



Q-Kernel
Reference Guide
Version 4.0-1775

Q-Kernel is a product of Quasarsoft Ltd.

Disclaimer

The information in this document is subject to change without notice. While the information herein is assumed to be accurate, Quasarsoft Ltd. (the manufacturer) assumes no responsibility for any errors or omissions.

The author makes and you receive no warranties or conditions, express, implied, statutory or in any communications with you. The manufacturer specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

In no event shall the manufacturer, its employees, its contractors, or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees, or expenses of any nature or kind.

Copyright notice

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the manufacturer. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license. If you have received this product under a Demo license for evaluation, you are entitled to evaluate it, but you may under no circumstances use it in a product. If you want to do so, you must obtain a fully licensed version from the manufacturer.

©Copyright 2007-2010 Quasarsoft Ltd

Trademarks

Names mentioned in this manual may be trademarks of their respective Companies. Brand and product names are trademarks or registered trademarks of their respective holders.



Quasarsoft Ltd
312 5th Ave Bay 14
Suite 354
Cochrane Alberta T4C 2E3
Canada
Tel. +1 (403) 450 3482
www.quasarsoft.com

About this document

This document describes the **Q-Kernel** (*The new generation RTOS*) Application Programming Interface (API) Most **Q-Kernel** documents, specially the user guides, are MCU specific but this manual is for all version of **Q-Kernel**. All versions have the same API.

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application, mainly C30 and the Linker
- The C Programming language
- The **Q-Kernel** user guide for your processor.

If you feel that your knowledge of C is not sufficient, we recommend *The C Programming Language* by Kernighan and Richie (ISBN 0-13-1103628), which describes the standard in C-programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

The intention of this manual is to give you a reference for all **Q-Kernel** API functions. For a more comprehensive description how to use **Q-Kernel** please read the **Q-Kernel** User guide.



Color ribbon

A color ribbon underneath every function description indicates when and from where the function can be called.

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

The example above describes that the function always be called except from an ISR. The function does not preempt.

Before Start

This indicator is be one of the following colors:

- **Red** indicates that it is not allowed to call the function before the start of the kernel with the function `qKrnStart()`.
- **Green** indicates that it is allowed to call the function before the start of the kernel with the function `qKrnStart()`. It is possible that the function real purpose is delayed until the start of the kernel.

Thread

This indicator is one of the following colors:

- **Red** indicates that it is not allowed to call the function from a thread.
- **Orange** indicates that the function can be called from a lightweight thread or Fiber but that the working depends on the input variables of the function.
- **Green** indicates that it is allowed to call the function from a thread.

Lightweight Thread

This indicator is one of the following colors:

- **Red** indicates that it is not allowed to call the function from a lightweight thread.
- **Orange** indicates that the function can be called from a lightweight thread but that the working depends on the input variables of the function. In most cases this is related to the timeout variable.
- **Green** indicates that it is allowed to call the function from a lightweight thread.

Fiber

This indicator is one of the following colors:

- **Red** indicates that it is not allowed to call the function from a fiber.
- **Orange** indicates that the function can be called from a fiber but that the working depends on the input variables of the function. In most cases this is related to the timeout variable.
- **Green** indicates that it is allowed to call the function from a fiber.

ISR

This indicator is one of the following colors:

- **Red** indicates that it is not allowed to call the function from an ISR.
- **Green** indicates that it is allowed to call the function from an ISR.

Preemption

This indicator is always yellow and the text can be the following:

- **No** Preemption means that the function never preempts if called from a thread or lightweight thread.
- **Always** Preemption means that the function always preempts if called from a thread or lightweight thread.
- **Possible** Preemption means that the function could preempts if called from a thread or lightweight thread.

Error Conditions

The error conditions are sometimes in blue and sometimes in white. See the example below. The system notifies the errors in blue only if the standard library is in use. See for more information the User Guide.

Error	Description
qERR_EVT_ISR	The function is called from an ISR
qERR_KRN_LICENSE	Creating this object violates the license criteria.

1.	qAddDays	12
2.	qAddHours	13
3.	qAddMinutes	14
4.	qAddSeconds	15
5.	qAddYears	16
6.	qCrtEnter	17
7.	qCrtExit	18
8.	qDateTime2DatTim	19
9.	qDatTim	20
10.	qDatTim2Date	21
11.	qDatTim2DateTime	22
12.	qDatTim2Time	23
13.	qDayOfWeek	24
14.	qDspEnter	25
15.	qDspExit	26
16.	qEdsAlloc (Only 16bit PIC's with EDS)	27
17.	qEdsAllocClr (Only 16bit PIC's with EDS)	28
18.	qEdsAllocFast (Only 16bit PIC's with EDS)	29
19.	qEdsAllocFastClr (Only 16bit PIC's with EDS)	30
20.	qEdsFree (Only 16bit PIC's with EDS)	31
21.	qEdsHeapAlloc (Only 16bit PIC's with EDS)	32
22.	qEdsHeapSize (Only 16bit PIC's with EDS)	33
23.	qEdsPool (Only 16bit PIC's with EDS)	34
24.	qEdsPoolAdd (Only 16bit PIC's with EDS)	35
25.	qEdsPoolNext (Only 16bit PIC's with EDS)	36
26.	qEdsPoolSize (Only 16bit PIC's with EDS)	37
27.	qEdsThrCreate (Only 16bit PIC's with EDS)	38
28.	qEdsThrCreateSuspended (PIC's with EDS)	40
29.	qEvtClear	42
30.	qEvtClose	43
31.	qEvtCreate	44
32.	qEvtOpen	45
33.	qEvtOpenNB	46
34.	qEvtOpenTO	47
35.	qEvtSignal	49
36.	qEvtSignalISR	50
37.	qEvtWait	51
38.	qEvtWaitNB	53
39.	qEvtWaitTO	55
40.	qFbrCreate	57
41.	qFbrEnqueueX (X = 0, 1 or 2)	58

42.	qFbrSpawnX (X = 1, 2, 3 or 4) (priority fibers)	59
43.	qFbrStatLapCycles	60
44.	qFbrStatLapPerc	61
45.	qFbrStatTotalCycles	62
46.	qFbrTrackPrioX (1 to 4)	63
47.	qFbrTrackQueued.....	64
48.	qFbrTrackScheduler	65
49.	qFixAlloc.....	66
50.	qFixAllocClr	67
51.	qFixCreate	68
52.	qFixClose	69
53.	qFixFree	70
54.	qHeapAlloc.....	71
55.	qHeapSize.....	72
56.	qKrnInit.....	73
57.	qKrnLicense	75
58.	qKrnError.....	76
59.	qKrnStack.....	77
60.	qKrnStart.....	78
61.	qKrnStatOff.....	79
62.	qKrnStatOn	80
63.	qKrnSwitchNotificationOff	81
64.	qKrnSwitchNotificationOn.....	82
65.	qKrnUSecOff	83
66.	qKrnUSecOn.....	84
67.	qKrnVersion	85
68.	qLwtCreate	86
69.	qLwtSleep.....	87
70.	qLwtYield.....	88
71.	qMemAlloc	89
72.	qMemAllocClr	90
73.	qMemAllocFast	91
74.	qMemAllocFastClr.....	92
75.	qMemFree.....	93
76.	qMemPool.....	94
77.	qMemPoolAdd.....	95
78.	qMemPoolNext	96
79.	qMemPoolSize	97
80.	qMemRealloc.....	98
81.	qMsgAlloc	99
82.	qMsgCopy	100

83.	qMsgFixAlloc	101
84.	qMsgFixCreate.....	102
85.	qMsgFree.....	103
86.	qMsgFreeISR.....	104
87.	qMsgMaxSize.....	105
88.	qMsgPublish	106
89.	qMsgPublishISR	107
90.	qMsgRead	108
91.	qMsgReadISR	109
92.	qMsgReceive	110
93.	qMsgReceiveNB	111
94.	qMsgReceiveTO	112
95.	qMsgSend	114
96.	qMsgSendNB	116
97.	qMsgSendTO	117
98.	qMsgWrite	119
99.	qMsgWriteISR	120
100.	qMtxClose.....	121
101.	qMtxCreate	122
102.	qMtxLock	124
103.	qMtxLockNB	125
104.	qMtxLockTO	126
105.	qMtxOpen	128
106.	qMtxOpenNB	129
107.	qMtxOpenTO	130
108.	qMtxUnlock	132
109.	qNtfError	133
110.	qNtfIdle	134
111.	qNtfSwitch	135
112.	qNtfStat.....	136
113.	qNtfPower.....	137
114.	qPipBlockSize	138
115.	qPipClose.....	139
116.	qPipCreate	140
117.	qPipEntries.....	143
118.	qPipGet	144
119.	qPipMaxBlocks.....	145
120.	qPipOpen	146
121.	qPipOpenNB	148
122.	qPipOpenTO	149
123.	qPipPut.....	151

124. qPipRead	152
125. qPipReadISR	153
126. qPipWrite	154
127. qPipWriteISR	155
128. qPubClose	156
129. qPubCreate	157
130. qPubOpen	158
131. qPubOpenNB	159
132. qPubOpenTO	160
133. qPubSubscribeFun	162
134. qPubSubscribePip	163
135. qPubSubscribeQue	164
136. qQueClose	165
137. qQueCreate	166
138. qQueOpen	167
139. qQueOpenNB	169
140. qQueOpenTO	170
141. qRtcAlarm	172
142. qRtcAlarm	173
143. qRtcGetDatTim	175
144. qRtcGetUptime	176
145. qRtcSetDatTim	177
146. qSemAcquire	178
147. qSemAcquireNB	179
148. qSemAcquireTO	180
149. qSemClose	182
150. qSemCreate	183
151. qSemOpen	185
152. qSemOpenNB	187
153. qSemOpenTO	188
154. qSemPermits	190
155. qSemRelease	191
156. qSemReleaseISR	192
157. qThrClose	193
158. qThrCreate	194
159. qThrCreateSuspended	196
160. qThrCurrent	198
161. qThrEvtClear	199
162. qThrEvtSignal	200
163. qThrEvtSignalISR	201
164. qThrEvtWait	202

165. qThrEvtWaitNB	204
166. qThrEvtWaitTO	206
167. qThrOpen	208
168. qThrOpenNB	210
169. qThrOpenTO.....	211
170. qThrResume.....	213
171. qThrResumeISR.....	214
172. qThrSetPriority	215
173. qThrSleep	216
174. qThrStack	218
175. qThrStatLapCycles	219
176. qThrStatLapPerc	220
177. qThrStatTotalCycles	221
178. qThrTrack	222
179. qThrSuspend.....	223
180. qThrYield	224
181. qTmrClose	225
182. qTmrCreate.....	227
183. qTmrOpen.....	229
184. qTmrOpenNB.....	231
185. qTmrOpenTO.....	232
186. qTmrStart.....	234
187. qTmrStartISR.....	235
188. qTmrStop	236
189. qTmrStopISR	238
190. qTmrUsec.....	239
191. qTmrUsecDiv1.....	240
192. qTmrUsecDiv256	241
193. qTmrUsecDiv64k	242
194. Errors.....	243
Event errors	243
Fiber errors	245
Kernel errors	246
Lightweight Thread errors	247
Message Errors	248
Mutex Errors	249
Pipe Errors	250
Queue Errors.....	251
Semaphore Errors.....	252
Thread Errors	253
Timer Errors.....	255

Real Time Clock Errors.....	257
Publish/subscribe Errors.....	258
Memory Errors.....	259

1. qAddDays

```

INT32U qAddDays (
    INT32U DatTim,    // The date-time to add to
    INTS Day) ;      // The number of days to add or
                    // subtract if negative

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function adds a number of days to the date-time and returns the new calculated date-time. The function does not test for over- or underflow.

Parameters and return value

Parameter	Description
Returns INT32U	The calculated new date
INT32U DatTim	The date-time in internal format.
INTS Day	The number of days to add.

Error conditions

None

2. qAddHours

```

INT32U qAddHours (
    INT32U DatTim,    // The date-time to add to
    INTS Hour);      // The number of hours to add or
                    // subtract if negative

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function adds a number of hours to the date-time and returns the new calculated date-time. The function does not test for over- or underflow.

Parameters and return value

Parameter	Description
Returns INT32U	The calculated new date
INT32U DatTim	The date-time in internal format.
INTS Hour	The number of hours to add.

Error conditions

None

3. qAddMinutes

```

INT32U qAddMinutes (
    INT32U DatTim,    // The date-time to add to
    INTS Minute);    // The number of minutes to add or
                    // subtract if negative

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function adds a number of minutes to the date-time and returns the new calculated date-time. The function does not test for over- or underflow.

Parameters and return value

Parameter	Description
Returns INT32U	The calculated new date
INT32U DatTim	The date-time in internal format.
INTS Minute	The number of minutes to add.

Error conditions

None

4. qAddSeconds

```

INT32U qAddSeconds (
    INT32U DatTim,    // The date-time to add to
    INTS Second);    // The number of seconds to add or
                    // subtract if negative

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function adds a number of seconds to the date-time and returns the new calculated date-time. The function does not test for over- or underflow.

Parameters and return value

Parameter	Description
Returns INT32U	The calculated new date
INT32U DatTim	The date-time in internal format.
INTS Second	The number of seconds to add.

Error conditions

None

5. qAddYears

```

INT32U qAddYears (
    INT32U DatTim,    // The date-time to add to
    INTS Year);      // The number of Years to add or
                    // subtract if negative

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function adds a number of years to the date-time and returns the new calculated date-time. The function does not test for over- or underflow.

Parameters and return value

Parameter	Description
Returns INT32U	The calculated new date
INT32U DatTim	The date-time in internal format.
INTS Year	The number of years to add.

Error conditions

None

6. qCrtEnter

```
void qCrtEnter(); // Enter a critical section
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function enters a critical section or if in a critical section it increases the count. Critical sections don't have a practical level limit. Every qCrtEnter() must be followed by an qCrtExit().

Parameters and return value

None

Error conditions

None

7. qCrtExit

```
void qCrtExit(); // Exit a critical section
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function exits a critical section. Every qCrtEnter() must be followed by an qCrtExit().

Parameters and return value

None

Error conditions

None

8. qDateTime2DatTim

```
INT32U qDateTime2DatTim(
    qtDATETIME DateTime) // The date-time to convert
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function converts a date-time in the structure to the internal data-time format. The function also checks if the date-time in the structure is valid. If not it will return zero.

Parameters and return value

Parameter	Description
Returns INT32U	The function returns the date-time in the internal format. If the date-time is incorrect the function returns zero.
qtDATETIME DateTime	A pointer to the date-time structure.

Error conditions

None

9. qDatTim

```

INT32U qDatTim(INT8U Year,    //
               INT8U Month,  //
               INT8U Hour,   //
               INT8U Minute, //
               INT8U Second); //

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns an internal DateTime value based on the input.

Parameters and return value

Parameter	Description
Returns INT32U	The function returns the date-time in the internal format. If the date-time is incorrect the function returns zero.
INT8U Year	The number of years.
INT8U Month	The month of the year from 1 to 12.
INT8U Day	The day of the month.
INT8U Hour	The hour of the day in 24 hour clock.
INT8U Minute	The number of minutes.
INT8U Second	The number of seconds.

Error conditions

None

10. qDatTim2Date

```
void qDatTim2Date(
    INT32U DatTim,           // The date-time to convert
    qtDATE Date);          // pointer to qTDate structure
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function converts date-time in internal format to a date structure.

Parameters and return value

Parameter	Description
INT32U DatTim	The date-time in internal format.
qtDATE Date	Pointer to the date structure for result

Error conditions

None

11. qDatTim2DateTime

```
void qDatTim2DateTime(
    INT32U DatTim,           // The date-time to convert
    qtDATETIME DateTime); // pointer to date-time structure
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function converts date-time in internal format to a date-time structure

Parameters and return value

Parameter	Description
INT32U DatTim	The date-time in internal format.
qtDATETIME DateTime	Pointer to the date-time structure for result

Error conditions

None

12. qDatTim2Time

```
void qDatTim2Time (
    INT32U DatTim,           // The date-time to convert
    qtTIME Time);          // pointer to time structure
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function converts date-time in internal format to a time structure.

Parameters and return value

Parameter	Description
INT32U DatTim	The date-time in internal format.
qtTIME Time	Pointer to the time structure for result

Error conditions

None

13. qDayOfWeek

```
INTU qDayOfWeek (
    INT32U DatTim)    // The date-time to find day of week
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function calculates the day of the week for a given date.

Parameters and return value

Parameter	Description
Returns INTU	The function returns day of the week for a given date. Monday=1, Tuesday=2 Sunday=7
INT32U DatTim	The date-time in internal format.

Error conditions

None

14. qDspEnter

```
void qDspEnter(); // Enter a DSP-Less section
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function must be used in native interrupts, when the DSP unit is used. Interrupts created with qISR() and qISR_FAST() don't require this function. qDspEnter() and qDspExit() must be used in pairs. The function can only be called from interrupts but that is not checked.

Parameters and return value

None

Error conditions

None

15. qDspExit

```
void qDspExit(); // Exit a DSP-Less section
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function exit a DSP-Less section. The function can only be called from interrupts but that is not checked.

Parameters and return value

None

Error conditions

None

16. qEdsAlloc (Only 16bit PIC's with EDS)

```
epVOID qEdsAlloc(           // Allocates memory from one of
    INTU Size);           // the EDS pools
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function allocates EDS memory and returns the pointer to the new allocated memory. If there is no memory available the function returns a null pointer.

The function first searches in the pool linked list if a pool of that size exists. If that's the case it allocates memory from that pool. If the pool is empty or a pool of that size does not exist it allocates it from the heap. It will always create a pool of the correct size.

The pointer is aligned on an integer boundary and the size is rounded up to a multiply of 8. The content on the memory is un-defined.

Parameters and return value

Parameter	Description
Returns epVOID	The function returns a pointer to the allocated EDS memory block or a null pointer when no memory is available.
INTU Size	The size of the memory block to allocate.

Error conditions

Error	Description
qERR_EDS_ISR	The function is called from an ISR.

17. qEdsAllocClr (Only 16bit PIC's with EDS)

```
epVOID qEdsAllocClr( // Allocates memory from the
    INTU Size);      // EDS memory pool and
                    // clears it.
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function allocates memory from the EDS memory pool, clears the memory (0x00) and returns the pointer to the new allocated memory. If there is no memory available the function returns a null pointer.

The function first searches in the pool linked list if a pool of that size exists. If that's the case it returns memory from that pool. If the pool is empty or a pool of that size does not exist it allocates it from the heap.

The pointer is aligned on an integer boundary and the size is rounded up to a multiply of 8. The content on the memory is zero.

Parameters and return value

Parameter	Description
Returns epVOID	The function returns an EDS pointer to the allocated memory block or a null pointer when no memory is available.
INTU Size	The size of the memory to allocate.

Error conditions

Error	Description
qERR_EDS_ISR	The function is called from an ISR.

18. qEdsAllocFast (Only 16bit PIC's with EDS)

```
epVOID qEdsAllocFast(    / Allocates memory from a
qtEPL pEpl);           // EDS memory pool
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function allocates memory from the EDS pool and returns a pointer to the new allocated memory. If there is no memory available the function returns a null pointer. If the pool is empty and it was not created the function allocates memory from the heap.

The content on the memory is un-defined.

Parameters and return value

Parameter	Description
Returns epVOID	The function returns a pointer to the allocated EDS memory block or a null pointer when no memory is available.
qtEPL pEpl	The address of a memory pool.

Error conditions

Error	Description
qERR_EDS_ISR	The function is called from an ISR.
qERR_EDS_ID	The object is not an EDS pool object, has not been created or points to no object at all.

19. qEdsAllocFastClr (Only 16bit PIC's with EDS)

```
epVOID qEdsAllocFastClr( // Allocates EDS memory from...
    qtEPL pEpl);         // ...a memory pool and...
                        // ...clears the memory.
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function allocates an EDS memory block from the pool, clears the memory (0x00) and returns a pointer to the new allocated memory. If there is no memory available the function returns a null pointer. If the pool is empty it allocates memory from the heap.

The content off the memory is zero.

Parameters and return value

Parameter	Description
Returns epVOID	The function returns a pointer to the allocated EDS memory block or a null pointer when no memory is available.
qtEPL pEpl	The memory pool to allocate from.

Error conditions

Error	Description
qERR_EDS_ISR	The function is called from an ISR.
qERR_EDS_ID	The object is not a EDS memory pool object, has not been created or points to no object at all.

20. qEdsFree (Only 16bit PIC's with EDS)

```
void qEdsFree (           // Frees EDS memory
  epVOID p) ;           // Pointer to the memory
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function frees EDS memory and returns the memory to one of the variable EDS memory pools.

