cQueue

1.2

Generated by Doxygen 1.8.13

Contents

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

examples/LibTst/LibTst.ino	4
examples/RolloverTest/RolloverTest.ino	4
examples/SimpleQueue/SimpleQueue.ino	4
src/cQueue.c **Queue handling library (designed in c on STM32)**	4
src/cQueue.h **Queue handling library (designed in c on STM32)**	12

3 Class Documentation

3.1 Queue_t Struct Reference

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

Public Attributes

- 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.
- 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

  sets to 0x5A5A after a first init of the queue

3.1.1 Member Data Documentation

3.1.1.1 cnt

`uint16_t Queue_t::cnt`

number of records not retrieved from the queue

3.1.1.2 impl

`QueueType Queue_t::impl`

Queue implementation: FIFO LIFO.

3.1.1.3 in

`uint16_t Queue_t::in`

number of records pushed into the queue

3.1.1.4 init

`uint16_t Queue_t::init`

sets to 0x5A5A after a first init of the queue

3.1.1.5 out

`uint16_t Queue_t::out`

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

3.1.1.6 ovw

`bool Queue_t::ovw`

Overwrite previous records when queue is full allowed.

3.1.1.7 queue

`uint8_t* Queue_t::queue`

Queue start pointer (when allocated)

3.1.1.8 rec_nb

`uint16_t Queue_t::rec_nb`

number of records in the queue

3.1.1.9 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

4 File Documentation

4.1 examples/LibTst/LibTst.ino File Reference

4.2 examples/RolloverTest/RolloverTest.ino File Reference

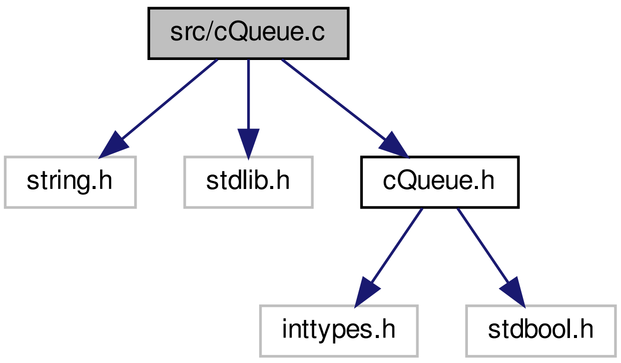
4.3 examples/SimpleQueue/SimpleQueue.ino File Reference

4.4 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 QUEUE_INITIALIZED 0x5AA5

    Queue initialized control value.
- #define INC_IDX(ctr, end, start)

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

    *Decrements buffer index **cnt** 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_clean (Queue_t ∗q)

    Clean 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.

4.4.1 Detailed Description

Queue handling library (designed in c on STM32)

Author

SMFSW

Copyright

BSD 3-Clause License (c) 2017, SMFSW

Queue handling library (designed in c on STM32)

4.4.2 Macro Definition Documentation

4.4.2.1 DEC_IDX

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

Value:

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

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

4.4.2.2 INC_IDX

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

Value:

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

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

4.4.2.3 QUEUE_INITIALIZED

```
#define QUEUE_INITIALIZED 0x5AA5
```

Queue initialized control value.

4.4.3 Function Documentation

4.4.3.1 q_clean()

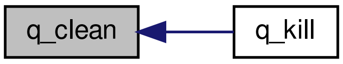
```
void q_clean (
            Queue_t * q )
```

Clean queue, restarting from empty queue.

Parameters

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

Here is the caller graph for this function:



4.4.3.2 q_drop()

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

Drop current record from queue.

Parameters

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

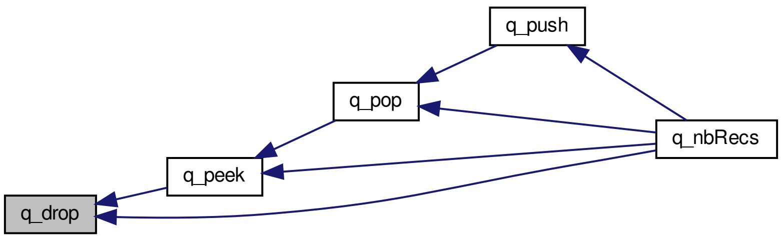
Returns

drop status

Return values

true	if succefully dropped from queue
false	if queue is empty

Here is the caller graph for this function:



4.4.3.3 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

4.4.3.4 q_kill()

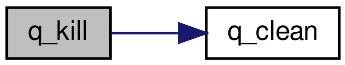
```
void q_kill (
            Queue_t * q )
```

Queue destructor: release dynamically allocated queue.

Parameters

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

Here is the call graph for this function:



4.4.3.5 q_peek()

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

Peek record from queue.

