cQueue

1.4

Generated by Doxygen 1.8.13

Contents

1 Deprecated List

Global q_clean

q_clean was already used in cQueue lib, alias is made to keep compatibility with earlier versions

Global q_nbRecs

q_nbRecs was already used in cQueue lib, alias is made to keep compatibility with earlier versions

Global q_pull

q_pull was already used in cQueue lib, alias is made to keep compatibility with earlier versions

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Data Structure Documentation

4.1 Queue_t Struct Reference

Queue type structure holding all variables to handle the queue.

```
#include <src/cQueue.h>
```

Data Fields

- QueueType impl

    Queue implementation: FIFO LIFO.
- bool ovw

    Overwrite previous records when queue is full allowed.
- uint16_t rec_nb

    number of records in the queue
- uint16_t rec_sz

    Size of a record.
- uint32_t queue_sz

    Size of the full queue.
- uint8_t ∗ queue

    Queue start pointer (when allocated)
- uint16_t in

    number of records pushed into the queue
- uint16_t out

    number of records pulled from the queue (only for FIFO)
- uint16_t cnt

    number of records not retrieved from the queue
- uint16_t init

    set to QUEUE_INITIALIZED after successful init of the queue and reset when killing queue

4.1.1 Detailed Description

Queue type structure holding all variables to handle the queue.

4.1.2 Field Documentation

4.1.2.1 cnt

`uint16_t Queue_t::cnt`

number of records not retrieved from the queue

4.1.2.2 impl

`QueueType Queue_t::impl`

Queue implementation: FIFO LIFO.

4.1.2.3 in

`uint16_t Queue_t::in`

number of records pushed into the queue

4.1.2.4 init

`uint16_t Queue_t::init`

set to QUEUE_INITIALIZED after successful init of the queue and reset when killing queue

4.1.2.5 out

`uint16_t Queue_t::out`

number of records pulled from the queue (only for FIFO)

4.1.2.6 ovw

```
bool Queue_t::ovw
```

Overwrite previous records when queue is full allowed.

4.1.2.7 queue

```
uint8_t* Queue_t::queue
```

Queue start pointer (when allocated)

4.1.2.8 queue_sz

```
uint32_t Queue_t::queue_sz
```

Size of the full queue.

4.1.2.9 rec_nb

```
uint16_t Queue_t::rec_nb
```

number of records in the queue

4.1.2.10 rec_sz

```
uint16_t Queue_t::rec_sz
```

Size of a record.

The documentation for this struct was generated from the following file:

- src/cQueue.h

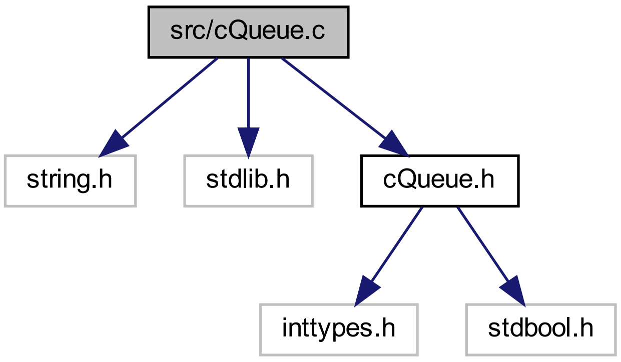
5 File Documentation

5.1 src/cQueue.c File Reference

Queue handling library (designed in c on STM32)

```
#include <string.h>
#include <stdlib.h>
#include "cQueue.h"
```

Include dependency graph for cQueue.c:



Macros

- #define INC_IDX(ctr, end, start)

  *Increments buffer index **ctr** rolling back to **start** when limit **end** is reached.*
- #define DEC_IDX(ctr, end, start)

  *Decrements buffer index **ctr** rolling back to **end** when limit **start** is reached.*

Functions

- void ∗ q_init (Queue_t ∗q, const uint16_t size_rec, const uint16_t nb_recs, const QueueType type, const bool overwrite)

  Queue initialization.
- void q_kill (Queue_t ∗q)

  Queue destructor: release dynamically allocated queue.
- void q_flush (Queue_t ∗q)

  Flush queue, restarting from empty queue.
- bool q_push (Queue_t ∗q, const void ∗record)

  Push record to queue.
- bool q_pop (Queue_t ∗q, void ∗record)

  Pop record from queue.
- bool q_peek (Queue_t ∗q, void ∗record)

  Peek record from queue.
- bool q_drop (Queue_t ∗q)

  Drop current record from queue.

