnx == or nnx[i]) then vz[i] := vx[i] nnz[i] := true else if (not nny == or nny[i]) then vz[i] := vy[i] nnz[i] := true else vz[i] := 0 nnz[i] := false endif endif

// Execute the following setter functions // All setters return a int status. If status is not 0, then // calling function executes macro cBYE(-1)

1. `set_jtype()`(z, jz)
2. `set_qtype()`(z, qz)
3. `set_length()`(z, nz)
4. `set_arr_ptr()`(z, vz)
5. `set_nn()`(z, nnz)