Parameters

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

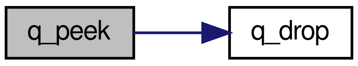
Returns

Peek status

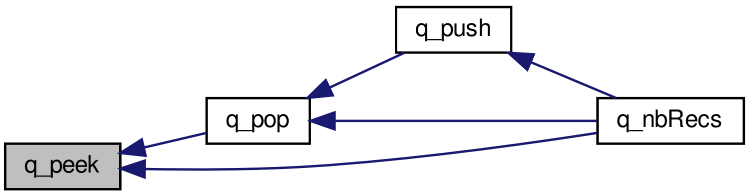
Return values

true	if succefully pulled from queue
false	if queue is empty

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.3.6 q_pop()

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

Pop record from queue.

Parameters

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

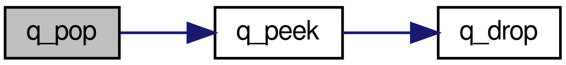
Returns

Pop status

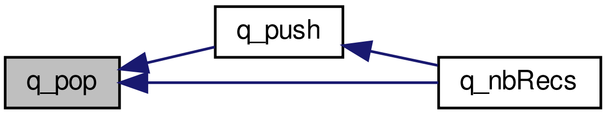
Return values

true	if succefully pulled from queue
false	if queue is empty

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.3.7 q_push()

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

Push record to queue.

Parameters

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

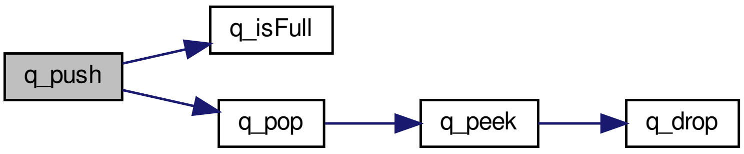
Returns

Push status

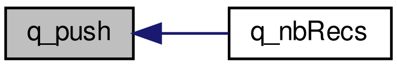
Return values

true	if succefully pushed into queue
false	if queue is full

Here is the call graph for this function:



Here is the caller graph for this function:

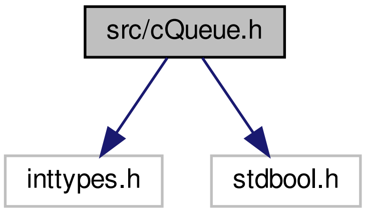


4.5 src/cQueue.h File Reference

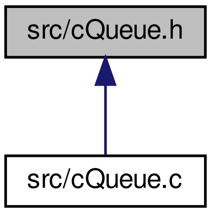
Queue handling library (designed in c on STM32)

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

Include dependency graph for cQueue.h:



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



Classes

- struct Queue_t

Macros

- #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

  As pull was already used in SMFSW libs, alias is made to keep compatibility with earlier versions.
- #define q_flush q_clean

  As flush is a common keyword, alias is made to empty queue.

Typedefs

- typedef enum enumQueueType QueueType
- typedef struct Queue_t Queue_t

Enumerations

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

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_clean (Queue_t ∗q)

  Clean queue, restarting from empty 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
- uint16_t q_nbRecs (const Queue_t ∗q)

  get number of records 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.

4.5.1 Detailed Description

Queue handling library (designed in c on STM32)

Author

SMFSW

Copyright

BSD 3-Clause License (c) 2017, SMFSW

Queue handling library (designed in c on STM32)

4.5.2 Macro Definition Documentation

4.5.2.1 q_flush

```
#define q_flush q_clean
```

As flush is a common keyword, alias is made to empty queue.

4.5.2.2 q_init_def

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

Some kind of average default for queue initialization.

4.5.2.3 q_pull

```
#define q_pull q_pop
```

As pull was already used in SMFSW libs, alias is made to keep compatibility with earlier versions.

4.5.3 Typedef Documentation

4.5.3.1 Queue_t

```
typedef struct Queue_t Queue_t
```

4.5.3.2 QueueType

```
typedef enum enumQueueType QueueType
```

4.5.4 Enumeration Type Documentation

4.5.4.1 enumQueueType

```
enum enumQueueType
```

Enumerator

FIFO	
LIFO	

4.5.5 Function Documentation

4.5.5.1 q_clean()

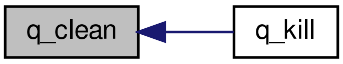
```
void q_clean (
            Queue_t * q )
```

Clean queue, restarting from empty queue.

Parameters

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

Here is the caller graph for this function:



4.5.5.2 q_drop()

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

Drop current record from queue.