Parameters and return value

Parameter	Description
epVOID	A pointer to the EDS memory. This must be the same pointer as returned from the qEdsAlloc() or qEdsAllocClr() functions.

Error conditions

Error	Description
qERR_EDS_ISR	The function is called from an ISR.
qERR_EDS_ID	The memory is not created as EDS memory or internal pointers are overwritten.

21. qEdsHeapAlloc (Only 16bit PIC's with EDS)

```
epVOID qEdsHeapAlloc( // Allocates heap EDS memory
    INTU Size);        // Size of the memory to
                       // allocate
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function allocates memory from the EDS heap and returns the pointer to the new allocated memory. If there is no memory available the function returns a null pointer.

The pointer is aligned on an integer boundary and the size is rounded to a multiply of the size of an integer. **The content off the memory is guaranteed zero.**

There is no EDS Heap free function because the heap does not support returning memory to the heap.

Parameters and return value

Parameter	Description
Returns epVOID	The function returns a pointer to the allocated EDS memory block or a null pointer when no memory is available.
INTU Size	The size of the memory to allocate

Error conditions

Error	Description
qERR_EDS_ISR	The function is called from an ISR.
qERR_EDS_DAMAGE	The internal memory structure has been damaged. This occurs when memory is allocated with a certain size and the application writes beyond the allocated size. This does not find all cases of damaged memory structures but will help the developer to find some of the issues.

22. qEdsHeapSize (Only 16bit PIC's with EDS)

```
INTU qEdsHeapSize(); // Returns free heap size
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function returns the free heap size.

Parameters and return value

Parameter	Description
Returns INTU	The function returns the free heap size.

Error conditions

Error	Description
qERR_EDS_ISR	The function is called from an ISR.

23. qEdsPool (Only 16bit PIC's with EDS)

```
qtEPL qEdsPool( // Returns the EDS pool with the
                // specified blocksize
                INTU Size); // Size of the blocks
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function returns a pointer to the EDS pool with the specified block size. If the pool does not exist it will create the pool otherwise it will simply return the existing pool.

Creating the pool will cost a small amount of memory which is allocated from the heap. The function will return a null pointer if the pool is not available and there is no memory available on the heap to create the pool.

Parameters and return value

Parameter	Description
Returns qtEPL	The function returns a pointer to the pool with the specified block size or a null pointer if a pool with that block size does not exist and there is no heap memory available to create the pool.
INTU Size	The size of the block. The size is rounded up to a multiply of 8.

Error conditions

Error	Description
qERR_EPL_ISR	The function is called from an ISR.

24. qEdsPoolAdd (Only 16bit PIC's with EDS)

```

INTU qEdsPoolAdd(           //
    qtEPL pEpl,           //
    INTU NbrBlocks);      //

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function adds a number of blocks to the EDS memory pool.

Parameters and return value

Parameter	Description
Returns INTU	The function returns the number memory blocks that where added.
qtEPL pEpl	The memory pool.
INTU NbrBlocks	The number of blocks that are to be added to the pool.

Error conditions

Error	Description
qERR_EPL_ISR	The function is called from an ISR.
qERR_EPL_ID	The object is not a memory pool object, has not been created or points to no object at all.

25. qEdsPoolNext (Only 16bit PIC's with EDS)

```
qtEPL qEplPoolNext( // Returns the next EDS
    qtEPL pEpl); // memory pool
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function returns a pointer to the next pool.

Parameters and return value

Parameter	Description
Returns qtEPL	The function returns the next EDS memory pool.
qtEPL pEpl	The memory pool

Error conditions

Error	Description
qERR_EDS_ISR	The function is called from an ISR.
qERR_EDS_ID	The object is not a memory pool object, has not been created or points to no object at all.

26. qEdsPoolSize (Only 16bit PIC's with EDS)

```
INTU qEdsPoolSize( //
qtEPL pEpl); //
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function returns the size of the pool.

Parameters and return value

Parameter	Description
Returns INTU	The function returns the size of the memory pool.
qtEPL pEpl	The memory pool

Error conditions

Error	Description
qERR_EPL_ISR	The function is called from an ISR.
qERR_EPL_ID	The object is not a memory pool object, has not been created or points to no object at all.

27. qEdsThrCreate (Only 16bit PIC's with EDS)

```
qtTCB qEdsThrCreate(
    char *pName,           // The Name of the thread
    void(*pFun)(void *pPar), // The thread function itself
    void *pPar,           // The parameter data
    INTU StackSize,       // The size of the stack
    INT8U Priority);      // The priority
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function will create a thread to be management by **Q-Kernel**. When the system is not yet started it will create the memory structures and other control information. When **Q-Kernel** is running it will do the same but it will make the thread ready to run and the thread will run if it becomes the highest priority thread. If other threads are waiting for this thread to be created those thread(s) are made run-able. The function executes the following steps:

- Allocates the TCB and the thread stack in EDS memory
- Populate the TCB
- Add the TCB to the ready list
- Reschedules if this thread becomes the highest priority thread
- Returns the pointer to the TCB

There is also a function available to create threads in a suspended mode. See `qEdtThrCreateSuspended()`.

Parameters

Parameter	Description
Returns qtTCB	The function returns a pointer to the Thread Control Block of the created thread.
char *pName	The name of the thread. The name must be unique within other thread objects or qNO_NAME which is a null pointer. qThrOpen() can be used to locate the Thread if the name is not a null pointer.
void(*pFun)(void *pPar)	The function that contains the code of the thread.
void *pPar	The parameter that is provided to the thread. This can be a pointer or any other type that fits in the pointer.
INTU StackSize	The size of the stack. This memory is allocated from variable memory.
INT8U Priority	The priority from 1 to 250. The higher the number the higher the priority.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_STACK	The stack size is smaller than 16 bytes.
qERR_THR_PRIO	The thread priority is 0 or greater than 250
qERR_THR_NAME_IN_USE	A thread with that name already exists
qERR_THR_MEMORY	There is no memory available to handle the request.
qERR_THR_NO_SHARED_STACK_SIZE	The stack is larger than the specified shared stack size.
qERR_THR_NO_EDS	There is no EDS memory available
qERR_KRN_LICENSE	Creating this object violates the license criteria.

The developer must give every thread a unique name or no name at all.

28. qEdsThrCreateSuspended (PIC's with EDS)

```

qtTCB qEdsThrCreate(
    char *pName,           // The Name of the thread
    void(*pFun)(void *pPar), // The thread function itself
    void *pPar,           // The parameter data
    INTU StackSize,       // The size of the stack
    INT8U Priority);      // The priority

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function will create a thread suspended to be management by *Q-Kernel*. When the system is not yet started it will create the memory structures and other control information. The function executes the following steps:

- Allocates the TCB and the thread stack in EDT memory
- Populate the TCB
- Add the TCB to the hibernate list
- Returns the pointer to the TCB

Parameters

Parameter	Description
Returns qtTCB	The function returns a pointer to the Thread Control Block of the created thread.
char *pName	The name of the thread. The name must be unique within other thread objects or qNO_NAME which is a null pointer. qThrOpen() can be used to locate the Thread if the name is not a null pointer.
void(*pFun)(void *pPar)	The function that contains the code of the thread.
void *pPar	The parameter that is provided to the thread. This can be a pointer or any other type that fits in the pointer.
INTU StackSize	The size of the stack. This memory is allocated from variable memory.
INT8U Priority	The priority from 1 to 250. The higher the number the higher the priority.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_STACK	The stack size is smaller than 16 bytes.
qERR_THR_PRIO	The thread priority is 0 or greater than 250
qERR_THR_NAME_IN_USE	A thread with that name already exists
qERR_THR_NO_SHARED_STACK_SIZE	The stack is larger than the specified shared stack size.
qERR_THR_NO_EDS	There is no EDS memory available
qERR_THR_MEMORY	There is no memory available to handle the request.
qERR_KRN_LICENSE	The thread that is created violates the license criteria.

The developer must give every thread a unique name or no name at all.

29. qEvtClear

```
INTU qEvtClear(
    qtEVT pEvt,           // The event set to clear
    INTU EventFlags);    // The flags to set
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

Clears one or more event flags in the event set and returns the events flags before the clear.

This is also the mechanism to get the event flags without changing the event flags. See the example below:

```
INTU flags;           // variable to return the flags
qtEVT *p;            // Set by create or open
flags = qEvtClear(p,0) // Null does not clear anything.
                    // It now just returns the event
                    // flags
```

Parameters and return value

Parameter	Description
Returns INTU	Returns the event flags before the clear operation.
qtEVT pEvt	A pointer to the event set object. Must be returned from qEvtCreate() or qEvtOpen() with the correct name.
INTU EventFlags	The event flags to clear. A value of zero does not clear any of the flags.

Error conditions

Error	Description
qERR_EVT_ISR	The function is called from an ISR.
qERR_EVT_ID	The object is not an event set object, has not been created or points to no object at all.

30. qEvtClose

```
void qEvtClose(
    qtEVT pEvt)           // The event set to close
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Closes an event set and returns the resources back to the resource pool. It is the developer responsibility to make sure that the event set is not used by other threads. The function will test if any thread is waiting on the event set but it does not detect if other threads are using this event set. The system will invalidate the object so other function can't use the object accidentally.

Parameters and return value

Parameter	Description
qtEVT pEvt	A pointer to the object. Must be returned from the qEvtCreate() or qEvtOpen() function with the correct name.

Error conditions

Error	Description
qERR_EVT_ISR	The function is called from an ISR.
qERR_EVT_ID	The object is not an event object, has not been created or points to no object at all.
qERR_EVT_IN_USE	One or more threads are waiting on the event. The system can't detect if other thread are using this event. The system will clear the object and use of the same address most likely results in qERR_EVT_ID

The developer is responsible for checking if the Event object is not used anymore.

31. qEvtCreate

```
qtEVT qEvtCreate (
    char *pName) ;           // The Name of the EventSet
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

Before an event set can be used, it has to be created by calling this function. On creation, all event flags are cleared. If there is an open request for the event with this name that thread will be readied, this creates a possible preemption. Multiple threads can wait for the object to be created and all threads will be readied.

The function needs memory for its operation, that's being freed when qEvtClose() end the use of the event. The function tries to allocate memory from the variable memory pool or from the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtEVT	The function returns a pointer to the event set object.
char *pName	The name of the event set. The name must be unique within other event set objects or qNO_NAME which is a null pointer. qEvtOpen() can be used to locate the object if the name is not a null pointer.

Error conditions

Error	Description
qERR_EVT_ISR	The function is called from an ISR
qERR_EVT_NAME_IN_USE	The name is already in use for another object
qERR_EVT_MEMORY	There is no memory available to handle the request.
qERR_KRN_LICENSE	Creating this object violates the license criteria.

The developer must give every object a unique name.

32. qEvtOpen

```
qtEVT qEvtOpen ( // Returns NULL when timed-out
    char *pName); // The Name of the EventSet
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function returns a pointer to an existing event set object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended.

Parameters and return value

Parameter	Description
Returns qtEVT	The function returns a pointer to the event object.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_EVT_ISR	The function is called from an ISR.
qERR_EVT_NO_START	The function is called before Q-Kernel is started.
qERR_EVT_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_EVT_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_EVT_NO_NAME	Events without a name can't be opened.
qERR_EVT_CRITICAL	This function cannot be called from within a critical section
qERR_EVT_MEMORY	There is no memory available to handle the open request.

33. qEvtOpenNB

```
qtEVT qEvtOpenNB(           //
    char *pName);           // The Name of the EventSet
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns a pointer to an existing event set object.

Parameters and return value

Parameter	Description
Returns qtEVT	The function returns a pointer to the event object. The function returns NULL if the object does not exist.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_EVT_ISR	The function is called from an ISR.
qERR_EVT_NO_NAME	Events without a name can't be opened.

34. qEvtOpenTO

```
qtEVT qEvtOpenTO(           // Returns NULL when timed-out
    char *pName,           // The Name of the object
    INT32S TimeOut);       // The timeout
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended. The timeout specifies how long the thread is willing to wait.

Parameters and return value

Parameter	Description
Returns qtEVT	The function returns a pointer to the event object. If the pointer is null the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_EVT_ISR	The function is called from an ISR.
qERR_EVT_NO_START	The function is called before Q-Kernel is started.
qERR_EVT_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_EVT_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_EVT_NO_NAME	Events without a name can't be opened.
qERR_EVT_CRITICAL	This function cannot be called from within a critical section
qERR_EVT_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.
qERR_EVT_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_EVT_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_EVT_MEMORY	There is no memory available to handle the open request.

35. qEvtSignal

```
void qEvtSignal(
    qtEVT pEvt,           // The event set to signal
    INTU EventFlags);    // The flags to set
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

Set one or more event flags in the event set. After the flags are set the threads that wait for events set are evaluated to see if they match the wait criteria. If any thread matches the criteria, then the one with highest priority is selected to run.

Threads that have set clear options will clear the flags but after all threads are checked. This is a significant difference with competing products because they clear flags during the process and that makes signaling unpredictable.

Parameters and return value

Parameter	Description
qtEVT pEvt	A pointer to the event set object. Must be returned from qEvtCreate() or qEvtOpen() with the correct name.
INTU EventFlags	The event flags to set. At least one flag should be set.

Error conditions

Error	Description
qERR_EVT_ID	The object is not an event set object, has not been created or points to no object at all.
qERR_EVT_ISR	The function is called from an ISR. Use qEvtSignalISR() for the same functionality available for ISR's.
qERR_EVT_NO_FLAGS	Not one flag in EventFlags is set

36. qEvtSignalISR

```
void qEvtSignal(
    qtEVT pEvt,           // The event set to signal
    INTU EventFlags);    // The flags to set
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Set one or more event flags in the event set. After the flags are set the threads that wait for events set are evaluated to see if they match the wait criteria. If any thread matches the criteria, then the one with highest priority is selected to run.

Threads that have set clear options will clear the flags but after all threads are checked. This is a significant difference with competing products because they clear flags during the process and that makes signaling unpredictable.

This function must be called from an ISR and the system will evaluate if it can execute the request immediately. If that's not possible it will spawn the request and will execute it if all interrupt requests are serviced.

Parameters and return value

Parameter	Description
qtEVT pEvt	A pointer to the event set object. Must be returned from qEvtCreate() or qEvtOpen() with the correct name.
INTU EventFlags	The event flags to set. At least one flag should be set.

Error conditions

Error	Description
qERR_EVT_ID	The object is not an event set object, has not been created or points to no object at all.
qERR_EVT_ISR	The function is not called from an ISR. Use qEvtSignal() for the same functionality available from fibers, threads and lightweight threads.
qERR_EVT_NO_FLAGS	Not one flag in EventFlags is set

37. qEvtWait

```

INTU qEvtWait(
    qtEVT pEvt,           // The event set to wait for
    INTU EventFlags,     // The flags to wait for
    qtWAIT WaitType);   // The type of wait (see below)

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

Wait for a specific set of event flags to be set. The required flags are specified in EventFlags. The function can wait for all specified flags set or any of the flags set. The developer can specify if the flags need to be cleared if the wait is over.

There is also a non-blocking version of this function. The non-blocking function is faster.

Parameters and return value

Parameter	Description
Returns INTU	Returns 0 if the function timed out. Any other value indicates success and returns the events flags that waked-up the thread before the optional clear.
qtEVT pEvt	A pointer to the event set object. Must be returned from qEvtCreate() or qEvtOpen() with the correct name.
INTU EventFlags	The event flags to wait for. At least one flag must be set.
qtWAIT WaitType	The type of wait. The following are defined. qWAIT_ALL means wait until all flags are set. This is also called the AND scenario. qWAIT_ALL_CLEAR means wait until all flags are set and if this situation occurs reset the flags that the thread was waiting for. qWAIT_ANY means wait until one of the flags is set. This is also called the OR scenario. qWAIT_ANY_CLEAR means wait until one of the flags is set and if this situation occurs reset the flags that triggered this operation. So not all flags that the thread was waiting for are reset.

Error conditions

Error	Description
qERR_EVT_ISR	The function is called from an ISR.
qERR_EVT_NO_START	The function is called before Q-Kernel is started.
qERR_EVT_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_EVT_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_EVT_ID	The object is not an event object, has not been created or points to no object at all.
qERR_EVT_NO_FLAGS	No flags in the EventFlags parameter is set
qERR_EVT_WAIT_TYPE	The event type is incorrect
qERR_EVT_CRITICAL	This function cannot be called from within a critical section.

38. qEvtWaitNB

```

INTU qEvtWait(
    qtEVT pEvt,           // The event set to wait for
    INTU EventFlags,     // The flags to wait for
    qtWAIT WaitType);    // The type of wait (see below)

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Wait for a specific set of event flags to be set. The required flags are specified in EventFlags. The function is non-blocking so it will not wait. The developer can specify if the flags need to be cleared if the function is successful.

Parameters and return value

Parameter	Description
Returns INTU	Returns 0 if the function is not successful. Any other value indicates success and returns the events flags that returned success before the optional clear.
qtEVT pEvt	A pointer to the event set object. Must be returned from qEvtCreate() or qEvtOpen() with the correct name.
INTU EventFlags	The event flags to wait for. At least one flag must be set.

Parameter	Description
qtWAIT WaitType	<p>This parameter is called the WaitType but it is more a comparer. The name is the same to be compatible with the other functions. The following are defined.</p> <p>qWAIT_ALL means wait until all flags are set. This is also called the AND scenario.</p> <p>qWAIT_ALL_CLEAR means wait until all flags are set and if this situation occurs reset the flags that the thread was waiting for.</p> <p>qWAIT_ANY means wait until one of the flags is set. This is also called the OR scenario.</p> <p>qWAIT_ANY_CLEAR means wait until one of the flags is set and if this situation occurs reset the flags that triggered this operation. So not all flags that the thread was waiting for are reset.</p>

Error conditions

Error	Description
qERR_EVT_ISR	The function is called from an ISR.
qERR_EVT_ID	The object is not an event object, has not been created or points to no object at all.
qERR_EVT_NO_FLAGS	No flags in the EventFlags parameter is set
qERR_EVT_WAIT_TYPE	The event type is incorrect

39. qEvtWaitTO

```

INTU qEvtWaitTO(
    qtEVT pEvt,           // The event set to wait for
    INTU EventFlags,     // The flags to wait for
    qtWAIT WaitType,     // The type of wait (see below)
    INT32S TimeOut);    // The maximum wait time

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible Preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

Wait for a specific set of event flags to be set. The required flags are specified in EventFlags. The function is non-blocking so it will not wait. The developer can specify if the flags need to be cleared if the function is successful.

This is the faster non-blocking version of qEvtWait(). Use qEvtWait() if the size of the code is a concern and qEvtWait() is already used in another place in the application because using both functions doubles the flash footprint.

Parameters and return value

Parameter	Description
Returns INTU	Returns 0 if the function is not successful. Any other value indicates success and returns the events flags that returned success before the optional clear.
qtEVT pEvt	A pointer to the event set object. Must be returned from qEvtCreate() or qEvtOpen() with the correct name.
INTU EventFlags	The event flags to wait for. At least one flag must be set.

Parameter	Description
qtWAIT WaitType	<p>The type of wait. The following are defined.</p> <p>qWAIT_ALL means wait until all flags are set. This is also called the AND scenario.</p> <p>qWAIT_ALL_CLEAR means wait until all flags are set and if this situation occurs reset the flags that the thread was waiting for.</p> <p>qWAIT_ANY means wait until one of the flags is set. This is also called the OR scenario.</p> <p>qWAIT_ANY_CLEAR means wait until one of the flags is set and if this situation occurs reset the flags that triggered this operation. So not all flags that the thread was waiting for are reset.</p>
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_EVT_ISR	The function is called from an ISR.
qERR_EVT_NO_START	The function is called before Q-Kernel is started.
qERR_EVT_ID	The object is not an event object, has not been created or points to no object at all.
qERR_EVT_NO_FLAGS	No flags in the EventFlags parameter is set
qERR_EVT_WAIT_TYPE	The event type is incorrect
qERR_EVT_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_EVT_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_EVT_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.

40. qFbrCreate

```
void qFbrCreate(
    INT8U Priority,           // The priority of the fiber
    void(*pFbr)(void));     // The fiber function itself
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function will create a priority fiber. The create does not execute the fiber but the qFbrSpawnX() will. Changing the priority fiber function is always possible.