5.1.1 Detailed Description

Queue handling library (designed in c on STM32)

Author

SMFSW

Date

2018/05/26

Copyright

BSD 3-Clause License (c) 2017-2018, SMFSW

Queue handling library (designed in c on STM32)

5.1.2 Macro Definition Documentation

5.1.2.1 DEC_IDX

```
#define DEC_IDX(
             ctr,
             end,
             start )
```

Value:

```
if (ctr > (start))  { ctr--; }      \
                                    else                { ctr = end-1; }
```

Decrements buffer index **ctr** rolling back to **end** when limit **start** is reached.

5.1.2.2 INC_IDX

```
#define INC_IDX(
             ctr,
             end,
             start )
```

Value:

```
if (ctr < (end-1))  { ctr++; }      \
                                    else                { ctr = start; }
```

Increments buffer index **ctr** rolling back to **start** when limit **end** is reached.

5.1.3 Function Documentation

5.1.3.1 q_drop()

```
bool q_drop (
            Queue_t * q )
```

Drop current record from queue.

Warning

If using q_push, q_pop, q_peek and/or q_drop in both interrupts and main application, you shall disable interrupts in main application when using these functions

Parameters

in,out	*q*	- pointer of queue to handle

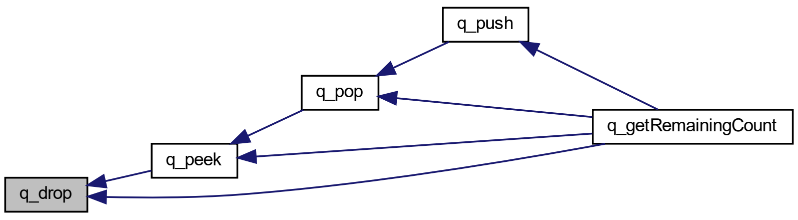
Returns

drop status

Return values

true	if successfully dropped from queue
false	if queue is empty

Here is the caller graph for this function:



5.1.3.2 q_flush()

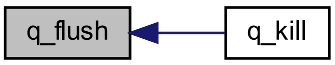
```
void q_flush (
            Queue_t * q )
```

Flush queue, restarting from empty queue.

Parameters

in,out	*q*	- pointer of queue to handle

Here is the caller graph for this function:



5.1.3.3 q_init()

```
void* q_init (
            Queue_t * q,
            const uint16_t size_rec,
            const uint16_t nb_recs,
            const QueueType type,
            const bool overwrite )
```

Queue initialization.

Parameters

in,out	*q*	- pointer of queue to handle
in	*size_rec*	- size of a record in the queue
in	*nb_recs*	- number of records in the queue
in	*type*	- Queue implementation type: FIFO, LIFO
in	*overwrite*	- Overwrite previous records when queue is full

Returns

NULL when allocation not possible, Queue tab address when successful

5.1.3.4 q_kill()

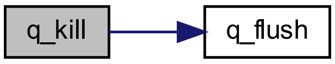
```
void q_kill (
            Queue_t * q )
```

Queue destructor: release dynamically allocated queue.

Parameters

`in,out`	*q*	- pointer of queue to handle

Here is the call graph for this function:



5.1.3.5 q_peek()

```
bool q_peek (
            Queue_t * q,
            void * record )
```

Peek record from queue.

Warning

If using q_push, q_pop, q_peek and/or q_drop in both interrupts and main application, you shall disable interrupts in main application when using these functions

Parameters

`in`	*q*	- pointer of queue to handle
`in,out`	*record*	- pointer to record to be peeked from queue

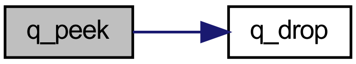
Returns

Peek status

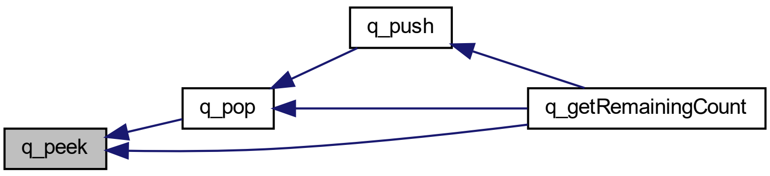
Return values

true	if successfully pulled from queue
false	if queue is empty

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.6 q_pop()

```
bool q_pop (
            Queue_t * q,
            void * record )
```

Pop record from queue.

