

# cQueue

1.1

Generated by Doxygen 1.8.13

## Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>2</b>
2.1	File List . . . . .	2
<b>3</b>	<b>Class Documentation</b>	<b>2</b>
3.1	Queue_t Struct Reference . . . . .	2
3.1.1	Member Data Documentation . . . . .	3
<b>4</b>	<b>File Documentation</b>	<b>4</b>
4.1	examples/LibTst/LibTst.ino File Reference . . . . .	4
4.2	examples/RolloverTest/RolloverTest.ino File Reference . . . . .	4
4.3	examples/SimpleQueue/SimpleQueue.ino File Reference . . . . .	4
4.4	src/cQueue.c File Reference . . . . .	4
4.4.1	Detailed Description . . . . .	5
4.4.2	Macro Definition Documentation . . . . .	6
4.4.3	Function Documentation . . . . .	6
4.5	src/cQueue.h File Reference . . . . .	12
4.5.1	Detailed Description . . . . .	14
4.5.2	Macro Definition Documentation . . . . .	14
4.5.3	Typedef Documentation . . . . .	15
4.5.4	Enumeration Type Documentation . . . . .	15
4.5.5	Function Documentation . . . . .	15
	<b>Index</b>	<b>23</b>

## 1 Class Index

### 1.1 Class List

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

<a href="#">Queue_t</a>	2
-------------------------	---

## 2 File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">examples/LibTst/LibTst.ino</a>	4
<a href="#">examples/RolloverTest/RolloverTest.ino</a>	4
<a href="#">examples/SimpleQueue/SimpleQueue.ino</a>	4
<a href="#">src/cQueue.c</a> Queue handling library (designed in c on STM32)	4
<a href="#">src/cQueue.h</a> 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

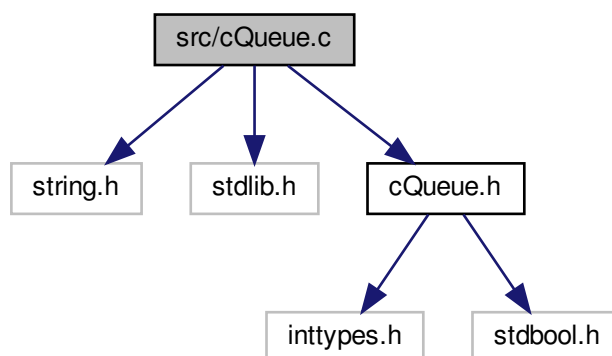
Queue handling library (designed in c on STM32)

```
#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, uint16\_t size\_rec, uint16\_t nb\_recs, `QueueType` type, 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, 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

#### Version

1.1

#### Date

2017/08/16

#### 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 **ctr** 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 **ctr** 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	<i>q</i>	- 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	<i>q</i>	- pointer of queue to handle
---------	----------	------------------------------

**Returns**

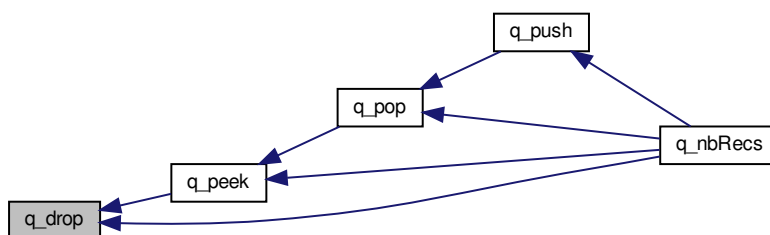
drop status

**Return values**

<i>true</i>	if succefully dropped from queue
<i>false</i>	if queue is empty



Here is the caller graph for this function:



#### 4.4.3.3 q\_init()

```

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

Queue initialization.

##### Parameters

in, out	<i>q</i>	- pointer of queue to handle
in	<i>size_rec</i>	- size of a record in the queue
in	<i>nb_recs</i>	- number of records in the queue
in	<i>type</i>	- Queue implementation type: FIFO, LIFO
in	<i>overwrite</i>	- 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	<i>q</i>	- 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	<i>q</i>	- pointer of queue to handle
in, out	<i>record</i>	- pointer to record to be peeked from queue

##### Returns

Peek status

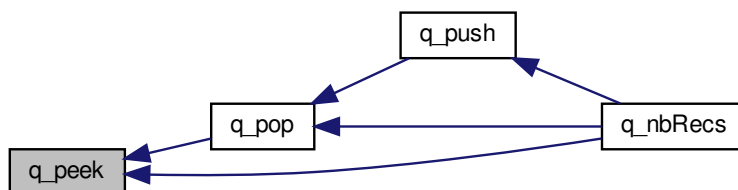
##### Return values

<i>true</i>	if successfully pulled from queue
<i>false</i>	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	<i>q</i>	- pointer of queue to handle
in, out	<i>record</i>	- pointer to record to be popped from queue