Parameters

Parameter	Description
INT8U Priority	The priority of the fiber. Must be between 1 and 4 inclusive. 4 is the highest priority and 1 is the lowest.
void(*pFbr)(void)	The function that contains the code of the fiber.

Error conditions

Error	Description
qERR_FBR_PRIO	The priority is smaller than 1 or larger than 4.
qERR_FBR_ISR	The function is called from an ISR.

41. qFbrEnqueueX (X = 0, 1 or 2)

```

void qFbrEnqueue2 (           //
    void (*pFbr) (void*,void*), // The function to queue
    void *pPar1                // Par1 for the function
    void *pPar2);             // Par2 for the function

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function queues a fiber for processing.

The last character (X=0, 1 or 2) specifies how many arguments are used.

Parameters and return value

Parameter	Description
void (*pFbr)(void*,void*)	The function/fiber to queue. This example specifies 2 parameters. The function type for 0 parameters is void(*pFbr)(). The function type for 1 parameter is void(*pFbr)(void*).
void *pPar1	Parameter 1 for the function. Cast the variable if the data type is not a void pointer. The size of the data may not exceed the size of a pointer.
void *pPar	Parameter 2 for the function. Cast the variable if the data type is not a void pointer. The size of the data may not exceed the size of a pointer.

Error conditions

None. The user has to define a proper size of the fiber queue. If the size of the fiber queue is too small the function will loop.

42. qFbrSpawnX (X = 1, 2, 3 or 4) (priority fibers)

```
void qTskSpawn1 ();
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

There are four functions to execute priority fibers. They are numbered 1 to 4 and 4 has the highest priority. The function will execute immediately if called from a thread and will execute after the ISR ended if called from an ISR.

If the function is called before the kernel is started or before the priority fiber is created with qFbrCreate() the behavior is undefined.

Parameters and return value

None

Error conditions

None

43. qFbrStatLapCycles

```
INT32U qFbrStatLapCycles ();
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function returns the number of cycles for all fibers and the scheduler during the last lap.

Parameters

Parameter	Description
Returns INT32U	Returns the number of cycles

Error conditions

None

44. qFbrStatLapPerc

```
INTU qFbrStatLapPerc ();
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns the percentage of CPU time during the last "lap" for all fibers and scheduler.

Parameters

Parameter	Description
Returns INTU	The percentage is specified as a value * 100. So 12.34% is returned as 1234.

Error conditions

None

45. qFbrStatTotalCycles

```
INT64U qFbrStatTotalCycles();
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function returns the total number of cycles since the start of the statistic gathering for all fibers and scheduler.

Parameters

Parameter	Description
Returns INT64U	The number of cycles

Error conditions

None

46. qFbrTrackPrioX (1 to 4)

```
void qFbrTrackPrio1(
    INTU *pAddr,    // The address of the bit to set/clear
    INTU BitNbr);  // The bit number
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function enables or disables tracking for priority fibers. The address and the bit-number specify which bit to set or clear. If the address points to is one of the I/O ports the developer is responsible for setting the TRIS for that port.

Tracking need to be configured for this function to work.

Parameters

Parameter	Description
INTU *pAddr	The address of the bit to set or to clear. A value of 0 disables the tracking. If tracking is disabled it will execute the same number of cycles as enabled.
INTU BitNbr	The bit-number. Valid values are 0 to 15 for the 16 bit version and 0 to 31 for the 32 bit version.

Error conditions

Error	Description
qERR_FBR_ISR	The function is called from an ISR
qERR_FBR_BIT_NBR	Bit number is invalid

47. qFbrTrackQueued

```
void qFbrTrackQueued(
    INTU *pAddr,    // The address of the bit to set/clr
    INTU BitNbr);  // The bit number
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function enables or disables tracking for queued fibers. The address and the bit-number specify which bit to set or clear. If the address points to is one of the I/O ports the developer is responsible for setting the TRIS for that port.

Tracking need to be configured for this function to work.

Parameters

Parameter	Description
INTU *pAddr	The address of the bit to set or to clear. A value of 0 disables the tracking. If tracking is disabled it will execute the same number of cycles as enabled.
INTU BitNbr	The bit-number. Valid values are 0 to 15 for the 16 bit version and 0 to 31 for the 32 bit version.

Error conditions

Error	Description
qERR_FBR_ISR	The function is called from an ISR
qERR_FBR_BIT_NBR	Bit number is invalid

48. qFbrTrackScheduler

```
void qFbrTrackScheduler(
    INTU *pAddr,    // The address of the bit to set/clr
    INTU BitNbr);  // The bit number
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function enables or disables tracking for the scheduler. The address and the bit-number specify which bit to set or clear. If the address points to is one of the I/O ports the developer is responsible for setting the TRIS for that port.

Tracking need to be configured for this function to work.

Parameters

Parameter	Description
INTU *pAddr	The address of the bit to set or to clear. A value of 0 disables the tracking. If tracking is disabled it will execute the same number of cycles as enabled.
INTU BitNbr	The bit-number. Valid values are 0 to 15 for the 16 bit version and 0 to 31 for the 32 bit version.

Error conditions

Error	Description
qERR_FBR_ISR	The function is called from an ISR
qERR_FBR_BIT_NBR	Bit number is invalid

49. qFixAlloc

```
void *qFixAlloc(           // Allocates memory from one of
    qtFIX pFix);         // the fixed pools
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function allocates memory from one of the fixed memory pools and returns the pointer to the new allocated memory. If there is no memory available the function returns a null pointer. The memory is not initialized.

Parameters and return value

Parameter	Description
Returns *void	The function returns a pointer to the allocated memory block or a null pointer when no memory is available.
qtFIX pFix	The fixed memory pool that has been created with qFixCreate().

Error conditions

None

50. qFixAllocClr

```
void *qFixAllocClr( // Allocates memory from one of
qtFIX pFix);      // the fixed pools
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function allocates memory from one of the fixed memory pools and returns the pointer to the new allocated memory. If there is no memory available the function returns a null pointer. The memory is cleared.

Parameters and return value

Parameter	Description
Returns *void	The function returns a pointer to the allocated memory block or a null pointer when no memory is available.
qtFIX pFix	The fixed memory pool that has been created with qFixCreate().

Error conditions

None

51. qFixCreate

```

qtFIX qFixCreate(           // The function creates
    INTU Size,              // a fixed memory pool
    INTU NbrBlocks);       //

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function creates a fixed memory pool with a specified size and a specified number of blocks.

Parameters and return value

Parameter	Description
Returns qtFIX	A pointer to the new fixed pool. The function returns null if there is no memory available.
INTU Size	The size of the memory block to allocate.
INTU NbrBlocks	The number of blocks that are allocated in the pool. A value of zero is allowed but the function will not read anything.

Error conditions

Error	Description
qFIX_ISR	The function is called from an ISR

52. qFixClose

```
void qFixClose(           // The function closes a fixed
qtFIX pFix);           // memory pool
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function closes a fixed memory pool and returns the memory to the pool.

Parameters and return value

Parameter	Description
qtFIX pFIX	The pool to close

Error conditions

Error	Description
qFIX_ISR	The function is called from an ISR

53. qFixFree

```
void qFixFree (           // De-allocate memory
void *p);                //
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function de-allocates memory and returns the memory to the pool. The developer is responsible that this is a valid pointer and that the memory shouldn't be used anymore. This is not checked by the function.

Parameters and return value

Parameter	Description
void *p	The address of the memory to return.

Error conditions

None

54. qHeapAlloc

```
void *qHeapAlloc(           // Allocates heap memory
    INTU Size);           // Size of the memory to
                          // allocate
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function allocates memory from the heap and returns the pointer to the new allocated memory. If there is no memory available the function returns a null pointer.

The pointer is aligned on an integer boundary and the size is rounded to a multiply of the size of an integer. **The content off the memory is guaranteed zero.**

There is no qHeapFree() function because the heap does not support returning memory to the heap.

Parameters and return value

Parameter	Description
Returns *void	The function returns a pointer to the allocated memory block or a null pointer when no memory is available.
INTU Size	The size of the memory to allocate

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.
qERR_MEM_DAMAGE	The internal memory structure has been damaged. This occurs when memory is allocated with a certain size and the application writes beyond the allocated size. This does not find all cases of damaged memory structures but will help the developer to find some of the issues.

55. qHeapSize

```
void *qHeapSize(); // Returns free heap size
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function returns the free heap size.

Parameters and return value

Parameter	Description
Returns INTU	The function returns the free heap size.

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.

56. qKrnInit

```
qtERR qKrnInit();
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function initializes the kernel. In more detail the function executes the following steps:

- Clears all its memory variables that are used by the system with the exception of persistent memory used to store errors.
- Setup the memory system and checks if it has enough memory for basic functionality.
- Create frequently used memory pools.
- Checks the license and distribution.
- Creates memory for the interrupt stack.
- If thread and fiber tracking is enabled it will setup the tracking.
- If statistics is enabled it will setup statistics.
- The system will start a critical section to prevent thread activity before the start of the system.
- Creates the Idle thread. (Idle thread will not run because the system is in a critical section that prevents thread activation.)
- The system will check if it was restarted by qKrnError(). If that's the case it will return a pointer the persisted error structure. If that's not the case the system will clear the BOR and POR in the RCON and will return a null pointer.

The function must be called before any Q-Kernel function and can only be called once. See the user guide for more information.

After this function is called the user can create threads, fibers and other objects. The reference manual specifies which function can be used before the system is started.

The size of the stack between the initialization and start of **Q-Kernel** is limited to the size of the interrupt stack. If more stack space is required the developer can create a thread with the highest priority and a larger stack and do the work there or increase the size of the interrupt stack.

Parameters and return value

Parameter	Description
Returns qtERR	The function returns a pointer to the persistent error structure if the processor was restarted by qKrnError() or null if that was not the case.

Error conditions

Error	Description
qERR_KRN_MEM_SIZE	There is not enough memory available to initialize the system and the defined objects.
qERR_KRN_LICENSE	Something is wrong with the license. If a Demo license is in use please check if the minor version for the license is the same as the Q-Kernel version. If a commercial version is used please contact the Q-Kernel support department by Email at support@quasarsoft.com
qERR_KRN_INT_STACK	There is not enough memory to create the interrupt stack. Please check the size of the interrupt stack
qERR_KRN_INTERRUPT	The kernel interrupt is the same as the kernel timer interrupt.
qERR_THR_*	Thread errors during the creation of the Idle thread.

57. qKrnLicense

```
INTU qKrnLicense(); // Returns the license number
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function returns the license number. Every license key contains a license number that is unique for every customer and every installation.

Parameters and return value

Parameter	Description
Returns INTU	The function returns the license number.

Error conditions

None

58. qKrnError

```
void qKrnError(      // This function initiates a reset
    INTU Error);    // and stores the error information
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function stores all error information in a special memory location that will not be cleared during reset and will reset the processor. Information stored will be available after a new qKrnInit() function.

Parameters and return value

Parameter	Description
INTU Error	The error that will be signaled.

Error conditions

None

59. qKrnStack

```
INTU qKrnStack (); // Returns free bytes on stack
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function returns the least number of bytes that were not used by the interrupt stack since the start of the RTOS. The developer can use this number to define the size of the interrupt stack. It does **not** specify the current number of space on the stack.

The function is not deterministic and should not be used in production systems. This function is normally used in debug sessions.

Parameters and return value

Parameter	Description
Returns INTU	The worst case number of bytes that were free on the interrupt stack

Error conditions

Error	Description
qERR_KRN_ISR	The function is called from an ISR.

60. qKrnStart

```
void qKrnStart(); // The system never returns here.
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Always preemption
---------------------	---------------	---------------------------	--------------	------------	--------------------------

Description

This function will start **Q-Kernel** and will not return to the caller. The function will execute the following steps:

- Setup the scheduler interrupt (qINTERRUPT)
- Test if the initialization is done and if this function is never called before.
- Setup the kernel timer if specified in the configuration
- Setup the statistics if specified in the configuration
- Setup the RTCC if specified by the system
- End the critical section that was started in qKrnInit() and this will enable thread switching
- Start the thread with the highest priority.

The function must be called after qKrnInit() and after at least after one call to qThrCreate(). The function can only called once.

Error conditions

Error	Description
qERR_KRN_NO_INIT	The system did not initialize itself. The function qKrnInit() must be called before this function.
qERR_KRN_STARTED	The system was already started.

61. qKrnStatOff

```
void qKrnStatOff();
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function enables statistics. This function only works if statistics are configured.

Parameters and return value

None

Error conditions

Error	Description
qERR_KRN_ISR	The function is called from an ISR.

62. qKrnStatOn

```
void qKrnStatOn();
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function disables statistics. This function only works if statistics are configured.

Parameters and return value

None

Error conditions

Error	Description
qERR_KRN_ISR	The function is called from an ISR.

63. qKrnSwitchNotificationOff

```
void qKrnSwitchNotificationOff();
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function instruct the scheduler not to call the function qNftSwitch() when it executes a context switch. See also qKrnSwitchNotificationOn().

Parameters

The function does not accept parameters.

Error conditions

The function does not return any error but running the function from within an ISR can create unpredictable results.

64. qKrnSwitchNotificationOn

```
void qKrnSwitchNotificationOn();
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function instruct the scheduler to call the function qNftSwitch() every time it executes a context switch. The function qNftSwitch() is defined in **Q-Kernel** as weak and just contains a return. The developer must define its own function to use this functionality. The application must be linked with a library that contains switch notification. The "Switch Notification" functionality is a heavy load on the system and Quasarsoft advises to use this functionality only if it is absolute required. See for more information the User guide.

Parameters

The function does not accept parameters.

Error conditions

The function does not return any error but running the function from within an ISR can create unpredictable results.

65. qKrnUsecOff

```
void qKrnUsecOff ();
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function disables μ Second gathering.

Parameters and return value

None

Error conditions

Error	Description
qERR_KRN_ISR	The function is called from an ISR.

66. qKrnUSecOn

```
void qKrnStatOn();
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function enables μ Second gathering.

Parameters and return value

None

Error conditions

Error	Description
qERR_KRN_ISR	The function is called from an ISR.
qERR_KRN_MHZ	The clock frequency is not a multiply of 1MHz.

67. qKrnVersion

```
qtVER qKrnVersion(); // Returns the version
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The version is described as major.minor-build So V2.2-1123 means it is major version 2, minor version 2 and build 1123. Quasarsoft Ltd uses a source control system for the in-house development and it will increase the build number after every build including documentation changes even if the build is never distributed.

A minor version increase means in most cases functional improvements.

Support requests should always contain version and build information.

The function returns the version in the structure. The structure is defined in qKernel.h and is listed below:

```
struct qsVER{
    INT16U build;
    INT8U minor;
    INT8U major;
};
typedef const struct qsVER * qtVER;
```

The following example test the minor version in an "if" statement:

```
if (qKrnVersion()->minor > 0x01) { // Test version
...     // Do something if version > 1
...
}
```

Parameters and return value

Parameter	Description
Returns qtVER	The function returns a pointer to the version structure.

Error conditions

None

68. qLwtCreate

```
void qLwtCreate(
    INT8U Priority,           // The LWT priority
    void(*pLwt)(void *p),   // The function itself
    void *pPar);            // The pointer to structure
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function will create a lightweight thread. When the system is not yet started it will create the memory structures and other control information. When **Q-Kernel** is running it will do the same and it will execute the lightweight thread when all threads are in wait mode.

Parameters

Parameter	Description
INT8U Priority	The priority of the lightweight thread. Must be a value between 1 and 250 inclusive. Higher number means higher priority
void(*pLwt)(void *p)	The function that contains the code of the fiber.
void *pPar	Data that will be available in the structure (Param) that the lightweight thread can use after it has been started.

Error conditions

Error	Description
qERR_LWT_ISR	The function is called from an ISR.
qERR_LWT_PRIORITY	The priority is not between 1 and 250.
qERR_LWT_MEMORY	There is no memory available to handle the request.

69. qLwtSleep

```
void qLwtSleep(
    INT32S TimeOut);           // The time to sleep
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Always preemption
---------------------	---------------	---------------------------	--------------	------------	--------------------------

Description

This function lets the current lightweight thread sleep. After the time has elapsed the lightweight thread is scheduled again.

Parameters and return value

Parameter	Description
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_LWT_NO_LWT	The function is called outside a lightweight thread.
qERR_LWT_TIME	The thread sleep time is incorrect.

70. qLwtYield

```
void qLwtYield();
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Always preemption
--------------	--------	--------------------	-------	-----	-------------------

Description

This function yields a lightweight thread. The system will activate the lightweight thread with the highest priority that is not in sleep mode. If this lightweight thread becomes the lightweight thread with the highest priority it is started again.

Parameters and return value

This function does not require any parameters or return values.

Error conditions

Error	Description
qERR_LWT_NO_LWT	The function is called outside a lightweight thread.

71. qMemAlloc

```
void *qMemAlloc(           // Allocates memory from one of
    INTU Size);           // the pools
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function allocates memory and returns the pointer to the new allocated memory. If there is no memory available the function returns a null pointer.

The function first searches in the pool linked list if a pool of that size exists. If that's the case it allocates memory from that pool. If the pool is empty or a pool of that size does not exist it allocates it from the heap. It will always create a pool of the correct size.

The pointer is aligned on an integer boundary and the size is rounded up to a multiply of 8. The content on the memory is un-defined.

Parameters and return value

Parameter	Description
Returns *void	The function returns a pointer to the allocated memory block or a null pointer when no memory is available.
INTU Size	The size of the memory block to allocate.

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.

72. qMemAllocClr

```
void *qMemAllocClr( // Allocates memory from the
    INTU Size);     // variable memory pool and
                   // clears it.
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function allocates memory from the variable memory pool, clears the memory (0x00) and returns the pointer to the new allocated memory. If there is no memory available the function returns a null pointer.

The function first searches in the pool linked list if a pool of that size exists. If that's the case it returns memory from that pool. If the pool is empty or a pool of that size does not exist it allocates it from the heap.

The pointer is aligned on an integer boundary and the size is rounded up to a multiply of 8. The content on the memory is zero.

Parameters and return value

Parameter	Description
Returns *void	The function returns a pointer to the allocated memory block or a null pointer when no memory is available.
INTU Size	The size of the memory to allocate.

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.

73. qMemAllocFast

```
void *qMemAllocFast( // Allocates memory from a
    qtMPL pMpl);    // memory pool
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function allocates memory from the pool and returns a pointer to the new allocated memory. If there is no memory available the function returns a null pointer. If the pool is empty and it was not created the function allocates memory from the heap.

The content on the memory is un-defined.

Parameters and return value

Parameter	Description
Returns *void	The function returns a pointer to the allocated memory block or a null pointer when no memory is available.
qtMPL pMpl	The address of a memory pool.

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.
qERR_MEM_ID	The object is not a memory pool object, has not been created or points to no object at all.

74. qMemAllocFastClr

```
void *qMemFixAllocClr( // Allocates memory from...
    qtMPL pMpl);      // ...a memory pool and...
                      // ...clears the memory.
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function allocates a memory block from the pool, clears the memory (0x00) and returns a pointer to the new allocated memory. If there is no memory available the function returns a null pointer. If the pool is empty it allocates memory from the heap.

The content off the memory is zero.

Parameters and return value

Parameter	Description
Returns *void	The function returns a pointer to the allocated memory block or a null pointer when no memory is available.
qtMPL pMpl	The memory pool to allocate from.

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.
qERR_MEM_ID	The object is not a memory pool object, has not been created or points to no object at all.

75. qMemFree

```
void qMemFree (           // Frees variable memory
void *p);                // Pointer to the memory
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function frees memory and returns the memory to one of the variable pools.

Parameters and return value

Parameter	Description
void *p	A pointer to the memory. This must be the same pointer as returned from the qMemAlloc() or qMemAllocClr() functions.

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.
qERR_MEM_ID	The memory is not created as variable memory or internal pointers are overwritten.

76. qMemPool

```

qtMPL qMemPool ( // Returns the pool with the
                  // specified blocksize
    INTU Size); // Size of the blocks

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function returns a pointer to the pool with the specified block size. If the pool does not exist it will create the pool otherwise it will simply return the existing pool.

Creating the pool will cost a small amount of memory which is allocated from the heap. The function will return a null pointer if the pool is not available and there is no memory available on the heap to create the pool.