Warning

If using q_push, q_pop, q_peek and/or q_drop in both interrupts and main application, you shall disable interrupts in main application when using these functions

Parameters

in	*q*	- pointer of queue to handle
in,out	*record*	- pointer to record to be popped from queue

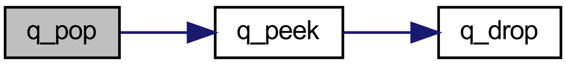
Returns

Pop status

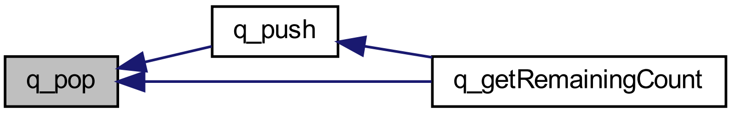
Return values

true	if successfully pulled from queue
false	if queue is empty

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.7 q_push()

```
bool q_push (
            Queue_t * q,
            const void * record )
```

Push record to queue.

Warning

If using q_push, q_pop, q_peek and/or q_drop in both interrupts and main application, you shall disable interrupts in main application when using these functions

Parameters

`in,out`	*q*	- pointer of queue to handle
`in`	*record*	- pointer to record to be pushed into queue

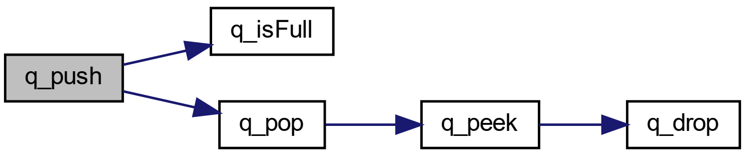
Returns

Push status

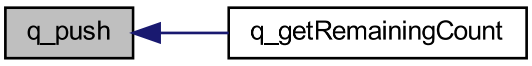
Return values

true	if successfully pushed into queue
false	if queue is full

Here is the call graph for this function:



Here is the caller graph for this function:

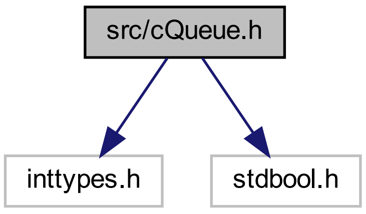


5.2 src/cQueue.h File Reference

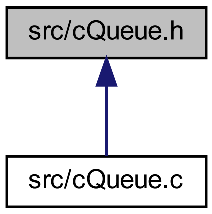
Queue handling library (designed in c on STM32)

```
#include <inttypes.h>
#include <stdbool.h>
```

Include dependency graph for cQueue.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Queue_t

    Queue type structure holding all variables to handle the queue.

Macros

- #define QUEUE_INITIALIZED 0x5AA5

    Queue initialized control value.
- #define q_init_def(q, sz) q_init(q, sz, 20, FIFO, false)

    Some kind of average default for queue initialization.
- #define q_pull q_pop
- #define q_nbRecs q_getCount
- #define q_clean q_flush

Typedefs

- typedef enum enumQueueType QueueType
- typedef struct Queue_t Queue_t

Enumerations

- enum enumQueueType { FIFO = 0, LIFO = 1 }

    Queue behavior enumeration (FIFO, LIFO)

Functions

- void ∗ q_init (Queue_t ∗q, const uint16_t size_rec, const uint16_t nb_recs, const QueueType type, const bool overwrite)

    Queue initialization.
- void q_kill (Queue_t ∗q)

    Queue destructor: release dynamically allocated queue.
- void q_flush (Queue_t ∗q)

    Flush queue, restarting from empty queue.
- bool q_isInitialized (const Queue_t ∗q)

    get initialization state of the queue
- bool q_isEmpty (const Queue_t ∗q)

    get emptiness state of the queue
- bool q_isFull (const Queue_t ∗q)

    get fullness state of the queue
- uint32_t q_sizeof (const Queue_t ∗q)

    get size of queue
- uint16_t q_getCount (const Queue_t ∗q)

    get number of records in the queue
- uint16_t q_getRemainingCount (const Queue_t ∗q)

    get number of records left in the queue
- bool q_push (Queue_t ∗q, const void ∗record)

    Push record to queue.
- bool q_pop (Queue_t ∗q, void ∗record)

    Pop record from queue.
- bool q_peek (Queue_t ∗q, void ∗record)

    Peek record from queue.
- bool q_drop (Queue_t ∗q)

    Drop current record from queue.

5.2.1 Detailed Description

Queue handling library (designed in c on STM32)

Author

SMFSW

Date

2018/05/26

Copyright

BSD 3-Clause License (c) 2017-2018, SMFSW

Queue handling library (designed in c on STM32)

5.2.2 Macro Definition Documentation

5.2.2.1 q_clean

```
#define q_clean q_flush
```

Deprecated q_clean was already used in cQueue lib, alias is made to keep compatibility with earlier versions

5.2.2.2 q_init_def

```
#define q_init_def(
              q,
              sz ) q_init(q, sz, 20, FIFO, false)
```

Some kind of average default for queue initialization.