Parameters

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

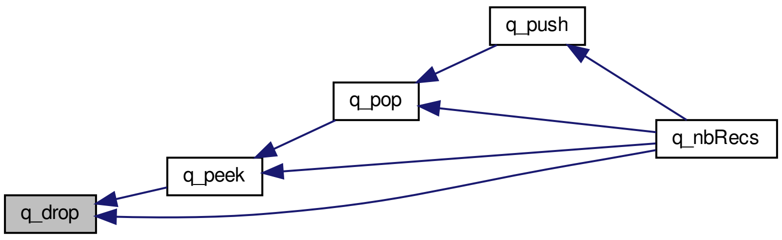
Returns

drop status

Return values

true	if succefully dropped from queue
false	if queue is empty

Here is the caller graph for this function:



4.5.5.3 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

4.5.5.4 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

4.5.5.5 q_isFull()

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

get fullness state of the queue

Parameters

in	*q*	- pointer of queue to handle

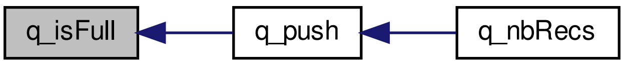
Returns

Queue fullness status

Return values

true	if queue is full
false	is not full

Here is the caller graph for this function:



4.5.5.6 q_kill()

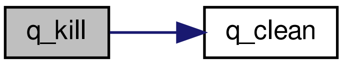
```
void q_kill (
             Queue_t * q )
```

Queue destructor: release dynamically allocated queue.

Parameters

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

Here is the call graph for this function:



4.5.5.7 q_nbRecs()

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

get number of records in the queue

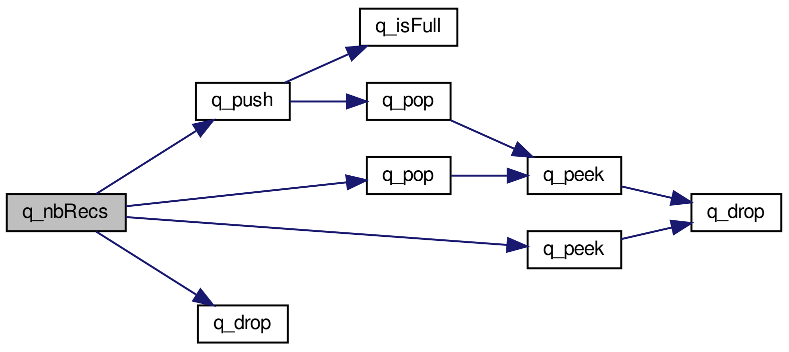
Parameters

in	*q*	- pointer of queue to handle

Returns

Number of records left in the queue

Here is the call graph for this function:



4.5.5.8 q_peek()

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

Peek record from queue.

Parameters

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

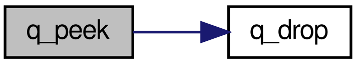
Returns

Peek status

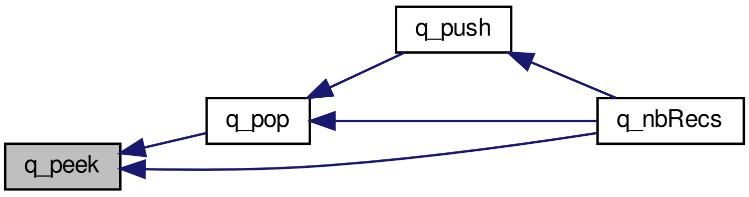
Return values

true	if succefully pulled from queue
false	if queue is empty

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.5.9 q_pop()

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

Pop record from queue.

Parameters

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

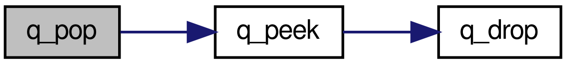
Returns

Pop status

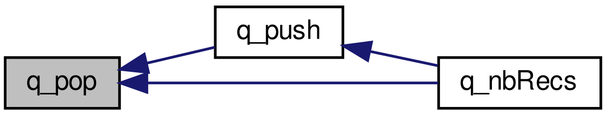
Return values

true	if succefully pulled from queue
false	if queue is empty

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.5.10 q_push()

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

Push record to queue.

Parameters

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

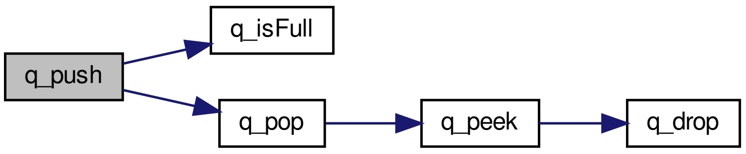
Returns

Push status

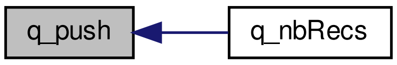
Return values

true	if succefully pushed into queue
false	if queue is full

Here is the call graph for this function:



Here is the caller graph for this function:



Index