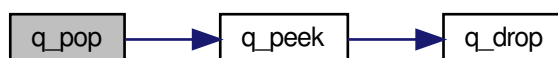
##### Returns

Pop status

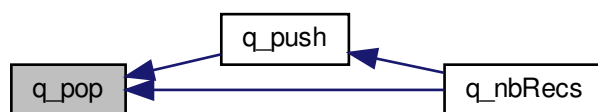
##### Return values

<i>true</i>	if succefully pulled from queue
<i>false</i>	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,
    void * record )
```

Push record to queue.

##### Parameters

in, out	<i>q</i>	- pointer of queue to handle
in	<i>record</i>	- pointer to record to be pushed into queue

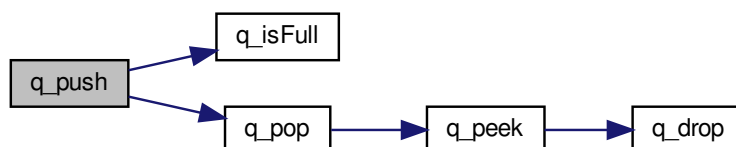
##### Returns

Push status

##### Return values

<i>true</i>	if succesfully pushed into queue
<i>false</i>	if queue is full

Here is the call graph for this function:



Here is the caller graph for this function:



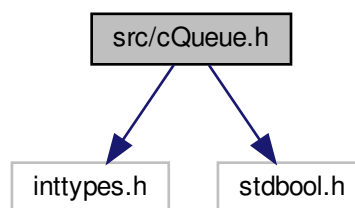
#### 4.5 src/cQueue.h File Reference

Queue handling library (designed in c on STM32)

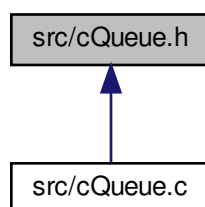
```
#include <inttypes.h>
```

```
#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, uint16_t size_rec, uint16_t nb_recs, QueueType type, 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 (Queue_t *q)`  
*get emptiness state of the queue*
- `bool q_isFull (Queue_t *q)`  
*get fullness state of the queue*
- `uint16_t q_nbRecs (Queue_t *q)`  
*get number of records in the queue*
- `bool q_push (Queue_t *q, 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

Version

1.1

Date

2017/08/16

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	<i>q</i>	- 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
---------	---	------------------------------

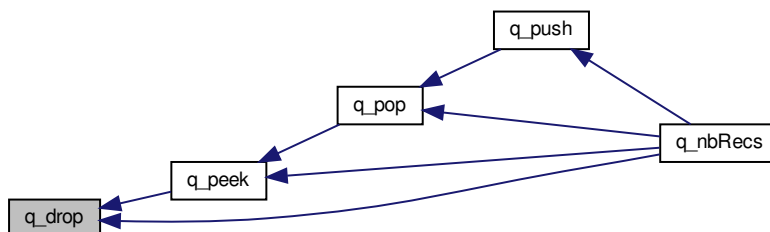
##### Returns

drop status

##### Return values

<i>true</i>	if succefully dropped from queue
<i>false</i>	if queue is empty

Here is the caller graph for this function:



## 4.5.5.3 q\_init()

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

Queue initialization.

## Parameters

in, out	<i>q</i>	- pointer of queue to handle
in	<i>size_rec</i>	- size of a record in the queue
in	<i>nb_recs</i>	- number of records in the queue
in	<i>type</i>	- Queue implementation type: FIFO, LIFO
in	<i>overwrite</i>	- 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 (
    Queue_t * q ) [inline]
```

get emptiness state of the queue

## Parameters

in	<i>q</i>	- pointer of queue to handle
----	----------	------------------------------

## Returns

Queue emptiness status

## Return values

<i>true</i>	if queue is empty
<i>false</i>	is not empty

## 4.5.5.5 q\_isFull()

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

get fullness state of the queue

#### Parameters

in	<i>q</i>	- pointer of queue to handle
----	----------	------------------------------

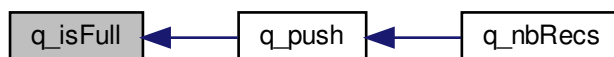
#### Returns

Queue fullness status

#### Return values

<i>true</i>	if queue is full
<i>false</i>	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	<i>q</i>	- pointer of queue to handle
---------	----------	------------------------------

Here is the call graph for this function:



## 4.5.5.7 q\_nbRecs()

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

get number of records in the queue

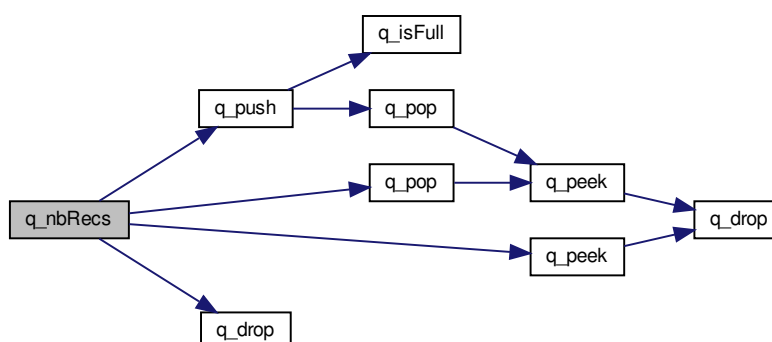
## Parameters

in	<i>q</i>	- 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	<i>q</i>	- pointer of queue to handle
in, out	<i>record</i>	- pointer to record to be peeked from queue

**Returns**

Peek status

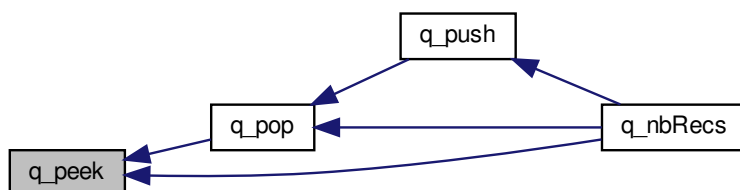
**Return values**

<i>true</i>	if succefully pulled from queue
<i>false</i>	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	<i>q</i>	- pointer of queue to handle
in, out	<i>record</i>	- pointer to record to be popped from queue

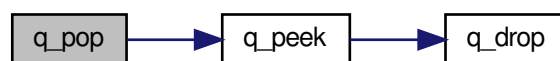
**Returns**

Pop status

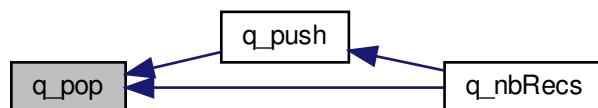
**Return values**

<i>true</i>	if successefully pulled from queue
<i>false</i>	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,  
    void * record )
```