5.2.2.3 q_nbRecs

```
#define q_nbRecs q_getCount
```

Deprecated q_nbRecs was already used in cQueue lib, alias is made to keep compatibility with earlier versions

5.2.2.4 q_pull

```
#define q_pull q_pop
```

Deprecated q_pull was already used in cQueue lib, alias is made to keep compatibility with earlier versions

5.2.2.5 QUEUE_INITIALIZED

```
#define QUEUE_INITIALIZED 0x5AA5
```

Queue initialized control value.

5.2.3 Typedef Documentation

5.2.3.1 Queue_t

```
typedef struct Queue_t Queue_t
```

5.2.3.2 QueueType

```
typedef enum enumQueueType QueueType
```

5.2.4 Enumeration Type Documentation

5.2.4.1 enumQueueType

```
enum enumQueueType
```

Queue behavior enumeration (FIFO, LIFO)

Enumerator

FIFO	First In First Out behavior.
LIFO	Last In First Out behavior.

5.2.5 Function Documentation

5.2.5.1 q_drop()

```
bool q_drop (
            Queue_t * q )
```

Drop current record from queue.

Warning

If using q_push, q_pop, q_peek and/or q_drop in both interrupts and main application, you shall disable interrupts in main application when using these functions

Parameters

in,out	*q*	- pointer of queue to handle

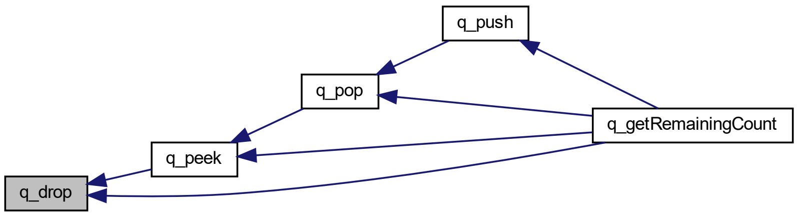
Returns

drop status

Return values

true	if successfully dropped from queue
false	if queue is empty

Here is the caller graph for this function:



5.2.5.2 q_flush()

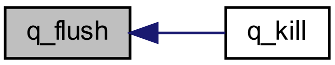
```
void q_flush (
            Queue_t * q )
```

Flush queue, restarting from empty queue.

Parameters

in,out	*q*	- pointer of queue to handle

Here is the caller graph for this function:



5.2.5.3 q_getCount()

```
uint16_t q_getCount (
             const Queue_t * q )  [inline]
```

get number of records in the queue

Parameters

in	*q*	- pointer of queue to handle

Returns

Number of records stored in the queue

5.2.5.4 q_getRemainingCount()

```
uint16_t q_getRemainingCount (
             const Queue_t * q )  [inline]
```

get number of records left in the queue

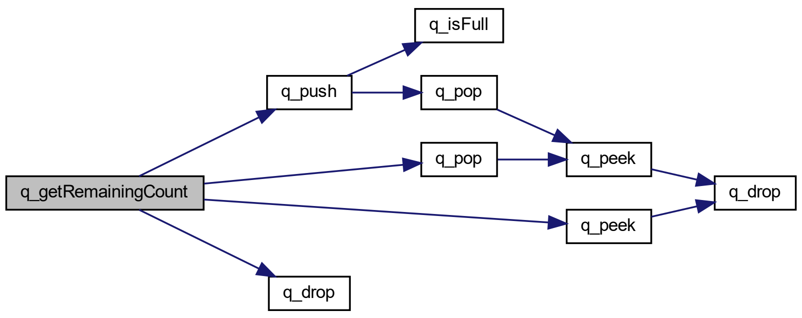
Parameters

in	*q*	- pointer of queue to handle

Returns

Number of records left in the queue

Here is the call graph for this function:



5.2.5.5 q_init()

```
void* q_init (
            Queue_t * q,
            const uint16_t size_rec,
            const uint16_t nb_recs,
            const QueueType type,
            const bool overwrite )
```

Queue initialization.