Parameters and return value

Parameter	Description
Returns qtMPL	The function returns a pointer to the pool with the specified block size or a null pointer if a pool with that block size does not exist and there is no heap memory available to create the pool.
INTU Size	The size of the block. The size is rounded up to a multiply of 8.

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.

77. qMemPoolAdd

```
INTU qMemPoolAdd(           //
    qtMPL pMpl,           //
    INTU NbrBlocks);      //
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function adds a number of blocks to the memory pool.

Parameters and return value

Parameter	Description
Returns INTU	The function returns the number memory blocks that where added.
qtMPL pMpl	The memory pool.
INTU NbrBlocks	The number of blocks that are to be added to the pool.

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.
qERR_MEM_ID	The object is not a memory pool object, has not been created or points to no object at all.

78. qMemPoolNext

```
qtMPL qMemPoolNext( // Returns the next memory
qtMPL pMpl); // pool
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function returns a pointer to the next pool.

Parameters and return value

Parameter	Description
Returns qtMPL	The function returns the next memory pool.
qtMPL pMpl	The memory pool

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.
qERR_MEM_ID	The object is not a memory pool object, has not been created or points to no object at all.

79. qMemPoolSize

```
INTU qMemPoolSize ( //
qtMPL pMpl); //
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function returns the size of the pool.

Parameters and return value

Parameter	Description
Returns INTU	The function returns the size of the memory pool.
qtMPL pMpl	The memory pool

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.
qERR_MEM_ID	The object is not a memory pool object, has not been created or points to no object at all.

80. qMemRealloc

```
void *qMemRealloc( // Reallocates memory
void *p, // memory to re-allocate
INTU Size); // the new size
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function re-allocates memory and returns the pointer to the new allocated memory. If there is no memory available the function returns a null pointer.

The function checks if the current memory fits the new size. If that is the case it just returns the pointer to the same memory block.

If the current memory does not fit the new size, new memory is allocated with the qMemAlloc() function. If memory of that size can't be allocated the function returns a null pointer. In that case the original memory block is still intact. If the memory is available the information in the original memory is copied into the new block and the existing memory block is returned to the pool. The function will return the new pointer.

If the p argument is NULL the function acts like qMemAlloc, allocating a block of memory and returning a pointer to it.

The pointer is aligned on an integer boundary and the size is rounded up to a multiple of 8. The content on the memory is un-defined.

Parameters and return value

Parameter	Description
Returns *void	The function returns a pointer to the allocated memory block or a null pointer when no memory is available.
void *p	Pointer to an existing memory block
INTU Size	The size of the memory block to allocate.

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.
qERR_MEM_ID	The memory has not been allocated as variable memory or internal pointers are overwritten.

81. qMsgAlloc

```
qtMSG qMsgAlloc( // Allocates a message
                INTU Size); // with the specified size
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Allocates a message and returns a pointer to the message. The use count is set to one. The function uses memory from the variable memory pool.

Parameters and return value

Parameter	Description
Returns qtMSG	A pointer to the message. The function returns a null pointer if there is no memory available.
INTU Size	The maximum size of the message. The Size must be between 2 and 65000 inclusive.

Error conditions

Error	Description
qERR_MSG_ISR	The function is called from an ISR.
qERR_MSG_SIZE	The size is incorrect.

82. qMsgCopy

```
qtMSG qMsgCopy ( // Allocates a message and copies
qtMSG pMsg) ; // to context of this message
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Allocates a message, copies the context of the message into the new message and returns a pointer to the message. The use count is set to one.

Parameters and return value

Parameter	Description
Returns qtMSG	A pointer to the message. The function returns a null pointer if there is no memory available.
INTU qtMSG	A pointer to the message to copy.

Error conditions

Error	Description
qERR_MSG_ISR	The function is called from an ISR but the message is variable memory and not fixed memory.
qERR_MSG_ID	The message is not a message returned by qMsgAlloc() or qMsgCopy().

83. qMsgFixAlloc

```
qtMSG qMsgFixAlloc( // Allocates a message
qtFIX pFix);      // From fixed memory
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Allocates a message and returns a pointer to the message. The use count is set to one. The function uses fixed memory blocks to allocate the memory so it can be called from an interrupt.

Parameters and return value

Parameter	Description
Returns qtMSG	A pointer to the message. The function returns a null pointer if there is no memory available.
qtFIX pFix	A pointer to a fixed memory block returned from qMsgFixCreate()

Error conditions

No errors are thrown

84. qMsgFixCreate

```
qtFIX qMsgFixCreate( // Create a fixed message pool
    INT16U MsgSize, // Size of the message
    INT16U NbrMessages); // Number of messages to hold
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

Allocates a message from the fixed message pool and returns a pointer to the fixed memory pool. It returns a null pointer if there is no memory available.

Parameters and return value

Parameter	Description
Returns qtFIX	A pointer to the fixed memory pool. The function returns a null pointer if there is no memory available.
INT16U MsgSize	The size of the message.
INT16U NbrMessages	The maximum number of messages the pool can hold.

Error conditions

Error	Description
qFIX_ISR	The function is called from an ISR

85. qMsgFree

```
void qMsgFree(           // Free a message
  qtMSG pMsg);         // This message
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Decrements the use-count of a message and if the use-count reaches 0 the message memory is de-allocated.

Parameters and return value

Parameter	Description
qtMSG pMsg	A pointer to the message.

Error conditions

Error	Description
qERR_MSG_ISR	The function is called from an ISR.
qERR_MSG_ID	The message is not allocated by qMsgAlloc().

86. qMsgFreeISR

```
void qMsgFreeISR(      // Free a message
qtMSG pMsg);         // This message
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Decrements the use-count of a message and if the use-count reaches 0 the message memory is de-allocated.

Parameters and return value

Parameter	Description
qtMSG pMsg	A pointer to the message.

Error conditions

Error	Description
qERR_MSG_ISR	The function is not called from an ISR.
qERR_MSG_ID	The message is not allocated by qMsgAlloc().

87. qMsgMaxSize

```
INT16U qMsgMaxSize(           // returns the max size of
qtMSG p);                   // the message
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function returns the maximum message size

Parameters and return value

Parameter	Description
INT16U	Returns the maximum message size
qtMSG p	The message pool

Error conditions

Error	Description
qERR_MSG_ID	This is not a valid message pool

88. qMsgPublish

```
void qMsgPublish(
    qtPUB pPub,           // The publish object
    qtMSG pMsg);        // The message to send
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function publishes a message to the subscribers.

The message system will increase the use-count of the message by one for every subscriber. This makes development simple because the thread does not have to keep track how the message is used by other threads or fibers.

Parameters and return value

Parameter	Description
qtQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.
qtMSG pMsg	The message to be published to all subscribers.

Error conditions

Error	Description
qERR_QUE_ISR	The function is called from an ISR.
qERR_QUE_ID	The object is not a queue object, has not been created or points to no object at all.
qERR_MSG_ID	The message is not allocated by qMsgAlloc().

89. qMsgPublishISR

```
void qMsgPublishISR(
    qtPUB pPub,           // The publish object
    qtMSG pMsg);         // The message to send
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function publishes a message to the subscribers.

The message system will increase the use-count of the message by one for every subscriber. This makes development simple because the thread does not have to keep track how the message is used by other threads or fibers.

Parameters and return value

Parameter	Description
qtQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.
qtMSG pMsg	The message to be published to all subscribers.

Error conditions

Error	Description
qERR_QUE_ISR	The function is not called from an ISR.
qERR_QUE_ID	The object is not a queue object, has not been created or points to no object at all.
qERR_MSG_ID	The message is not allocated by qMsgAlloc().

90. qMsgRead

```
qtMSG qMsgRead( // returns pointer to message
qtPIP pPip); // The pipe to read from
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function reads a message from a pipe. The function returns NULL if there is no information in the pipe.

The system will notify the writer that it has read the pipe.

Parameters and return value

Parameter	Description
Returns qtMSG	The message read from the pipe or NULL
qtPIP pPip	The pipe object to read from.

Error conditions

Error	Description
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.
qERR_PIP_ISR	The function is called from an interrupt handler.

91. qMsgReadISR

```
qtMSG qMsgRead( // returns pointer to message
qtPIP pPip); // The pipe to read from
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function reads a message from a pipe. The function returns NULL if there is no information in the pipe.

The system will notify the writer that it has read the pipe.

Parameters and return value

Parameter	Description
Returns qtMSG	The message read from the pipe or NULL
qtPIP pPip	The pipe object to read from.

Error conditions

Error	Description
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.
qERR_PIP_ISR	The function is not called from an interrupt handler.

92. qMsgReceive

```
qtMSG qMsgReceive ( // Returns a message
qtQUE pQue); // The message queue to read
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

The function will return immediately if there is a message available or wait for a message to become available. Multiple threads can wait for a message but only the thread with the highest priority will receive the message. Other waiting threads will not receive this message.

Parameters and return value

Parameter	Description
Returns qtMSG	Returns a pointer to the message. If a time-out occurs the function will return a null pointer.
qtQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.

Error conditions

Error	Description
qERR_QUE_ISR	The function is called from an ISR.
qERR_QUE_NO_START	The function is called before the kernel is started.
qERR_QUE_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_QUE_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_QUE_ID	The object is not a queue object, has not been created or points to no object at all.
qERR_QUE_CRITICAL	The function is called within a critical section.

93. qMsgReceiveNB

```
qtMSG qMsgReceiveNB (
    qtQUE pQue);           // The message queue to read
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function will return immediately with a message if there is one available or returns a NULL pointer.

Parameters and return value

Parameter	Description
Returns qtMSG	Returns a pointer to the message. The function will return a null pointer if no message is available.
qtQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.

Error conditions

Error	Description
qERR_QUE_ISR	The function is called from an ISR.
qERR_QUE_NO_START	The function is called before the kernel is started.
qERR_QUE_ID	The object is not a queue object, has not been created or points to no object at all.

94. qMsgReceiveTO

```

qtMSG qMsgReceiveTO( // Returns a message
qtQUE pQue, // The message queue to read
INT32S TimeOut); // The Timeout

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

The function will return immediately if there is a message available or wait for a message to become available. The TimeOut specifies how long the thread is willing to wait. Multiple threads can wait for a message but only the thread with the highest priority will receive the message. Other waiting threads will not receive this message.

Parameters and return value

Parameter	Description
Returns qtMSG	Returns a pointer to the message. If a time-out occurs the function will return a null pointer.
qtQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_QUE_ISR	The function is called from an ISR.
qERR_QUE_NO_START	The function is called before the kernel is started.
qERR_QUE_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_QUE_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_QUE_ID	The object is not a queue object, has not been created or points to no object at all.
qERR_QUE_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_QUE_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_QUE_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.
qERR_QUE_CRITICAL	This function cannot be called from within a critical section.

95. qMsgSend

```
INTU qMsgSend (
    qtQUE pQue,           // The queue to send to
    qtMSG pMsg);        // The message to send
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function sends a message to a queue and if there is no space it will wait until there is space available. If there is a receiving thread waiting it will immediately deliver the message to the waiting thread without placing the message in the queue. A receiving thread will be made ready to run.

The message system will increase the use-count of the message by one so the sender can free the message immediately. This makes development simple because the thread does not have to keep track how the message is used by other threads or fibers.

Parameters and return value

Parameter	Description
Returns INTU	A zero value indicates that the function timed-out. A value of 1 indicates success.
qtQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.
qtMSG pMsg	The message to send to the queue or a waiting thread.

Error conditions

Error	Description
qERR_QUE_ISR	The function is called from an ISR.
qERR_QUE_NO_START	The function is called before Q-Kernel is started.
qERR_QUE_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_QUE_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_QUE_ID	The object is not a queue object, has not been created or points to no object at all.
qERR_MSG_ID	The message is not allocated by qMsgAlloc().
qERR_QUE_CRITICAL	This function cannot be called from within a critical section.

96. qMsgSendNB

```

INTU qMsgSend(
    qtQUE pQue,           // The queue to send to
    qtMSG pMsg);        // The message to send

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function sends a message to a queue and will return a non-zero value if successful. If there is no space it will return a value of zero. If there is a receiving thread waiting it will immediately deliver the message to the waiting thread without placing the message in the queue. Multiple threads can send messages but only one thread can wait on a queue to become available for sending.

The message system will increase the use-count of the message by one so the sender can free the message immediately. This makes development simple because the thread does not have to keep track how the message is used by other threads or fibers.

Parameters and return value

Parameter	Description
Returns INTU	A zero value indicates that the queue is full. A value of 1 indicates success.
qtQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.
qtMSG pMsg	The message to send to the queue or a waiting thread.

Error conditions

Error	Description
qERR_QUE_ISR	The function is called from an ISR.
qERR_QUE_ID	The object is not a queue object, has not been created or points to no object at all.
qERR_MSG_ID	The message is not allocated by qMsgAlloc().

97. qMsgSendTO

```

INTU qMsgSendTO (
    qtQUE pQue,           // The queue to send to
    qtMSG pMsg,          // The message to send
    INT32S TimeOut);    // The timeout

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function sends a message to a queue and if there is no space it will wait until there is space available or will timeout. If there is a receiving thread waiting it will immediately deliver the message to the waiting thread without placing the message in the queue. A receiving thread will be made ready to run.

The message system will increase the use-count of the message by one so the sender can free the message immediately. This makes development simple because the thread does not have to keep track how the message is used by other threads or fibers.

Parameters and return value

Parameter	Description
Returns INTU	A zero value indicates that the function timed-out. A value of 1 indicates success.
qtQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.
qtMSG pMsg	The message to send to the queue or a waiting thread.
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_QUE_ISR	The function is called from an ISR.
qERR_QUE_NO_START	The function is called before Q-Kernel is started.
qERR_QUE_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_QUE_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_QUE_ID	The object is not a queue object, has not been created or points to no object at all.
qERR_MSG_ID	The message is not allocated by qMsgAlloc().
qERR_QUE_CRITICAL	This function cannot be called from within a critical section.
qERR_QUE_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_QUE_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_QUE_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.

98. qMsgWrite

```

INTU qMsgWrite(           //
    qtPIP pPip,          // The pipe to write into
    qtMSG pMsg);         // A pointer to the message

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
--------------	--------	--------------------	-------	-----	---------------------

Description

This function writes a message into the pipe. The function returns 0 if there is no space in the pipe and 1 if the message is written.

The system will notify the reader that this function has been executed.

Parameters and return value

Parameter	Description
Returns INTU	Zero if no space and 1 if the message is written.
qtPIP pPip	The pipe object to write to.
qtMSG pMsg	The message to be written in the pipe.

Error conditions

Error	Description
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.
qERR_PIP_ISR	The function is called from an interrupt handler.
qERR_MSG_ID	The object is not a message object, has not been created or points to no object at all.

99. qMsgWriteISR

```

INTU qMsgWriteISR(    //
    qtPIP pPip,      // The pipe to write into
    qtMSG pMsg);     // A pointer to the message

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function writes a message into the pipe. The function returns 0 if there is no space in the pipe and 1 if the message is written.

The system will notify the reader that this function has been executed.

Parameters and return value

Parameter	Description
Returns INTU	Zero if no space and 1 if the message is written.
qtPIP pPip	The pipe object to write to.
qtMSG pMsg	The message to be written in the pipe.

Error conditions

Error	Description
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.
qERR_PIP_ISR	The function is not called from an interrupt handler.
qERR_MSG_ID	The object is not a message object, has not been created or points to no object at all.

100. qMtxClose

```
void qMtxClose(
    qtMTX pMtx)           // The mutex to close
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Closes a mutex and returns the resources back to the resource pool. It is the developer responsibility to make sure that the mutex is not used by other threads. The function will test if any thread has the mutex locked but it does not detect if other threads are using this mutex object. The system will invalidate the object so other function can't use the object accidentally.

Parameters and return value

Parameter	Description
qtMTX pMtx	A pointer to the object. Must be returned from the qMtxCreate() or qMtxOpen() function with the correct name.

Error conditions

Error	Description
qERR_MTX_ISR	The function is called from an ISR.
qERR_MTX_ID	The object is not a mutex object, has not been created or points to no object at all.
qERR_MTX_IN_USE	A thread has locked the mutex so it can't be closed. The system can't detect if other threads are using this mutex.

The developer is responsible for checking if the mutex object is not used anymore.

101. qMtxCreate

```
qtMTX qMtxCreate(
    char *pName,           // The Name of the mutex
    INT8U Lock);          // !=0 create it locked
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

Before a mutex can be used, it has to be created by calling this function. On creation, the mutex can be locked. If there is an open request for the mutex with this name that thread will be readied, this creates a possible preemption. Multiple threads can wait for the object to be created and all threads will be readied.

The function needs memory for its operation, that's being freed when qMtxClose() end the use of the mutex. The function tries to allocate memory from the variable pool or from the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtMTX	The function returns a pointer to the mutex object.
char *pName	The name of the mutex. The name must be unique within other mutex objects or qNO_NAME which is a null pointer. qMtxOpen() can be used to locate the object if the name is not a null pointer.
INT8U Lock	A non-zero value will lock the Mutex after creation.

Error conditions

Error	Description
qERR_MTX_ISR	The function is called from an ISR.
qERR_MTX_FBR	The function is called from a fiber. Only threads or lightweight threads can own mutexes.
qERR_MTX_NAME_IN_USE	The name is already in use for another object
qERR_MTX_MEMORY	There is no memory available to handle the request.
qERR_KRN_LICENSE	Creating this object violates the license criteria.

102. qMtxLock

```
INTU qMtxLock (
    qtMTX pMtx);           // The mutex to lock
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
--------------	--------	--------------------	-------	-----	---------------------

Description

Wait for the mutex to become available. The function will return immediately if the mutex is available or wait for the mutex to become available. The function implements the priority inheritance mechanism to overcome priority inversion. Mutexes are owned by threads and for that reason it is impossible to lock a mutex from a fiber.

There is also a non-blocking version of this function. The non-blocking function is faster. Use this function if the size of the code is a concern and this function is already used in another place in the application because using both functions doubles the flash footprint.

Parameters and return value

Parameter	Description
Returns INTU	Returns 0 if timed out and 1 if mutex is locked
qtMTX pMtx	A pointer to the mutex object. Must be returned from qMtxCreate() or qMtxOpen() with the correct name.

Error conditions

Error	Description
qERR_MTX_ISR	The function is called from an ISR.
qERR_MTX_ID	The object is not a mutex object, has not been created or points to no object at all.
qERR_MTX_FBR	The function is called from a fiber. Only threads or lightweight threads can own mutexes.
qERR_MTX_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_MTX_NO_START	The function is called before Q-Kernel is started.

103. qMtxLockNB

```
INTU qMtxLockNB (
    qtMTX pMtx; )           // The mutex to lock
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Check if the mutex is available and lock it. Mutexes are owned by threads and for that reason it is impossible to lock a mutex from a fiber.

This is the non-blocking version of qMtxLock(). This function is faster. Use qMtxLock() if the size of the code is a concern and qMtxLock() is already used in another place in the application because using both functions doubles the flash footprint.

Parameters and return value

Parameter	Description
Returns INTU	Returns 0 if timed out and 1 if mutex is locked
qtMTX pMtx	A pointer to the mutex object. Must be returned from qMtxCreate() or qMtxOpen() with the correct name.

Error conditions

Error	Description
qERR_MTX_ISR	The function is called from an ISR.
qERR_MTX_ID	The object is not a mutex object, has not been created or points to no object at all.
qERR_MTX_FBR	The function is called from a fiber. Only threads or lightweight threads can own mutexes.
qERR_MTX_NO_START	The function is called before Q-Kernel is started.

104. qMtxLockTO

```

INTU qMtxLockTO (
    qtMTX pMtx,           // The mutex to lock
    INT32S TimeOut);     //

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

Wait for the mutex to become available. The function will return immediately if the mutex is available or wait for the mutex to become available. The function implements the priority inheritance mechanism to overcome priority inversion. Mutexes are owned by threads and for that reason it is impossible to lock a mutex from a fiber.

There is also a non-blocking version of this function. The non-blocking function is faster. Use this function if the size of the code is a concern and this function is already used in another place in the application because using both functions doubles the flash footprint.