Push record to queue.

**Parameters**

in, out	<i>q</i>	- pointer of queue to handle
in	<i>record</i>	- pointer to record to be pushed into queue

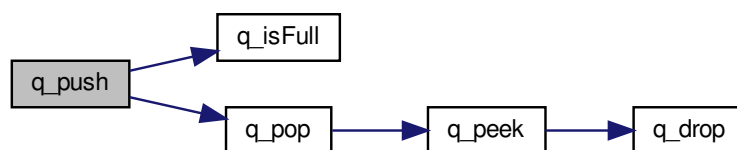
**Returns**

Push status

**Return values**

<i>true</i>	if succefully pushed into queue
<i>false</i>	if queue is full

Here is the call graph for this function:



Here is the caller graph for this function:



## Index

cQueue.c  
  DEC\_IDX, 6  
  INC\_IDX, 6  
  q\_clean, 6  
  q\_drop, 7  
  q\_init, 8  
  q\_kill, 8  
  q\_peek, 9  
  q\_pop, 10  
  q\_push, 11  
  QUEUE\_INITIALIZED, 6  
cQueue.h  
  enumQueueType, 15  
  q\_clean, 15  
  q\_drop, 16  
  q\_flush, 14  
  q\_init, 16  
  q\_init\_def, 14  
  q\_isEmpty, 17  
  q\_isFull, 17  
  q\_kill, 18  
  q\_nbRecs, 19  
  q\_peek, 19  
  q\_pop, 20  
  q\_pull, 14  
  q\_push, 21  
  Queue\_t, 15  
  QueueType, 15  
cnt  
  Queue\_t, 3  
DEC\_IDX  
  cQueue.c, 6  
enumQueueType  
  cQueue.h, 15  
examples/LibTst/LibTst.ino, 4  
examples/RolloverTest/RolloverTest.ino, 4  
examples/SimpleQueue/SimpleQueue.ino, 4  
INC\_IDX  
  cQueue.c, 6  
impl  
  Queue\_t, 3  
in  
  Queue\_t, 3  
init  
  Queue\_t, 3  
out  
  Queue\_t, 3  
ovw  
  Queue\_t, 3  
q\_clean  
  cQueue.c, 6  
  cQueue.h, 15  
q\_drop  
  cQueue.c, 7  
  cQueue.h, 16  
q\_flush  
  cQueue.h, 14  
q\_init  
  cQueue.c, 8  
  cQueue.h, 16  
q\_init\_def  
  cQueue.h, 14  
q\_isEmpty  
  cQueue.h, 17  
q\_isFull  
  cQueue.h, 17  
q\_kill  
  cQueue.c, 8  
  cQueue.h, 18  
q\_nbRecs  
  cQueue.h, 19  
q\_peek  
  cQueue.c, 9  
  cQueue.h, 19  
q\_pop  
  cQueue.c, 10  
  cQueue.h, 20  
q\_pull  
  cQueue.h, 14  
q\_push  
  cQueue.c, 11  
  cQueue.h, 21  
QUEUE\_INITIALIZED  
  cQueue.c, 6  
queue  
  Queue\_t, 3  
Queue\_t, 2  
  cQueue.h, 15  
  cnt, 3  
  impl, 3  
  in, 3  
  init, 3  
  out, 3  
  ovw, 3  
  queue, 3  
  rec\_nb, 4  
  rec\_sz, 4  
QueueType  
  cQueue.h, 15  
rec\_nb  
  Queue\_t, 4  
rec\_sz  
  Queue\_t, 4  
src/cQueue.c, 4  
src/cQueue.h, 12