Parameters

in,out	*q*	- pointer of queue to handle
in	*size_rec*	- size of a record in the queue
in	*nb_recs*	- number of records in the queue
in	*type*	- Queue implementation type: FIFO, LIFO
in	*overwrite*	- Overwrite previous records when queue is full

Returns

NULL when allocation not possible, Queue tab address when successful

5.2.5.6 q_isEmpty()

```
bool q_isEmpty (
            const Queue_t * q )  [inline]
```

get emptiness state of the queue

Parameters

in	*q*	- pointer of queue to handle

Returns

Queue emptiness status

Return values

true	if queue is empty
false	is not empty

5.2.5.7 q_isFull()

```
bool q_isFull (
            const Queue_t * q )  [inline]
```

get fullness state of the queue

Parameters

in	*q*	- pointer of queue to handle

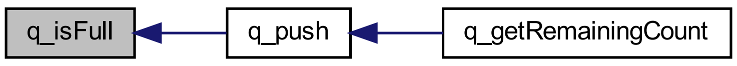
Returns

Queue fullness status

Return values

true	if queue is full
false	is not full

Here is the caller graph for this function:



5.2.5.8 q_isInitialized()

```
bool q_isInitialized (
             const Queue_t * q )  [inline]
```

get initialization state of the queue

Parameters

in	*q*	- pointer of queue to handle

Returns

Queue initialization status

Return values

true	if queue is allocated
false	is queue is not allocated

5.2.5.9 q_kill()

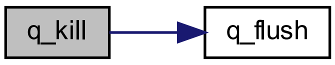
```
void q_kill (
            Queue_t * q )
```

Queue destructor: release dynamically allocated queue.

Parameters

in,out	*q*	- pointer of queue to handle

Here is the call graph for this function:



5.2.5.10 q_peek()

```
bool q_peek (
            Queue_t * q,
            void * record )
```

Peek record from queue.

Warning

If using q_push, q_pop, q_peek and/or q_drop in both interrupts and main application, you shall disable interrupts in main application when using these functions

Parameters

in	*q*	- pointer of queue to handle
in,out	*record*	- pointer to record to be peeked from queue

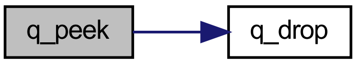
Returns

Peek status

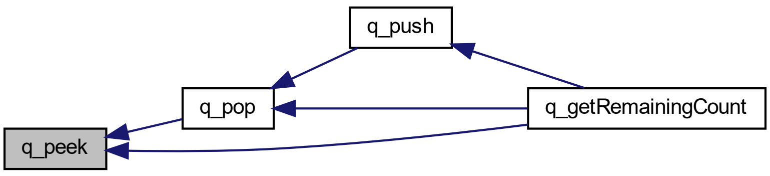
Return values

true	if successfully pulled from queue
false	if queue is empty

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.5.11 q_pop()

```
bool q_pop (
            Queue_t * q,
            void * record )
```

Pop record from queue.

Warning

If using q_push, q_pop, q_peek and/or q_drop in both interrupts and main application, you shall disable interrupts in main application when using these functions

Parameters

`in`	*q*	- pointer of queue to handle
`in,out`	*record*	- pointer to record to be popped from queue

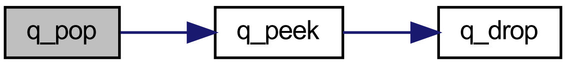
Returns

Pop status

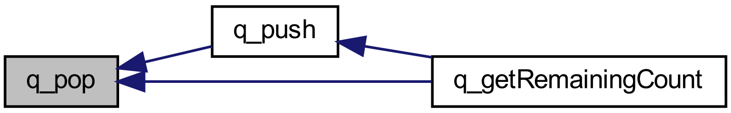
Return values

true	if successfully pulled from queue
false	if queue is empty

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.5.12 q_push()

```
bool q_push (
            Queue_t * q,
            const void * record )
```

Push record to queue.

Warning

If using q_push, q_pop, q_peek and/or q_drop in both interrupts and main application, you shall disable interrupts in main application when using these functions

Parameters

`in,out`	*q*	- pointer of queue to handle
`in`	*record*	- pointer to record to be pushed into queue

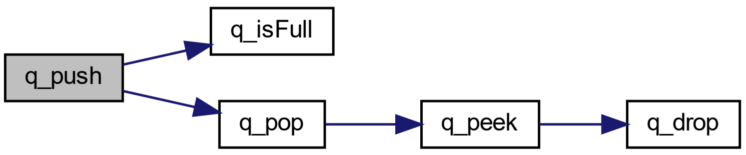
Returns

Push status

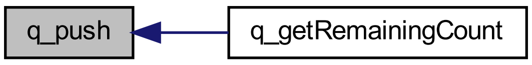
Return values

true	if successfully pushed into queue
false	if queue is full

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.5.13 q_sizeof()

```
uint32_t q_sizeof (
            const Queue_t * q )  [inline]
```

get size of queue

Remarks

Size in bytes (like sizeof)

Parameters

in	*q*	- pointer of queue to handle

Returns

Size of queue in bytes

Index