Parameters and return value

Parameter	Description
Returns INTU	Returns 0 if timed out and 1 if mutex is locked
qtMTX pMtx	A pointer to the mutex object. Must be returned from qMtxCreate() or qMtxOpen() with the correct name.
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_MTX_ISR	The function is called from an ISR.
qERR_MTX_NO_START	The function is called before Q-Kernel is started.
qERR_MTX_ID	The object is not a mutex object, has not been created or points to no object at all.
qERR_MTX_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_MTX_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_MTX_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.
qERR_MTX_FBR	The function is called from a fiber. Only threads or lightweight threads can own mutexes.
qERR_MTX_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.

105. qMtxOpen

```
qtMTX qMtxOpen (           // Returns NULL when timed-out
  char *pName);           // The Name of the Mutex
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing mutex object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended.

Parameters and return value

Parameter	Description
Returns qtMTX	The function returns a pointer to the mutex object. If this pointer is a null pointer the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_MTX_ISR	The function is called from an ISR.
qERR_MTX_NO_START	The function is called before Q-Kernel is started.
qERR_MTX_FBR	The function is called from a fiber. Only threads or lightweight threads can own mutexes.
qERR_MTX_LWT	This function cannot be called from within a critical section.
qERR_MTX_NO_NAME	Mutexes without a name can't be opened.
qERR_MTX_CRITICAL	This function cannot be called from within a critical section.
qERR_MTX_MEMORY	There is no memory available to handle the open request.

106. qMtxOpenNB

```
qtMTX qMtxOpenNB(           //
    char *pName);           // The name of the mutex
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns a pointer to an existing mutex object.

Parameters and return value

Parameter	Description
Returns qtMTX	The function returns a pointer to the event object. The function returns NULL if the object does not exist.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_PIP_ISR	The function is called from an ISR.
qERR_PIP_NO_NAME	Pipes without a name can't be opened.

107. qMtxOpenTO

```
qtMTX qMtxOpenTO(           // Returns NULL when timed-out
    char *pName,           // The Name of the object
    INT32S TimeOut);      // The timeout
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended. The timeout specifies how long the thread is willing to wait.

Parameters and return value

Parameter	Description
Returns qtMTX	The function returns a pointer to the mutex object. If the pointer is null the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_MTX_ISR	The function is called from an ISR.
qERR_MTX_NO_START	The function is called before Q-<i>Kernel</i> is started.
qERR_MTX_FBR	The function is called from a fiber. Only threads or lightweight threads can own mutexes.
qERR_MTX_LWT	This function cannot be called from within a critical section.
qERR_MTX_NO_NAME	Mutexes without a name can't be opened.
qERR_MTX_CRITICAL	This function cannot be called from within a critical section.
qERR_MTX_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_MTX_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_MTX_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.
qERR_MTX_MEMORY	There is no memory available to handle the open request.

108. qMtxUnlock

```
void qMtxUnlock(
    qtMTX pMtx);           // The mutex to unlock
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
-----------------	--------	-----------------------	-------	-----	---------------------

Description

This function unlocks a locked mutex. If any threads are trying to lock the mutex, then the one with highest priority is selected and given the lock that was just released. That thread is enabled for thread scheduling purposes.

Parameters and return value

Parameter	Description
qtMTX pMtx	A pointer to the mutex object. Must be returned from qMtxCreate() or qMtxOpen() with the correct name.

Error conditions

Error	Description
qERR_MTX_ISR	The function is called from an ISR.
qERR_MTX_ID	The object is not a mutex object, has not been created or points to no object at all.
qERR_MTX_FBR	The function is called from a fiber. Only threads or lightweight threads can own mutexes.
qERR_MTX_NO_START	The function is called before Q-Kernel is started.
qERR_MTX_OWNER	The mutex is not unlocked by the owner of the mutex.

109. qNtfError

```
INTU qNtfError(
    INTU Error);           // The Error that has be Signaled
```

Description

The developer can provides this function and it will be called when **Q-Kernel** throws an unrecoverable error. A prime example of an unrecoverable error is calling a create function from within an ISR. A prime example of a recoverable error is a time-out on a wait for an event.

The application should as a minimum log the error and restart the system. More advanced recover methods are deleting the thread and try to continue. Most errors are not signaled when the system uses optimized versions of the **Q-Kernel** that don't check.

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_MEMORY	There is no memory available to handle the request.

The blue colored error qERR_TMR_ISR will not be notified with the no-checking versions. The qERR_TMR_MEMORY error will be notified in all versions.

In most **Q-Kernel** implementations the developer does not have to specify this function. In that case the default function will be executed. The default function will set the variables qvErrNbr and qvErrTcb. In some **Q-Kernel** implementations the developer has to provide the this function. See the user guide for more information.

Parameters and return value

Parameter	Description
Return INTU	The developer must return 0 if the function is called with 0.
INTU Error	The error that will be signaled. If this value is zero there is no error and the function should return zero.

A common mistake is that the function can't handle the 0 case.

110. qNtfIdle

```
void qNtfIdle(); // Called from idle thread.
```

Description

The developer provides this function and it will be called from the idle thread. It will be called repeatedly. The idle thread code is listed below:

```
void qqIdleThread (void* p) { // The idle thread
    while(1) {
        ... .. // Handle lightweight threads
        qNtfIdle(); // qNtfIdle() called as fiber
    }
}
```

The developer does not have to specify this function because there is a default function that just returns to the caller.

This function is only called in the demo and basic version of **Q-Kernel**. The full version will call the power management functions.

111. qNtfSwitch

```
void qNtfSwitch(  
    qtTCB *pCurThr,           // The Thread that is preempted  
    qtTCB *pNextThr);       // Next Thread to run
```

Description

The developer can or must provide this function. The function will be called when switch notification is enabled and **Q-Kernel** switches from one thread to another thread. Some **Q-Kernel** implementations provide a "weak" function that will be called when the developer does not provides one.

The developer can use the qtTCB structure to store data. The first parameter is the TCB of the thread that will be preempted and the second parameter is the TCB of the thread that will we activated. Because the function will be called from a fiber or a thread, stack requirements can come from the interrupt stack or a thread stack. The developer should use this function only if absolute necessary because it creates overhead. Also keep the stack usage to the absolute minimum.

This function can be used to save and restore specific thread context. An example of this is to save specific CPU registers like TBLPAG. By saving and restoring those registers they can be used by more than one thread without mutex or critical sections.

After the start of **Q-Kernel** switch notification is off, so before this function will be called by **Q-Kernel** the switch notification must be enabled by executing the function qKrnSwitchNotificationOn().The default function, defined as "weak" simply returns to the caller.

The function qKrnSwitchNotificationOff() is also available to disable switch notification.

112. qNtfStat

```
void qNtfStat(); //
```

Description

This function will be called at the end of every lap (every second) when statistics is configured and enabled. The developer can use this hook to extend the statistic functionality.

Parameters and return value

None

113. qNtfPower

```
qtPWR qNtfPower(
    qtPWR advise);    //
```

Description

This function will be called if the processor does not have anything to do, meaning all interrupts are handled, fibers are run and all threads are in a wait mode. The parameter provided specifies the advised power saving mode. The return variable specifies the idle mode that the developer wants to set.

While maximum power savings requires “design for power savings”, moderate savings can be accomplished without much effort. The following example code enables moderate savings:

```
qtPWR qNtfPower(qtPWR advise) {
    if (advise==qPWR_NORMAL) // No power mode switch
        return qPWR_NORMAL; // Then just return
    return qPWR_IDLE;       // Always set Idle for ...
};                          // ... advise sleep or idle
```

This example will just switch to idle mode and keeps all hardware devices running. It is simple to implement and power savings could be significant depending on the situation.

This function is only called in the full license scenario.

Parameters and return value

Parameter	Description
Returns qtPWR	The power mode that the system should set.
qtPWR advise	Advised power mode.

114. qPipBlockSize

```
INTU qPipBlockSize( // returns block size
qtPIP pPip);
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns the block size and is the same as the parameter BlockSize in qPipCreate().

Parameters and return value

Parameter	Description
Returns INTU	The block size of the pipe. This is the same as the parameter BlockSize in qPipCreate()
qtPIP pPip	The pipe object to get the information from.

Error conditions

Error	Description
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.

115. qPipClose

```
void qPipClose(
    qtPIP pPip)           // The Pipe to close
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Closes a pipe and returns the resources back to the resource pool. It is the developer responsibility to make sure that the pipe is not used by other threads, fibers or ISRs. The system will invalidate the object so other function can't use the object accidentally.

Parameters and return value

Parameter	Description
qtPIP pPip	A pointer to the object. Must be returned from the qPipCreate() or qPipOpen() function with the correct name.

Error conditions

Error	Description
qERR_PIP_ISR	The function is called from an ISR.
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.

The developer is responsible for checking that the pipe is not is use.

116. qPipCreate

```

qtPIP qPipCreate(
    char *pName,                // The Name of the pipe
    INT16U BlockSize,           // Size of one block
    INT16U MaxBlocks,           // Maximum blocks
    void (*pNtfReader)(qtPIP,INTU), // Notify reader
    void (*pNtfWriter)(qtPIP,INTU)); // Notify writer

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
-----------------	--------	-----------------------	-------	-----	---------------------

Description

Before a pipe can be used, it has to be created by calling this function. If there is an open request for the pipe with this name that thread will be readied, which creates a possible preemption. Multiple threads can wait for the object to be created and all threads will be readied.

A pipe is a communication mechanism between threads, fibers and ISR's. The pipe can be read or written by an ISR's but not both at the same time and only from one ISR level. If a pipe is written to, the notify reader function is called and when a pipe is read from, the notify writer function is called.

If the BlockSize is a multiply of the integer size all reads and writes must be integer aligned. The compiler aligns structure so that is normally not a problem.

If the pipe is used for messages you must specify the BlockSize as sizeof(qtMSG).

The function needs memory to create the object. It tries to allocate memory from the variable memory pool or the heap.

The developer must give every object a unique name.

Parameters and return value

Parameter	Description
Returns qtPIP	The function returns a pointer to the pipe object.
char *pName	The name of the pipe. The name must be unique within other pipe objects or qNO_NAME which is a null pointer. qPipOpen() can be used to locate the object if the name is not a null pointer.
INT8U BlockSize	The size of one block. The system will use this value and the next one to determine the size of the pipe. Use the C sizeof() keyword so the value is automatically adjusted if you migrate to another processor. Use sizeof(qtMSG) for messages.
INT16U MaxBlocks	The number of blocks that the pipe can hold. The system will allocate enough space to hold the data. The minimum size is 2.
void (*pNtfReader) (qtPIP, INTU)	The reader notification function that will be called every time something is delivered in the pipe. The first parameter of the function contains a pointer to the pipe object and the second parameter is the number of blocks written.
void (*pNtfWriter) (qtPIP, INTU)	The writer notification function that will be called every time something is read from the pipe. The first parameter of the function contains a pointer to the pipe object and the second parameter is the number of blocks read.

Error conditions

Error	Description
qERR_PIP_ISR	The function is called from an ISR.
qERR_PIP_BLOCK_SIZE	The blocks size is incorrect.
qERR_PIP_NBR_BLOCKS	The number of blocks is incorrect
qERR_PIP_NAME_IN_USE	The name is already in use for another object
qERR_PIP_MEMORY	There is no memory available to handle the request.
qERR_KRN_LICENSE	Creating this object violates the license criteria.

117. qPipEntries

```
INTU qPipEntries( // returns number of block ...
qtPIP pPip); // ... currently in the pipe
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns the number of entries in the pipe.

Parameters and return value

Parameter	Description
Returns INTU	The number of entries in the pipe. If the pipe is empty it returns 0.
qtPIP pPip	The pipe object to get the information from.

Error conditions

Error	Description
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.

118. qPipGet

```

INTU qPipGet(           // returns number of elements read
  qtPIP pPip,         // The pipe to read from
  void *pBuffer,      // A pointer to the data
  INTU NbrBlocks);    // Size of the buffer in blocks

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function reads information from a pipe. It reads never more than the specified number of blocks to prevent overflow. The function returns the number of blocks read or zero when there is no information in the pipe.

The buffer is specified as void but it is the developer's responsibility to define the correct data type. This function does not call the notification writer and is not protected by a critical section. It is the developer's responsibility to protect the integrity of the data. This function is added as convenient way to read data from pipes without synchronization overhead. Use qPipRead() or qPipReadISR() for synchronized reading from a pipe.

Parameters and return value

Parameter	Description
Returns INTU	The number of elements read.
qtPIP pPip	The pipe object to read from.
void *pBuffer	The buffer to read the information in. The developer must specify the correct buffer type. If the BlockSize is a multiply of the integer size the buffer MUST be integer aligned.
INTU NbrBlocks	The number of blocks that the buffer can contain. A value of zero is allowed but the function will not read anything.

Error conditions

No errors are thrown

119. qPipMaxBlocks

```
INTU qPipMaxBlocks( // returns maximum number of ...
qtPIP pPip);      // ... blocks in the pipe
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns the maximum number of blocks that fit in the pipe and is the same as the parameter MaxBlocks in qPipCreate()

Parameters and return value

Parameter	Description
Returns INTU	The maximum number of blocks in the pipe. This is the same as the parameter MaxBlocks in qPipCreate()
qtPIP pPip	The pipe object to get the information from.

Error conditions

Error	Description
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.

120. qPipOpen

```
qtPIP qPipOpen (           // Returns NULL when timed-out
    char *pName);         // The Name of the queue
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing pipe object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended. The timeout specifies how long the thread is willing to wait.

The function needs memory for its operation, that's being freed when qPipCreate() activates the thread. It tries to allocate memory from the variable memory pool or the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtPIP	The function returns a pointer to the pipe object. If the pointer is a null pointer the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_PIP_ISR	The function is called from an ISR.
qERR_PIP_NO_START	The function is called before the kernel is started.
qERR_PIP_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_PIP_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_PIP_NO_NAME	Pipes without a name can't be opened.
qERR_PIP_CRITICAL	This function cannot be called from within a critical section.
qERR_PIP_MEMORY	There is no memory available to handle the open request.

121. qPipOpenNB

```
qtPIP qPipOpenNB(           //
    char *pName);           // The Name of the EventSet
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns a pointer to an existing pipe object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended. The timeout specifies how long the thread is willing to wait.

The function needs memory for its operation, that's being freed when qPipCreate() activates the thread. It tries to allocate memory from the variable pool or from the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtPIP	The function returns a pointer to the event object. The function returns NULL if the object does not exist.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_PIP_ISR	The function is called from an ISR.
qERR_PIP_NO_NAME	Pipes without a name can't be opened.
qERR_PIP_MEMORY	There is no memory available to handle the open request.

122. qPipOpenTO

```

qtPIP qPipOpenTO(           // Returns NULL when timed-out
    char *pName,           // The Name of the object
    INT32S TimeOut);       // The timeout

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended. The timeout specifies how long the thread is willing to wait.

The function needs memory for its operation, that's being freed when qPipCreate() activates the thread. It tries to allocate memory from the variable pool or from the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtPIP	The function returns a pointer to the pipe object. If the pointer is null the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_PIP_ISR	The function is called from an ISR.
qERR_PIP_NO_START	The function is called before Q-Kernel is started.
qERR_PIP_NO_NAME	Objects without a name can't be opened.
qERR_PIP_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_PIP_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_PIP_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.
qERR_PIP_CRITICAL	This function cannot be called from within a critical section.
qERR_PIP_MEMORY	There is no memory available to handle the open request.

123. qPipPut

```

INTU qPipPut(           // returns nbr of elements written
  qtPIP pPip,          // The pipe to write into
  void *pBuffer,       // A pointer to the data
  INTU NbrBlocks);     // Size of buffer in number of ...
                       // ... elements

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function writes information into the pipe. The function returns the number of blocks written. If the return value is not equal to the NbrBlocks the pipe is full and the function returns the number of blocks written. The minimum number of elements to write is one.

The buffer is specified as void but it is the developer's responsibility to define the correct data type. This function does not call the notification reader and is not protected by a critical section. It is the developer's responsibility to protect the integrity of the data. This function is added as convenient way to write data to pipes without synchronization overhead. Use qPipWrite() or qPipWriteISR() for synchronized writing to a pipe.

Parameters and return value

Parameter	Description
Returns INTU	The number of elements written.
qtPIP pPip	The pipe object to write to.
void *pBuffer	The buffer that contains the information. The developer must specify the correct buffer type.
INTU NbrBlocks	The size of the buffer in number of elements. This MUST be one or higher.

Error conditions

This function does not throw errors.

124. qPipRead

```

INTU qPipRead(           // returns number of elements read
    qtPIP pPip,         // The pipe to read from
    void *pBuffer,      // A pointer to the data
    INTU NbrBlocks);    // Size of the buffer in blocks

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function reads information from a pipe. It reads never more than the specified number of blocks to prevent overflow. The function returns the number of blocks read or zero when there is no information in the pipe. The buffer is specified as void and it is the developer's responsibility to define the correct data type.

The system will notify the writer that it has read the pipe. While the manipulation of the buffer is in a critical section, to allow multiple readers, the notification writer is called out-side the critical section so it can bring the calling thread in a blocking state.

Parameters and return value

Parameter	Description
Returns INTU	The number of elements read.
qtPIP pPip	The pipe object to read from.
void *pBuffer	The buffer to read the information in. The developer must specify the correct buffer type.
INTU NbrBlocks	The number of elements that the buffer can contain of buffer. This must be one or higher.

Error conditions

Error	Description
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.
qERR_PIP_ISR	The function is called from an ISR.
qERR_PIP_NBR_BLOCKS	The buffer size is incorrect. Must be 1 or higher.

125. qPipReadISR

```

INTU qPipReadISR(      // returns number of elements read
    qtPIP pPip,        // The pipe to read from
    void *pBuffer,     // A pointer to the data
    INTU NbrBlocks);  // Size of the buffer in blocks

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function reads information from a pipe. It reads never more than the specified number of blocks to prevent overflow. The function returns the number of blocks read or zero when there is no information in the pipe. The buffer is specified as void and it is the developer's responsibility to define the correct data type.

The system will notify the writer that it has read the pipe. The notification writer is called in-side the interrupt so the notification writer can only use functions that can be called from an interrupt service routine.

Parameters and return value

Parameter	Description
Returns INTU	The number of elements read.
qtPIP pPip	The pipe object to read from.
void *pBuffer	The buffer to read the information in. The developer must specify the correct buffer type.
INTU NbrBlocks	The number of elements that the buffer can contain of buffer. This must be one or higher.

Error conditions

Error	Description
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.
qERR_PIP_ISR	The function is not called from an ISR.
qERR_PIP_NBR_BLOCKS	The buffer size is incorrect. Must be 1 or higher.

126. qPipWrite

```

INTU qPipWrite(           // returns nbr of elements written
    qtPIP pPip,          // The pipe to write into
    void *pBuffer,       // A pointer to the data
    INTU NbrBlocks);     // Size of buffer in number of ...
                        // ... blocks

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function writes information into the pipe. The function returns the number of blocks written. If the return value is not equal to the NbrBlocks the pipe is full and the function returns the number of blocks written. The buffer is specified as void but it is the developer's responsibility to define the correct data type.

The system will notify the reader that it has written the pipe. While the manipulation of the buffer is in a critical section, to allow multiple writers, the notification reader is called out-side the critical section so it can bring the calling thread in a blocking state.

Parameters and return value

Parameter	Description
Returns INTU	The number of elements written.
qtPIP pPip	The pipe object to write to.
void *pBuffer	The buffer that contains the information. The developer must specify the correct buffer type.
INTU NbrBlocks	The size of the buffer in number of blocks.

Error conditions

Error	Description
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.
qERR_PIP_ISR	The function is called from an ISR.
qERR_PIP_NBR_BLOCKS	The buffer size is incorrect. Must be 1 or higher.

127. qPipWriteISR

```

INTU qPipWriteISR( // returns nbr of elements written
    qtPIP pPip,    // The pipe to write into
    void *pBuffer, // A pointer to the data
    INTU NbrBlocks); // Size of buffer in number of ...
                    // ... blocks

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function writes information into the pipe. The function returns the number of blocks written. If the return value is not equal to the NbrBlocks the pipe is full and the function returns the number of blocks written. The buffer is specified as void but it is the developer's responsibility to define the correct data type.

The system will notify the reader that it has written the pipe. The notification reader is called in-side the interrupt so the notification reader can only use functions that can be called from an interrupt service routine.

Parameters and return value

Parameter	Description
Returns INTU	The number of elements written.
qtPIP pPip	The pipe object to write to.
void *pBuffer	The buffer that contains the information. The developer must specify the correct buffer type.
INTU NbrBlocks	The size of the buffer in number of blocks.

Error conditions

Error	Description
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.
qERR_PIP_ISR	The function is not called from an ISR.
qERR_PIP_NBR_BLOCKS	The buffer size is incorrect. Must be 1 or higher.

128. qPubClose

```
void qPubClose (
    qtPUB pPub)           // The publisher to close
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Closes the publisher and all subscribers and returns the resources back to the resource pool. The system will invalidate the object so other function can't use the object accidentally.

Parameters and return value

Parameter	Description
qtPUB pPub	A pointer to the object. Must be returned from the qPubCreate() or qPubOpen() function with the correct name.

Error conditions

Error	Description
qERR_PUB_ISR	The function is called from an ISR.
qERR_PUB_ID	The object is not a publisher object, has not been created or points to no object at all.

129. qPubCreate

```
qtPUB qPubCreate (
    char *pName);           // The Name of the semaphore
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
--------------	--------	--------------------	-------	-----	---------------------

Description

Before the publish/subscribe mechanism can be used, it has to be created by calling this function. The subscribers can open the publication and subscriber to the publication after creation of the publication.

Parameters and return value

Parameter	Description
Returns qtPUB	The function returns a pointer to the publication object.
char *pName	The name of the publication. The name must be unique within other publication objects or qNO_NAME which is a null pointer. qPubOpen() can be used to locate the object if the name is not a null pointer.

Error conditions

Error	Description
qERR_PUB_ISR	The function is called from an ISR.
qERR_PUB_NAME_IN_USE	The name is already in use for another object
qERR_PUB_MEMORY	There is no memory available to handle the request.
qERR_KRN_LICENSE	Creating this object violates the license criteria.

130. qPubOpen

```
qtPUB qPubOpen (           // Returns NULL when timed-out
    char *pName);         // The Name of the semaphore
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
--------------	--------	--------------------	-------	-----	---------------------

Description

This function returns a pointer to an existing Publish object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended.

Parameters and return value

Parameter	Description
Returns qtPUB	The function returns a pointer to the publish object.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_PUB_ISR	The function is called from an ISR.
qERR_PUB_NO_START	The function is called before Q-Kernel is started.
qERR_PUB_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_PUB_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_PUB_NO_NAME	Semaphores without a name can't be opened.
qERR_PUB_CRITICAL	This function cannot be called from within a critical section.
qERR_PUB_MEMORY	There is no memory available to handle the open request. All open functions return the same error.

131. qPubOpenNB

```
qtPUB qPubOpenNB (           //
    char *pName);           // The Name of the object
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns a pointer to an existing publish object.

Parameters and return value

Parameter	Description
Returns qtPUB	The function returns a pointer to the Publish object. The function returns NULL if the object does not exist.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_PUB_ISR	The function is called from an ISR.
qERR_PUB_NO_NAME	Object without a name can't be opened.

132. qPubOpenTO

```

qtPUB qPubOpenTO(           // Returns NULL when timed-out
    char *pName,           // The Name of the object
    INT32S TimeOut);       // The timeout

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing Publish object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended. The timeout specifies how long the thread is willing to wait.

Parameters and return value

Parameter	Description
Returns qtPUB	The function returns a pointer to the publish object. If the pointer is null the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_PUB_ISR	The function is called from an ISR.
qERR_PUB_NO_START	The function is called before Q-Kernel is started.
qERR_PUB_NO_NAME	Objects without a name can't be opened.
qERR_PUB_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_PUB_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_PUB_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.
qERR_PUB_CRITICAL	This function cannot be called from within a critical section.
qERR_PUB_MEMORY	There is no memory available to handle the open request.

133. qPubSubscribeFun

```
void qPubSubscribeFun(
    qtPUB pPub,          // The publisher to subscribe
    void (*pFun)(void*, qtMSG);
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function binds the subscriber function to the publisher. Every time a message is published by the publisher the function is called with a NULL parameter and the message.

Parameters and return value

Parameter	Description
qtPUB pPub	A pointer to the publish object. Must be returned from qPubCreate() or qPubOpen() with the correct name.
void (*pFun)(void*, qtMSG)	The function that will be called by the publisher of a message. The first parameter is always NULL by convention and the second parameter is the message.

Error conditions

Error	Description
qERR_PUB_ISR	The function is called from an ISR.
qERR_PUB_ID	The object is not a publish object, has not been created or points to no object at all.

134. qPubSubscribePip

```
void qPubSubscribePip(
    qtPUB pPub,      // The publisher to subscribe
    qtPIP pPip);
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function binds the subscriber function to a pipe. Every time a message is published by the publisher the message is written in the pipe.

Parameters and return value

Parameter	Description
qtPUB pPub	A pointer to the publish object. Must be returned from qPubCreate() or qPubOpen() with the correct name.
qtPIP pPip	The pipe that will be used to send the message.

Error conditions

Error	Description
qERR_PUB_ISR	The function is called from an ISR.
qERR_PUB_ID	The object is not a publish object, has not been created or points to no object at all.
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.

135. qPubSubscribeQue

```
void qPubSubscribeQue (
    qtPUB pPub,          // The publisher to subscribe
    qtQUE pQue);
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function binds the subscriber function to a queue. Every time a message is published by the publisher the message is sent to the queue.

Parameters and return value

Parameter	Description
qtPUB pPub	A pointer to the publish object. Must be returned from qPubCreate() or qPubOpen() with the correct name.
qtQUE pQue	The pipe that will be used to send the message.

Error conditions

Error	Description
qERR_PUB_ISR	The function is called from an ISR.
qERR_PUB_ID	The object is not a publish object, has not been created or points to no object at all.
qERR_QUE_ID	The object is not a queue object, has not been created or points to no object at all.

136. qQueClose

```
void qQueClose (
    qtQUE pQue);           // The queue to close
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Closes a queue and returns the resources back to the resource pool. It is the developer responsibility to make sure that the queue is not used by other threads or fibers. The function will test if any thread is waiting on the queue but it does not detect if other threads are using this queue object. The system will invalidate the object so other function can't use the object accidentally.

Parameters and return value

Parameter	Description
qtQUE pQue	A pointer to the object. Must be returned from the qQueCreate() or qQueOpen() function with the correct name.

Error conditions

Error	Description
qERR_QUE_ISR	The function is called from an ISR.
qERR_QUE_ID	The object is not a queue object, has not been created or points to no object at all.
qERR_QUE_IN_USE	Other thread(s) are waiting for the queue. The system can't detect if other threads are using the queue.

The developer is responsible for checking if the queue object is not used anymore.

137. qQueCreate

```
qtQUE qQueCreate(
    char *pName,           // The Name of the queue
    INT16U Size);         // Nbr of message in queue
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

Before a queue can be used, it has to be created by calling this function. If there is an open request for the queue with this name that thread will be readied, which creates a possible preemption. Multiple threads can wait for the object to be created and all threads will be readied. The function needs memory to create the object. It tries to allocate memory from the variable memory pool or the heap.

Parameters and return value

Parameter	Description
Returns qtQUE	The function returns a pointer to the queue object.
char *pName	The name of the queue. The name must be unique within other queue objects or qNO_NAME which is a null pointer. qQueOpen() can be used to locate the object if the name is not a null pointer.
INTU Size	The number of messages in the queue. The system will allocate an array of message of this size. The minimum size is 1.

Error conditions

Error	Description
qERR_QUE_ISR	The function is called from an ISR.
qERR_QUE_NAME_IN_USE	The name is already in use for another object
qERR_QUE_SIZE	The size of the queue is incorrect.
qERR_QUE_MEMORY	There is no memory available to handle the request.
qERR_KRN_LICENSE	Creating this object violates the license criteria.

138. qQueOpen

```
qtQUE qQueOpen (           // Returns NULL when timed-out
    char *pName);         // The Name of the queue
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing queue object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended. The timeout specifies how long the thread is willing to wait.

The function needs memory for its operation, that's being freed when qQueCreate() activates the thread. It tries to allocate memory from the variable memory pool or the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtQUE	The function returns a pointer to the queue object. If the pointer is a null pointer the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_QUE_ISR	The function is called from an ISR.
qERR_QUE_NO_START	The function is called before the kernel is started.
qERR_QUE_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_QUE_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_QUE_NO_NAME	Message queues without a name can't be opened.
qERR_QUE_CRITICAL	This function cannot be called from within a critical section.
qERR_QUE_MEMORY	There is no memory available to handle the open request.

139. qQueueOpenNB

```
qtQUE qQueueOpenNB (           //
    char *pName);             // The Name of the EventSet
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns a pointer to an existing queue object.

Parameters and return value

Parameter	Description
Returns qtQUE	The function returns a pointer to the queue object. The function returns NULL if the object does not exist.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_QUEUE_ISR	The function is called from an ISR.
qERR_QUEUE_NO_NAME	Queues without a name can't be opened.

140. qQueOpenTO

```

qtQUE qQueOpenTO(           // Returns NULL when timed-out
    char *pName,           // The Name of the object
    INT32S TimeOut);       // The timeout

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended. The timeout specifies how long the thread is willing to wait.

The function needs memory for its operation, that's being freed when qQueCreate() activates the thread. It tries to allocate memory from the variable pool or from the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtQUE	The function returns a pointer to the queue object. If the pointer is null the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_QUE_ISR	The function is called from an ISR.
qERR_QUE_NO_START	The function is called before Q-Kernel is started.
qERR_QUE_NO_NAME	Objects without a name can't be opened.
qERR_QUE_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_QUE_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_QUE_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.
qERR_QUE_CRITICAL	This function cannot be called from within a critical section.
qERR_QUE_MEMORY	There is no memory available to handle the open request.

141. qRtcAlarm

```
void qRtcAlarm(
    INT32S Time,           // The time
    void (*pFbr)(void *p)); // The function to call
void (*pFbr)();         // The parameter
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function sets an alarm and will call the function as a fiber when the time expires. If the specified time has already been reached, the function is called immediately.

Parameters and return value

Parameter	Description
void (*pFbr)()	The function to call after the time expires
void *pParam	The parameter for the function. This allows to share the function for several timers
INTU Time	The time in milliseconds for the timer to expire and call the function. Valid values are between 1 and 60,000 inclusive for the 8-bit and 16 bit versions and between 1 and 2,000,000,000 inclusive for the 32 bit versions.

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_FUNCTION	No function specified.
qERR_TMR_MEMORY	There is no memory available to handle the request.
qERR_KRN_LICENSE	Creating this object violates the license criteria.

142. qRtcAlarm

```

void qTmrCreate(
    void (*pFbr)(void *p), // The function to call
    void *pParam,          // The parameter when called
    INT32S Time);          // The real time to expire

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function sets an alarm and will call the function as a fiber when the time expires. If the specified time has already been reached, the function is called immediately.

Parameters and return value

Parameter	Description
Returns qtTMR	The function returns a pointer to the timer object.
void (*pFbr)(void *p)	The function to call after the time expires.
void *pParam	The parameter for the function. This allows sharing of the function for several alarms.
INT32S Time	<p>A positive timeout value is not allowed because it specifies cycles to wait.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_FUNCTION	No function specified.
qERR_RTC_1_1_2010	The time is incorrect because it is positive or before January 1 st 2010
qERR_TMR_MEMORY	There is no memory available to handle the request.
qERR_KRN_LICENSE	Creating this object violates the license criteria.

143. qRtcGetDatTim

```
INT32U qRtcGetDatTim(); //
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
--------------	--------	--------------------	-------	-----	---------------------

Description

This function returns the current data time in internal format. The function will return 0 if the date time has not been set.

Parameters and return value

Parameter	Description
Returns INT32U	Date time in internal format.

Error conditions

None

144. qRtcGetUptime

```
INT32U qRtcGetUptime(); //
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
--------------	--------	--------------------	-------	-----	---------------------

Description

This function returns the uptime.

Parameters and return value

Parameter	Description
Returns INT32U	The uptime in seconds

Error conditions

None

145. qRtcSetDatTim

```
void qRtcSetDatTim(
    INT32U DatTim);    //
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function sets the date time. This function could preempt if the date time is set back.

Parameters and return value

Parameter	Description
INT32U DatTim	The correct date time in internal format

Error conditions

Error	Description
qERR_RTC_ISR	The function is called in an ISR
qERR_RTC_NO_START	The system is not started
qERR_RTC_FBR	The function is called from a fiber
qERR_RTC_LWT	The function is called from a lightweight thread
qERR_RTC_CRITICAL	The function is called from a critical section
qERR_RTC_1_1_2010	The specified date time is before 1/1/2010

146. qSemAcquire

```
INTU qSemAcquire (
    qtSEM pSem) ;           // Semaphore to acquire the permit
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
--------------	--------	--------------------	-------	-----	---------------------

Description

Acquires a permit, if one is available and returns immediately, with the value of 1, reducing the number of available permits by one. If no permit is available then the current thread will be preempted. The function returns 0 if the specified waiting time elapses. A value of 1 will be returned if the permit is acquired.

There is also a non-blocking version of this function. The non-blocking function is faster. Use this function if the size of the code is a concern and this function is already used in another place in the application because using both functions doubles the flash footprint.

Parameters and return value

Parameter	Description
Returns INTU	Returns 0 if timed out and 1 if permit is granted
qtSEM pSem	A pointer to the semaphore object. Must be returned from qSemCreate() or qSemOpen() with the correct name.

Error conditions

Error	Description
qERR_SEM_ISR	The function is called from an ISR.
qERR_SEM_NO_START	The function is called before Q-Kernel is started.
qERR_SEM_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_SEM_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_SEM_NO_NAME	Semaphores without a name can't be opened.
qERR_SEM_CRITICAL	This function cannot be called from within a critical section.

147. qSemAcquireNB

```
INTU qSemAcquireNB (
    qtSEM pSem);           // Semaphore to acquire the permit
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
-----------------	--------	-----------------------	-------	-----	---------------

Description

Acquires a permit and returns immediately.

This is the faster non-blocking version of qSemAcquire(). Use qSemAcquire() if the size of the code is a concern and qSemAcquireNB() is already used in another place in the application because using both functions doubles the flash footprint.

Parameters and return value

Parameter	Description
Returns INTU	Returns 0 if timed out and 1 if permit is granted
qtSEM pSem	A pointer to the semaphore object. Must be returned from qSemCreate() or qSemOpen() with the correct name.

Error conditions

Error	Description
qERR_SEM_ISR	The function is called from an ISR.
qERR_SEM_NO_START	The function is called before Q-Kernel is started.
qERR_SEM_ID	The object is not a semaphore object, has not been created or points to no object at all.

148. qSemAcquireTO

```
INTU qSemAcquireTO (
    qtSEM pSem,           // Semaphore to acquire the permit
    INT32S TimeOut);     // The timeout
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
--------------	--------	--------------------	-------	-----	---------------------

Description

Acquires a permit, if one is available and returns immediately, with the value of 1, reducing the number of available permits by one. If no permit is available then the current thread will be preempted. The function returns 0 if the specified waiting time elapses. A value of 1 will be returned if the permit is acquired.

There is also a non-blocking version of this function. The non-blocking function is faster. Use this function if the size of the code is a concern and this function is already used in another place in the application because using both functions doubles the flash footprint.

Parameters and return value

Parameter	Description
Returns INTU	Returns 0 if timed out and 1 if permit is granted
qtSEM pSem	A pointer to the semaphore object. Must be returned from qSemCreate() or qSemOpen() with the correct name.
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_SEM_ISR	The function is called from an ISR.
qERR_SEM_NO_START	The function is called before Q-Kernel is started.
qERR_SEM_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_SEM_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_SEM_NO_NAME	Semaphores without a name can't be opened.
qERR_SEM_CRITICAL	This function cannot be called from within a critical section.
qERR_SEM_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_SEM_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_SEM_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.

149. qSemClose

```
void qSemClose (
    qtSEM pSem)           // The semaphore to close
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Closes semaphore and returns the resources back to the resource pool. It is the developer responsibility to make sure that the semaphore is not used by other threads or fibers. The function will test if any thread is waiting on the semaphore but it does not detect if other thread or fibers are using this semaphore object. The system will invalidate the object so other function can't use the object accidentally.

Parameters and return value

Parameter	Description
qtSEM pSem	A pointer to the object. Must be returned from the qSemCreate() or qSemOpen() function with the correct name.

Error conditions

Error	Description
qERR_SEM_ISR	The function is called from an ISR.
qERR_SEM_ID	The object is not a semaphore object, has not been created or points to no object at all.
qERR_SEM_IN_USE	The semaphore object is in use by other threads or fibers. More specific other threads are waiting for it. The system can't detect of other thread or fibers are using this semaphore.

The developer is responsible for checking if the semaphore object is not used anymore.

150. qSemCreate

```
qtSEM qSemCreate (
    char *pName,           // The Name of the semaphore
    INTU Permits);        // The number of initial permits
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
--------------	--------	--------------------	-------	-----	---------------------

Description

Before a semaphore can be used, it has to be created by calling this function. The creating thread or fiber specifies the initial number of permits and a name for the semaphore object. If there is an open request for the semaphore with this name that thread will be readied, which creates a possible preemption. Multiple threads can wait for the object to be created and all threads will be readied.

The function needs memory to create the object. It tries to allocate memory from the variable pool or the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtSEM	The function returns a pointer to the semaphore object.
char *pName	The name of the semaphore. The name must be unique within other semaphore objects or qNO_NAME which is a null pointer. qSemOpen() can be used to locate the object if the name is not a null pointer.
INTU Permits	The number of initial permits available. Maximum is qMAX_PERMITS. This value is specified in qKernel.h

Error conditions

Error	Description
qERR_SEM_ISR	The function is called from an ISR.
qERR_SEM_OVERFLOW	The number of permits is greater than the maximum.
qERR_SEM_NAME_IN_USE	The name is already in use for another object
qERR_SEM_MEMORY	There is no memory available to handle the request.
qERR_KRN_LICENSE	Creating this object violates the license criteria.

151. qSemOpen

```
qtSem qSemOpen (           // Returns NULL when timed-out
    char *pName);         // The Name of the semaphore
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing semaphore object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended. The timeout specifies how long the thread is willing to wait.

The function needs memory for its operation, that's being freed when qSemCreate() activates the thread. It tries to allocate memory from the variable pool or from the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtSEM	The function returns a pointer to the semaphore object. If the pointer is a null pointer the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_SEM_ISR	The function is called from an ISR.
qERR_SEM_NO_START	The function is called before Q-Kernel is started.
qERR_SEM_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_SEM_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_SEM_NO_NAME	Semaphores without a name can't be opened.
qERR_SEM_CRITICAL	This function cannot be called from within a critical section.
qERR_SEM_MEMORY	There is no memory available to handle the open request. All open functions return the same error.

152. qSemOpenNB

```
qtSEM qSemOpenNB (           //
    char *pName) ;           // The Name of the EventSet
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns a pointer to an existing semaphore object.

Parameters and return value

Parameter	Description
Returns qtSEM	The function returns a pointer to the semaphore object. The function returns NULL if the object does not exist.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_SEM_ISR	The function is called from an ISR.
qERR_SEM_NO_NAME	Semaphores without a name can't be opened.

153. qSemOpenTO

```
qtSEM qSemOpenTO (           // Returns NULL when timed-out
    char *pName,             // The Name of the object
    INT32S TimeOut);         // The timeout
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended. The timeout specifies how long the thread is willing to wait.

The function needs memory for its operation, that's being freed when qSemCreate() activates the thread. It tries to allocate memory from the variable pool or from the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtSEM	The function returns a pointer to the semaphore object. If the pointer is null the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_SEM_ISR	The function is called from an ISR.
qERR_SEM_NO_START	The function is called before Q-Kernel is started.
qERR_SEM_NO_NAME	Objects without a name can't be opened.
qERR_SEM_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_SEM_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_SEM_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.
qERR_SEM_CRITICAL	This function cannot be called from within a critical section.
qERR_SEM_MEMORY	There is no memory available to handle the open request.

154. qSemPermits

```
INTU qSemPermits (
    qtSEM pSem);    // The semaphore to get permits for
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

The function returns the number of permits available in this semaphore. This method is typically used for debugging and testing purposes.

Parameters and return value

Parameter	Description
Returns INTU	The function returns the number of permits available.
qtSEM pSem	A pointer to the semaphore object. Must be returned from qSemCreate() or qSemOpen() with the correct name.

Error conditions

Error	Description
qERR_SEM_ISR	The function is called from an ISR.
qERR_SEM_ID	The object is not a semaphore object, has not been created or points to no object at all.

155. qSemRelease

```
void qSemRelease (
    qtSEM pSem);           // The semaphore that contains
                          // the permit to release
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

The function releases a permit and increasing the number of available permits by one. If any threads are trying to acquire a permit, then the one with highest priority is selected and given the permit that was just released. That thread is enabled for scheduling purposes.

There is no requirement that a thread or fiber that releases a permit must have acquired that permit. Correct usage of a semaphore is established by programming convention in the application.

Permits can be released from Interrupt Service Routines (ISR)

Parameters and return value

Parameter	Description
qtSEM pSem	A pointer to the semaphore object. Must be returned from qSemCreate() or qSemOpen() with the correct name.

Error conditions

Error	Description
qERR_SEM_ID	The object is not a semaphore object, has not been created or points to no object at all.
qERR_SEM_OVERFLOW	The semaphore count is beyond $2^{15}-1$ for 16 bit systems and $2^{31}-1$ for 32 bit versions.

156. qSemReleaseISR

```
void qSemReleaseISR(
    qtSEM pSem);           // The semaphore that contains
                          // the permit to release
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

The function releases a permit and increasing the number of available permits by one. If any threads are trying to acquire a permit, then the one with highest priority is selected and given the permit that was just released. That thread is enabled for scheduling purposes.

There is no requirement that a thread or fiber that releases a permit must have acquired that permit. Correct usage of a semaphore is established by programming convention in the application.

This function must be called from an ISR and the system will evaluate if it can execute the request immediately. If that's not possible it will spawn the request and will execute it if all interrupt requests are serviced.

Permits can be released from Interrupt Service Routines (ISR)

Parameters and return value

Parameter	Description
qtSEM pSem	A pointer to the semaphore object. Must be returned from qSemCreate() or qSemOpen() with the correct name.

Error conditions

Error	Description
qERR_SEM_ID	The object is not a semaphore object, has not been created or points to no object at all.
qERR_SEM_OVERFLOW	The semaphore count is beyond 2 ¹⁵ -1 for 16 bit systems and 2 ³¹ -1 for 32 bit versions.
qERR_SEM_ISR	The function is called from an ISR.

157. qThrClose

```
void qThrClose(
    qtTCB pTcb)           // The thread to close
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Closes a thread and returns the resources back to the resource pool. It is the developer responsibility to make sure that the thread does not wait for any object and is not referenced by other threads. This function is automatically called if a thread ends its function.

The function executes the following steps:

- De-allocates the TCB and the thread stack
- Remove the TCB from the Ready or wait list

It's the developer responsible to check that the thread is not used.

Parameters and return value

Parameter	Description
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_LWT	The function tries to close the idle thread.
qERR_THR_ID	The thread control block is not a TCB, has not been created or points to no TCB at all.
qERR_THR_IN_USE	The event object is in use by other threads. More specific other thread(s) are waiting for it. The system can't detect if other threads are using this event.

158. qThrCreate

```
qtTCB qThrCreate(  
    char *pName,           // The Name of the thread  
    void(*pFun)(void *pPar), // The thread function itself  
    void *pPar,           // The parameter data  
    INTU StackSize,       // The size of the stack  
    INT8U Priority);       // The priority
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function will create a thread to be management by **Q-Kernel**. When the system is not yet started it will create the memory structures and other control information. When **Q-Kernel** is running it will do the same but it will make the thread ready to run and the thread will run if it becomes the highest priority thread. If other threads are waiting for this thread to be created those thread(s) are made run-able. The function executes the following steps:

- Allocates the TCB and the thread stack
- Populate the TCB
- Add the TCB to the ready list
- Reschedules if this thread becomes the highest priority thread
- Returns the pointer to the TCB

There is also a function available to create threads in a suspended mode. See `qThrCreateSuspended()`.

Parameters

Parameter	Description
Returns qtTCB	The function returns a pointer to the Thread Control Block of the created thread.
char *pName	The name of the thread. The name must be unique within other thread objects or qNO_NAME which is a null pointer. qThrOpen() can be used to locate the Thread if the name is not a null pointer.
void(*pFun)(void *pPar)	The function that contains the code of the thread.
void *pPar	The parameter that is provided to the thread. This can be a pointer or any other type that fits in the pointer.
INTU StackSize	The size of the stack. This memory is allocated from variable memory.
INT8U Priority	The priority from 1 to 250. The higher the number the higher the priority.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_STACK	The stack size is smaller than 16 bytes.
qERR_THR_PRIO	The thread priority is 0 or greater than 250
qERR_THR_NAME_IN_USE	A thread with that name already exists
qERR_THR_MEMORY	There is no memory available to handle the request.
qERR_KRN_LICENSE	Creating this object violates the license criteria.

The developer must give every thread a unique name or no name at all.

159. qThrCreateSuspended

```

qtTCB qThrCreate(
    char *pName,           // The Name of the thread
    void(*pFun)(void *pPar), // The thread function itself
    void *pPar,           // The parameter data
    INTU StackSize,       // The size of the stack
    INT8U Priority);      // The priority

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function will create a thread suspended to be management by *Q-Kernel*. When the system is not yet started it will create the memory structures and other control information. The function executes the following steps:

- Allocates the TCB and the thread stack
- Populate the TCB
- Add the TCB to the hibernate list
- Returns the pointer to the TCB

Parameters

Parameter	Description
Returns qtTCB	The function returns a pointer to the Thread Control Block of the created thread.
char *pName	The name of the thread. The name must be unique within other thread objects or qNO_NAME which is a null pointer. qThrOpen() can be used to locate the Thread if the name is not a null pointer.
void(*pFun)(void *pPar)	The function that contains the code of the thread.
void *pPar	The parameter that is provided to the thread. This can be a pointer or any other type that fits in the pointer.
INTU StackSize	The size of the stack. This memory is allocated from variable memory.
INT8U Priority	The priority from 1 to 250. The higher the number the higher the priority.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_STACK	The stack size is smaller than 16 bytes.
qERR_THR_PRIO	The thread priority is 0 or greater than 250
qERR_THR_NAME_IN_USE	A thread with that name already exists
qERR_THR_MEMORY	There is no memory available to handle the request.
qERR_KRN_LICENSE	The thread that is created violates the license criteria.

The developer must give every thread a unique name or no name at all.

160. qThrCurrent

```
qtTCB qThrCurrent(); // Return the current TCB
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function will return a pointer to the current TCB.

Parameters

Parameter	Description
Returns qtTCB	The function returns a pointer to the Thread Control Block of the current thread.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.

161. qThrEvtClear

```

INTU qThrEvtClear (
    qtTCB pTcb,           // The thread event to clear
    INTU EventFlags);    // The flags to clear

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Clears one or more event flags in the thread event set. The function returns the events flags before the clear.

This is also the mechanism to get the event flags without changing the event flags. See the example below:

```

INTU flags;              // variable to return the flags
qtTCB p;                // Set by create or open
flags = qThrEvtClear(p,0) // Null does not clear anything. It
...                     // now just returns the event flags

```

Parameters and return value

Parameter	Description
Returns INTU	Returns the event flags before the clear operation.
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.
INTU EventFlags	The event flags to clear. A value of zero does not clear anything.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_ID	The object is not a thread object, has not been created or points to no object at all.

162. qThrEvtSignal

```
void qThrEvtSignal(
    qtTHR pThr,           // The thread event set to signal
    INTU EventFlags);    // The flags to set
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Set one or more event flags in the thread event set. The thread can use this function to set its own event set flags. After the flags are set the thread is evaluated to see if they match the wait criteria. If it matches the wait criteria and it will get the ready state and if it has the highest priority it will be selected to run.

Parameters and return value

Parameter	Description
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.
INTU EventFlags	The event flags to set. At least one flag should be set.

Error conditions

Error	Description
qERR_THR_ID	The object is not a thread object, has not been created or points to no object at all.
qERR_THR_NO_FLAGS	Not one flag in EventFlags is set

163. qThrEvtSignalISR

```
void qThrEvtSignalISR(
    qtTHR pThr,           // The thread event set to signal
    INTU EventFlags);    // The flags to set
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

Set one or more event flags in the thread event set. The thread can use this function to set its own event set flags. After the flags are set the thread is evaluated to see if they match the wait criteria. If it matches the wait criteria and it will get the ready state and if it has the highest priority it will be selected to run.

This function must be called from an ISR and the system will evaluate if it can execute the request immediately. If that's not possible it will spawn the request and will execute it if all interrupt requests are serviced.

Parameters and return value

Parameter	Description
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.
INTU EventFlags	The event flags to set. At least one flag should be set.

Error conditions

Error	Description
qERR_THR_ID	The object is not a thread object, has not been created or points to no object at all.
qERR_THR_NO_FLAGS	Not one flag in EventFlags is set
qERR_THR_ISR	The function is not called from an ISR.

164. qThrEvtWait

```
INTU qThrEvtWait(
    INTU EventFlags,           // The flags to wait for
    INT8U WaitType);         // The type of wait (see
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

Wait for a specific set of thread event flags to be set. The required flags are specified in EventFlags. The function can wait for all specified flags set or any of the flags set. The developer can specify if the flags need to be cleared if the wait is over.

There is also a non-blocking version of this function. The non-blocking function is faster. Use this function if the size of the code is a concern and this function is already used in another place in the application because using both functions doubles the flash footprint.

Parameters and return value

Parameter	Description
Returns INTU	Returns 0 if the function timed out or the events flags that waked-up the thread before the optional clear if the function is successful.
INTU EventFlags	The event flags to wait for. At least one flag must be set.
INT8U WaitType	<p>The type of wait. The following are defined.</p> <p>qWAIT_ALL means wait until all flags are set. This is also called the AND scenario.</p> <p>qWAIT_ALL_CLEAR means wait until all flags are set and if this situation occurs reset the flags that the thread was waiting for.</p> <p>qWAIT_ANY means wait until one of the flags is set. This is also called the OR scenario.</p> <p>qWAIT_ANY_CLEAR means wait until one of the flags is set and if this situation occurs reset the flags that triggered this operation. So not all flags that the thread was waiting for are reset.</p>

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_NO_START	The function is called before Q-Kernel is started.
qERR_THR_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_THR_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_THR_NO_FLAGS	No flags in the EventFlags parameter is set
qERR_THR_WAIT_TYPE	The event type is incorrect
qERR_THR_CRITICAL	This function cannot be called from within a critical section.

165. qThrEvtWaitNB

```

INTU qThrEvtWaitNB(
    INTU EventFlags,           // The flags to wait for
    INT8U WaitType);         // The type of wait (see below)

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function will check if a specific set of thread event flags are set. The required flags are specified in EventFlags. The function returns immediately. The developer can specify if the flags need to be cleared if the wait is over.

This is the faster non-blocking version of qThrEvtWait(). Use qThrEvtWait() if the size of the code is a concern and qThrEvtWait() is already used in another place in the application because using both functions doubles the flash footprint.

Parameters and return value

Parameter	Description
Returns INTU	Returns 0 if the function timed out or the events flags that waked-up the thread before the optional clear if the function is successful.
INTU EventFlags	The event flags to wait for. At least one flag must be set.
INT8U WaitType	<p>The type of wait. The following are defined.</p> <p>qWAIT_ALL means wait until all flags are set. This is also called the AND scenario.</p> <p>qWAIT_ALL_CLEAR means wait until all flags are set and if this situation occurs reset the flags that the thread was waiting for.</p> <p>qWAIT_ANY means wait until one of the flags is set. This is also called the OR scenario.</p> <p>qWAIT_ANY_CLEAR means wait until one of the flags is set and if this situation occurs reset the flags that triggered this operation. So not all flags that the thread was waiting for are reset.</p>

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_NO_START	The function is called before Q-Kernel is started.
qERR_THR_NO_FLAGS	No flags in the EventFlags parameter is set
qERR_THR_WAIT_TYPE	The event type is incorrect

166. qThrEvtWaitTO

```

INTU qThrEvtWaitTO(
    INTU EventFlags,      // The flags to wait for
    INT8U WaitType,      // The type of wait (see below)
    INT32S TimeOut);     // The time out

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

Wait for a specific set of thread event flags to be set. The required flags are specified in EventFlags. The function can wait for all specified flags set or any of the flags set. The developer can specify if the flags need to be cleared if the wait is over.

There is also a non-blocking version of this function. The non-blocking function is faster. Use this function if the size of the code is a concern and this function is already used in another place in the application because using both functions doubles the flash footprint.

Parameters and return value

Parameter	Description
Returns INTU	Returns 0 if the function timed out or the events flags that waked-up the thread before the optional clear if the function is successful.
INTU EventFlags	The event flags to wait for. At least one flag must be set.
INT8U WaitType	<p>The type of wait. The following are defined.</p> <p>qWAIT_ALL means wait until all flags are set. This is also called the AND scenario.</p> <p>qWAIT_ALL_CLEAR means wait until all flags are set and if this situation occurs reset the flags that the thread was waiting for.</p> <p>qWAIT_ANY means wait until one of the flags is set. This is also called the OR scenario.</p> <p>qWAIT_ANY_CLEAR means wait until one of the flags is set and if this situation occurs reset the flags that triggered this operation. So not all flags that the thread was waiting for are reset.</p>

Parameter	Description
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_NO_START	The function is called before Q-Kernel is started.
qERR_THR_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_THR_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_THR_NO_FLAGS	No flags in the EventFlags parameter is set
qERR_EVT_WAIT_TYPE	The event type is incorrect
qERR_THR_CRITICAL	This function cannot be called from within a critical section.
qERR_THR_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_THR_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_THR_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.

167. qThrOpen

```
qtTCB qThrOpen (           // Returns NULL when timed-out
    char *pName);         // The Name of the thread
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing thread. If the thread is created before this function is executed the function returns immediately. If the thread with that name does not exist the thread is suspended.

The function needs memory for its operation, that's being freed when qThrCreate() activates the thread. It tries to allocate memory from the variable memory pool or the heap.

If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtTCB	The function returns a pointer to the thread control block. If the pointer is a null pointer the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_NO_START	The function is called before Q-Kernel is started.
qERR_THR_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_THR_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_THR_NO_NAME	Threads without a name can't be opened.
qERR_THR_CRITICAL	This function cannot be called from within a critical section.
qERR_THR_MEMORY	There is no memory available to handle the open request. All open functions return the same error.

168. qThrOpenNB

```
qtTCB qThrOpenNB (           //
    char *pName);           // The name of the thread
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns a pointer to an existing thread object.

Parameters and return value

Parameter	Description
Returns qtTCB	The function returns a pointer to the thread object. The function returns NULL if the object does not exist.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_NO_NAME	Threads without a name can't be opened.

169. qThrOpenTO

```
qtTCB qThrOpenTO(           // Returns NULL when timed-out
    char *pName,           // The Name of the object
    INT32S TimeOut);       // The timeout
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended. The timeout specifies how long the thread is willing to wait.

The function needs memory for its operation, that's being freed when qThrCreate() activates the thread. It tries to allocate memory from the variable pool or from the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtTCB	The function returns a pointer to the thread object. If the pointer is null the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_NO_START	The function is called before Q-Kernel is started.
qERR_THR_NO_NAME	Objects without a name can't be opened.
qERR_THR_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_THR_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_THR_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.
qERR_THR_CRITICAL	This function cannot be called from within a critical section.
qERR_THR_MEMORY	There is no memory available to handle the open request.

170. qThrResume

```
void qThrResume (      // Wake-up a thread
    qtTCB pTcb);      // The TCB of the thread to wake-up
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function resumes a thread that sleeps. If the thread is not sleeping this function has no effect and does not set an error condition. The function can be called from a thread, ISR or fiber.

Parameters and return value

Parameter	Description
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.

Error conditions

Error	Description
qERR_THR_ID	The thread control block is not a TCB, has not been created or points to no TCB at all.

171. qThrResumeISR

```
void qThrResumeISR( // Wake-up a thread
    qtTCB pTcb); // The TCB of the thread to wake-up
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function resumes a thread that sleeps. If the thread is not sleeping this function has no effect and does not set an error condition.

This function must be called from an ISR and the system will evaluate if it can execute the request immediately. If that's not possible it will spawn the request and will execute it if all interrupt requests are serviced.

Parameters and return value

Parameter	Description
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.

Error conditions

Error	Description
qERR_THR_ID	The thread control block is not a TCB, has not been created or points to no TCB at all.
qERR_THR_ISR	The function is called from an ISR.

172. qThrSetPriority

```
void qThrSetPriority( // Set the priority for a thread
qtTCB pTcb, // The TCB of the thread
INT8U Priority); // The new priority
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function set the priority for a thread. The thread can set its own priority by specifying qThrCurrent() in the pTCB argument.

Parameters and return value

Parameter	Description
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.
INT8U Priority	The priority of the thread. Must be between 1 and 250 inclusive.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_LWT	The function tries to change the priority of the Idle thread.
qERR_THR_ID	The thread control block is not a TCB, has not been created or points to no TCB at all.
qERR_THR_PRIO	The priority is out of range.

173. qThrSleep

```
void qThrSleep(           // Let the thread sleep
  INT32S Time);          // Sleeptime in milliseconds
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Always preemption
-----------------	--------	-----------------------	-------	-----	-------------------

Description

This function lets the current thread sleep for a specified time.

Parameters and return value

Parameter	Description
INT32S Time	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_NO_START	The function is called before Q-Kernel is started.
qERR_THR_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_THR_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_THR_ID	The object is not a queue object, has not been created or points to no object at all.
qERR_THR_NO_RTCC	The specified Timeout requires a RTCC which is not available. (qRTCC=0)

Error	Description
qERR_THR_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_THR_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.
qERR_THR_CRITICAL	The function is called within a critical section.

174. qThrStack

```
INTU qThrStack ( // Returns the stack-size in use
qtTCB pTcb); // The TCB of the thread
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function returns the maximum number of bytes that has been in use on the stack until now.

The function is not deterministic and should not be used in production systems. This function is normally used in debug sessions.

Parameters and return value

Parameter	Description
Returns INTU	This function returns the maximum number of bytes that has been in use on the stack until now.
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_ID	The thread control block is not a TCB, has not been created or points to no TCB at all.

175. qThrStatLapCycles

```
INT32U qThrStatLapCycles (
    qtTCB pTcb);           // The TCB of the thread
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns the number of cycles for all fibers and the scheduler during the last lap.

Parameters

Parameter	Description
Returns INT32U	Returns the number of cycles
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_ID	The thread control block is not a TCB, has not been created or points to no TCB at all.
qERR_THR_STAT_OFF	Statistics is not running

176. qThrStatLapPerc

```
INTU qThrStatLapPerc (
    qtTCB pTcb);    // The TCB of the thread
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function returns the percentage of CPU time during the last "lap" for the specified thread.

Parameters

Parameter	Description
Returns INTU	The percentage is specified as a value * 100. So 12.34% is returned as 1234.
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_ID	The thread control block is not a TCB, has not been created or points to no TCB at all.
qERR_THR_STAT_OFF	Statistics is not running

177. qThrStatTotalCycles

```
INT64U qThrStatTotalCycles (
    qtTCB pTcb);           // The TCB of the thread
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns the total number of cycles since the start of the statistic gathering for the specified thread.

Parameters

Parameter	Description
Returns INT64U	The number of cycles
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_ID	The thread control block is not a TCB, has not been created or points to no TCB at all.
qERR_THR_STAT_OFF	Statistics is not running

178. qThrTrack

```
void qThrTrack (
    qtTCB pTcb,           // The TCB of the thread
    INTU *pAddr,         // The address of the bit to set/clr
    INTU BitNbr);       // The bit number
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function enables or disables tracking for threads. The address and the bit-number specify which bit to set or clear. If the address points to is one of the I/O ports the developer is responsible for setting the TRIS for that port.

Tracking need to be configured for this function to work.

Parameters and return value

Parameter	Description
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.
INTU *pAddr	The address of the bit to set or to clear. A value of 0 disables the tracking. If tracking is disabled it will execute the same number of cycles as enabled.
INTU BitNbr	The bit-number. Valid values are 0 to 15 for the 16 bit version and 0 to 31 for the 32 bit version.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR
qERR_THR_BIT_NBR	Bit number is invalid

179. qThrSuspend

```
void qThrSuspend( // Returns success or failure
qtTCB pTcb); // The TCB of the thread
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns suspend a waiting or ready thread. The thread can only be activated by the qThrResume() function.

Parameters and return value

Parameter	Description
qtTCB pTcb	A pointer to the thread control block. Must be returned from qThrCreate() or from qThrOpen() with the correct name. Use the function qThrCurrent() to provide the thread control block of the current thread.

Error conditions

Error	Description
qERR_THR_ID	The thread control block is not a TCB, has not been created or points to no TCB at all.
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_IDLE	The function tries to suspend the idle thread.

180. qThrYield

```
void qThrYield(); // Yield the thread
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
--------------	--------	--------------------	-------	-----	---------------------

Description

This function yields a thread. This function only as meaning is there are threads with the same priority. The developer can implement corporative scheduling with this function. This is not recommended. Other mechanisms like fibers are in most cases more suitable.

Parameters and return value

This function does not require any parameters or return values.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_NO_START	The function is called before Q-Kernel is started.
qERR_THR_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_THR_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.

181. qTmrClose

```
void qTmrClose(
    qtTMR pTmr, // The timer to close
    qtTMR_SIGNAL SignalOptions); // The signal options
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

Stops and closes a timer and returns the resources back to the resource pool. It is the developer responsibility to make sure that the timer is not used. The system will invalidate the object so other function can't use the object accidentally.

If the timer is stopped with the option `qTMR_SIGNAL_EXPIRE` the function is called from the thread or fiber that executes this function. This means that the function does not always runs in the context of a fiber. If that's the case then the parameter of the function contains the TCB of the thread.

Parameters and return value

Parameter	Description
qtTMR pTmr	A pointer to the object. Must be returned from the <code>qTmrCreate()</code> or <code>qTmrOpen()</code> function with the correct name.
qtTMR_SIGNAL SignalOptions	<ul style="list-style-type: none"> • <code>qTMR_SIGNAL_NOT</code> will not signal the timer function but just removes the timer for the system. • <code>qTMR_SIGNAL_NOW</code> will signal immediately, before the due time, and then removes the timer for the system. (The function is called with the parameter set to 0) • <code>qTMR_SIGNAL_EXPIRE</code> will signal at the normal time and then removes the timer for the system if the timer is not stopped. If the timer is stopped it will be removed immediately.

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_ID	The object is not a timer object, has not been created or points to no object at all.
qERR_TMR_SIGNAL_OPT	The SignalOptions are invalid. Please use one of the options as described above.
qERR_TMR_FBR	The function is called from a fiber with the option qTMR_SIGNAL_NOW. This option can only be used from a thread. Fibers includes all functions called from fibers like qNtfSwitch() or expired timer and alarm functions.

182. qTmrCreate

```

qtTMR qTmrCreate (
    char *pName,           // The Name of the timer
    void (*pFbr)(void *p), // The function to call
    void *pParam,         // The parameter when called
    qtTMR_TYPE TimerType, // Timer options
    INT32S Time);         // The number of seconds
                          // or cycles before the

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
--------------	--------	--------------------	-------	-----	---------------------

Description

Before a timer can be used, it has to be created by calling this function. The creating thread or fiber specifies the name for the timer object, the fiber function, timer type and the time the timer expires. If there is an open request for the timer with this name that thread will be readied, which creates a possible preemption. Multiple threads can wait for the object to be created and all threads will be readied.

The function needs memory for its operation, that's being freed when qTmrClose() end the use of the timer. The function tries to allocate memory from the variable pool or from the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtTMR	The function returns a pointer to the timer object.
char *pName	The name of the timer. The name must be unique within other timer objects or qNO_NAME which is a null pointer. qTmrOpen() can be used to locate the object if the name is not a null pointer.
void (*pFbr)(void *p)	The function to call after the time expires
void *pParam	The parameter for the function. This allows to share the function for several timers

Parameter	Description
qtTMR_TYPE TimerType	<p>The timer type options.</p> <ul style="list-style-type: none"> • qtTMR_TYPE_PERIODIC specifies a periodic timer. • qtTMR_TYPE_ONE_SHOT specifies a one shot timer. • qtTMR_TYPE_PERIODIC_MANUAL_START specifies a periodic timer that must be started manual. • qtTMR_TYPE_ONE_SHOT_MANUAL_START specifies a one shot timer that must be started manual.
INT32S Time	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_NAME_IN_USE	The name is already in use for another timer object
qERR_TMR_TYPE	The type is not valid
qERR_TMR_FUNCTION	No function specified.
qERR_TMR_TIME	The time is incorrect.
qERR_TMR_MEMORY	There is no memory available to handle the request.
qERR_KRN_LICENSE	Creating this object violates the license criteria.

183. qTmrOpen

```
qtTMR qTmrOpen (           // Returns NULL when timed-out
    char *pName);         // The Name of the timer
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
-----------------	--------	-----------------------	-------	-----	---------------------

Description

This function returns a pointer to an existing timer object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended.

The function needs memory for its operation, that's being freed when qTmrCreate() activates the thread. It tries to allocate memory from the variable pool or from the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtTMR	The function returns a pointer to the timer object. If the pointer is a null pointer the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened..

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_NO_START	The function is called before Q-Kernel is started.
qERR_TMR_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.
qERR_TMR_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.
qERR_TMR_NO_NAME	Timers without a name can't be opened.

Error	Description
qERR_TMR_CRITICAL	This function cannot be called from within a critical section.
qERR_TMR_MEMORY	There is no memory available to handle the open request. All open functions return the same error.

184. qTmrOpenNB

```
qtTMR qTmrOpenNB (           //
    char *pName);           // The Name of the timer
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns a pointer to an existing timer object.

Parameters and return value

Parameter	Description
Returns qtTMR	The function returns a pointer to the timer object. The function returns NULL if the object does not exist.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_NO_NAME	Timers without a name can't be opened.

185. qTmrOpenTO

```

qtTMR qTmrOpenTO(           // Returns NULL when timed-out
    char *pName,           // The Name of the object
    INT32S TimeOut);       // The timeout

```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------------

Description

This function returns a pointer to an existing object. If the object is created before this function is executed the function returns immediately. If the object with that name does not exist the thread is suspended. The timeout specifies how long the thread is willing to wait.

The function needs memory for its operation, that's being freed when qTmrCreate() activates the thread. It tries to allocate memory from the variable pool or from the heap. If this fails the system throws the error to indicate failure. To prevent this error it is a good practice to create enough objects during initialization.

Parameters and return value

Parameter	Description
Returns qtTMR	The function returns a pointer to the timer object. If the pointer is null the function timed-out.
char *pName	The name of the object to open. Objects without a name, qNO_NAME or a NULL pointer, cannot be opened.
INT32S TimeOut	<p>A positive timeout value specifies a short wait-time in cycles. We advise to use the macros qUSEC() and qMSEC() so the value is independent of the system clock. The minimum is 1 μSecond.</p> <p>A negative value specifies the real-time wait time in seconds. The macro's qSEC() and qDTM() are available for specifying the wait-time in seconds and for specific times based on year, month, day, hour, minute and second.</p>

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_NO_START	The function is called before Q-Kernel is started.
qERR_TMR_NO_NAME	Objects without a name can't be opened.
qERR_TMR_NO_RTCC	The specified TimeOut requires a RTCC which is not available. (qRTCC=0)
qERR_TMR_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)
qERR_TMR_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.
qERR_TMR_CRITICAL	This function cannot be called from within a critical section.
qERR_TMR_MEMORY	There is no memory available to handle the open request.

186. qTmrStart

```
void qTmrStart(
    qtTMR pTmr);           // The timer to start
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function starts a timer after it is stopped. If the timer is still running it just calculates the new expire time and activates the timer.

This function is normally used to start a one-shot timer but it can be used to start every stopped timer.

Parameters and return value

Parameter	Description
qtTMR pTmr	A pointer to the timer object. Must be returned from qTmrCreate() or qTmrOpen() with the correct name.

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_ID	The object is not a timer object, has not been created or points to no object at all.

187. qTmrStartISR

```
void qTmrStartISR(
    qtTMR pTmr);           // The timer to start
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function starts a timer after it is stopped. If the timer is still running it just calculates the new expire time and activates the timer.

This function is normally used to start a one-shot timer but it can be used to start every stopped timer.

Parameters and return value

Parameter	Description
qtTMR pTmr	A pointer to the timer object. Must be returned from qTmrCreate() or qTmrOpen() with the correct name.

Error conditions

Error	Description
qERR_TMR_ISR	The function is not called from an ISR.
qERR_TMR_ID	The object is not a timer object, has not been created or points to no object at all.

188. qTmrStop

```
void qTmrStop(
    qtTMR pTmr, // The timer to stop
    qtTMR_SIGNAL SignalOptions); // Signal options
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	Possible preemption
--------------	--------	--------------------	-------	-----	---------------------

Description

This function stops a timer. There are several stop options to control the signaling.

If the timer is stopped with the option `qTMR_SIGNAL_EXPIRE` the function is called from the thread or fiber that executes this function. This means that the function does not always runs in the context of a fiber. If that's the case then the parameter of the function contains the TCB of the thread.

Parameters and return value

Parameter	Description
qtTMR qTmr	A pointer to the timer object. Must be returned from <code>qTmrCreate()</code> or <code>qTmrOpen()</code> with the correct name.
qtTMR_SIGNAL SignalOptions	<ul style="list-style-type: none"> • <code>qTMR_SIGNAL_NOT</code> specifies that the timer is stopped immediately without signaling. • <code>qTMR_SIGNAL_NOW</code> will signal immediately, before the due time, and then removes the timer for the system. (The function is called with the parameter set to 0) • <code>qTMR_SIGNAL_EXPIRE</code> stops the timer after it expires. This means more or less that the timer becomes a one-shot timer.
INTU Time	The time for the timer to expire and call the fiber.

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_ID	The object is not a timer object, has not been created or points to no object at all.
qERR_TMR_SIGNAL_OPT	The SignalOptions are not valid
qERR_TMR_FBR	The function is called from a fiber with the option qTMR_SIGNAL_NOW. This option can only be used from a thread. Fibers includes all functions called from fibers like qNtfSwitch() or expired timer and alarm functions.
qERR_TMR_LWT	The function is called from a lightweight thread with the option qTMR_SIGNAL_NOW. This option can only be used from a thread.

189. qTmrStopISR

```
void qTmrStopISR(
    qtTMR pTmr, // The timer to stop
    qtTMR_SIGNAL SignalOptions); // Signal options
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function stops a timer from an interrupt. There are several stop options to control the signaling. The function is deferred and runs always in a fiber.

Parameters and return value

Parameter	Description
qtTMR qTmr	A pointer to the timer object. Must be returned from qTmrCreate() or qTmrOpen() with the correct name.
qtTMR_SIGNAL SignalOptions	<ul style="list-style-type: none"> qtTMR_SIGNAL_NOT specifies that the timer is stopped immediately without signaling. qtTMR_SIGNAL_NOW will signal immediately, before the due time, and then removes the timer for the system. (The function is called with the parameter set to 0) qtTMR_SIGNAL_EXPIRE stops the timer after it expires. This means more or less that the timer becomes a one-shot timer.

Error conditions

Error	Description
qERR_TMR_ISR	The function is not called from an ISR.
qERR_TMR_ID	The object is not a timer object, has not been created or points to no object at all.
qERR_TMR_SIGNAL_OPT	The SignalOptions are not valid
qERR_TMR_FBR	The function is called from a fiber with the option qtTMR_SIGNAL_NOW. This option can only be used from a thread. Fibers includes all functions called from fibers like qNtfSwitch() or expired timer and alarm functions.

190. qTmrUsec

```
INT64U qTmrUsec() ;// Returns system up-time in  $\mu$ Sec
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns the number of μ Sec that the system is up. This function is the most accurate of the μ Second functions. It overflows after 584,000 years so it has a large range. The disadvantage is that it is an expensive function in size and speed. The value is derived from the **Q-Kernel** timer and managed by the statistics code. See the user guide for more information.

Alternatives are qTmrUsecDiv1(), qTmrUsecDiv256() and qTmrUsecDiv64K(). These functions return a 32 bit number but have a limited range.

Parameters and return value

Parameter	Description
Returns INT64U	The number of μ Sec the system is up

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_NO_USEC	The code that keeps track of μ Seconds is not enabled. Use qKrnUsecOn() to start the μ Second tracking.

191. qTmrUsecDiv1

```
INT32U qTmrUsecDiv1(); // Returns system up-time in
                       // μSec/1 in 32 bit
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns the number of μSec that the system is up. It overflows in about 1 hour and the granularity is 1 μSec .

Alternatives are qTmrUsec(), qTmrUsecDiv256() and qTmrUsecDiv64K().

Parameters and return value

Parameter	Description
Returns INT32U	The number of μSec the system is up

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_NO_USEC	The code that keeps track of $\mu\text{Seconds}$ is not enabled. Use qKrnUsecOn() to start the μSecond tracking.

192. qTmrUsecDiv256

```
INT32U qTmrUsecDiv256(); // Returns system up-time in
                        // μSec/256 in 32 bit
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
---------------------	---------------	---------------------------	--------------	------------	----------------------

Description

This function returns the number of $\mu\text{Sec}/(256)$ that the system is up. It overflows in about 12 days and the granularity is 256 μSec .

Alternatives are qTmrUsec(), qTmrUsecDiv1 () and qTmrUsecDiv64K().

This function is not available in all versions of **Q-Kernel**. Please refer to the User Guide for more information.

Parameters and return value

Parameter	Description
Returns INT32U	The number of $\mu\text{Sec}/256$ the system is up

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_NO_USEC	The code that keeps track of $\mu\text{Seconds}$ is not enabled. Use qKrnUsecOn() to start the μSecond tracking.

193. qTmrUsecDiv64k

```
INT32U qTmrUsecDiv64k(); // Returns system up-time in
                        // μSec/(2^16) in 32 bit
```

Before Start	Thread	Lightweight Thread	Fiber	ISR	No preemption
--------------	--------	--------------------	-------	-----	---------------

Description

This function returns the number of $\mu\text{Sec}/(2^{16})$ that the system is up. It overflows in about 9 years and the granularity is ~ 65 millisecond.

Alternatives are `qTmrUsec()`, `qTmrUsecDiv1()` and `qTmrUsecDiv256()`.

This function is not available in all versions of **Q-Kernel**. Please refer to the User Guide for more information.

Parameters and return value

Parameter	Description
Returns INT32U	The number of $\mu\text{Sec}/(2^{16})$ the system is up

Error conditions

Error	Description
qERR_TMR_ISR	The function is called from an ISR.
qERR_TMR_NO_USEC	The code that keeps track of $\mu\text{Seconds}$ is not enabled. Use <code>qKrnUsecOn()</code> to start the μSecond tracking.

194. Errors

Event errors

Error	Description	Error#
qERR_EVT_ID	The object is not an event object, has not been created or points to no object at all.	0x1100
qERR_EVT_NO_START	The function is called before Q-Kernel is started.	0x1101
qERR_EVT_ISR	The function is called from an ISR.	0x1102
qERR_EVT_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.	0x1103
qERR_EVT_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.	0x1104
qERR_EVT_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.	0x1105
qERR_EVT_MEMORY	There is no memory available to handle the open request.	0x1106
qERR_EVT_NAME_IN_USE	The name is already in use for another object.	0x1107
qERR_EVT_NO_NAME	Events without a name can't be opened.	0x1108
qERR_EVT_CRITICAL	This function cannot be called from within a critical section.	0x1109
qERR_EVT_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x110A
qERR_EVT_NO_RTCC	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x110B
qERR_EVT_IN_USE	One or more threads are waiting on the event. The system can't detect if other thread are using this event. The system will clear the object and use of the same address most likely results in qERR_EVT_ID.	0x1110

Error	Description	Error#
qERR_EVT_NO_FLAGS	Not one flag in EventFlags is set.	0x1111
qERR_EVT_STACK_LIMIT_1		0x1112
qERR_EVT_STACK_LIMIT_2		0x1113
qERR_EVT_WAIT_TYPE	The event type is incorrect.	0x1114

Fiber errors

Error	Description	Error#
qERR_FBR_ISR	The function is called from an ISR.	0x1210
qERR_FBR_PRIO	The priority is smaller than 1 or larger than 4.	0x1211
qERR_FBR_MEMORY	There is no memory available to handle the open request.	0x1212
qERR_FBR_BIT_NBR		0x1213
qERR_FBR_THREAD	The function is called from a thread, a lightweight thread or before the system has started.	0x1214
qERR_FBR_QUEUE_FULL	The queue is full and the entry is no set.	0x1215

Kernel errors

Error	Description	Error#
qERR_KRN_ISR	The function is called from an ISR.	0x1310
qERR_KRN_MEM_SIZE	There is not enough memory available to initialize the system and the defined objects.	0x1311
qERR_KRN_STARTED	The system was already started.	0x1312
qERR_KRN_NO_INIT	The system did not initialize itself. The function qKrnInit() must be called before this function.	0x1313
qERR_KRN_TIMER		0x1314
qERR_KRN_STAT_TIMER		0x1315
qERR_KRN_SAME_TIMER		0x1316
qERR_KRN_INTERRUPT	The kernel interrupt is the same as the kernel timer interrupt.	0x1317
qERR_KRN_NO_STAT	The system is linked without statistics.	0x1318
qERR_KRN_STAT		0x1319
qERR_KRN_LICENSE	Creating this object violates the license criteria.	0x131A
qERR_KRN_MEMORY	There is no variable memory available to handle the request.	0x131B
qERR_KRN_MHZ		0x131C

Lightweight Thread errors

Error	Description	Error#
qERR_LWT_ISR	The function is called from an ISR.	0x1410
qERR_LWT_MEMORY	There is no variable memory available to handle the request.	0x1411
qERR_LWT_PRIORITY	The priority is not between 1 and 250.	0x1412
qERR_LWT_NO_PARENT		0x1413
qERR_LWT_TIME	The thread sleep time is incorrect.	0x1414
qERR_LWT_NO_LWT	The function is called outside a lightweight thread.	0x1415

Message Errors

Error	Description	Error#
qERR_MSG_ID	The message is not a message returned by qMsgAlloc() or qMsgCopy().	0x1510
qERR_MSG_ISR	The function is called from an ISR.	0x1511
qERR_MSG_SIZE	The size is incorrect.	0x1512

Mutex Errors

Error	Description	Error#
qERR_MTX_ID	The object is not a mutex object, has not been created or points to no object at all.	0x1600
qERR_MTX_NO_START	The function is called before Q-Kernel is started.	0x1601
qERR_MTX_ISR	The function is called from an ISR.	0x1602
qERR_MTX_FBR	The function is called from a fiber. Only threads or lightweight threads can own mutexes.	0x1603
qERR_MTX_LWT	This function cannot be called from within a critical section.	0x1604
qERR_MTX_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.	0x1605
qERR_MTX_MEMORY	There is no memory available to handle the open request.	0x1606
qERR_MTX_NAME_IN_USE	The name is already in use for another object.	0x1607
qERR_MTX_NO_NAME	Mutexes without a name can't be opened.	0x1608
qERR_MTX_CRITICAL	This function cannot be called from within a critical section.	0x1609
qERR_MTX_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x160A
qERR_MTX_NO_RTCC	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x160B
qERR_MTX_IN_USE	A thread has locked the mutex so it can't be closed. The system can't detect if other threads are using this mutex.	0x1610
qERR_MTX_LOCKED		0x1611
qERR_MTX_OWNER	The mutex is not unlocked by the owner of the mutex.	0x1612

Pipe Errors

Error	Description	Error#
qERR_PIP_ID	The object is not a pipe object, has not been created or points to no object at all.	0x1700
qERR_PIP_NO_START	The function is called before Q-Kernel is started.	0x1701
qERR_PIP_ISR	The function is called from an ISR.	0x1702
qERR_PIP_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.	0x1703
qERR_PIP_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.	0x1704
qERR_PIP_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.	0x1705
qERR_PIP_MEMORY	There is no memory available to handle the open request.	0x1706
qERR_PIP_NAME_IN_USE	The name is already in use for another object.	0x1707
qERR_PIP_NO_NAME	Pipes without a name can't be opened.	0x1708
qERR_PIP_BLOCK_SIZE	The blocks size is incorrect.	0x1709
qERR_PIP_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x170A
qERR_PIP_NO_RTCC	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x170B
qERR_PIP_MAX_BLOCK_SIZE		0x1710
qERR_PIP_MAX_BLOCKS		0x1711
qERR_PIP_NBR_BLOCKS		0x1712

Queue Errors

Error	Description	Error#
qERR_QUE_ID	The object is not a queue object, has not been created or points to no object at all.	0x1800
qERR_QUE_NO_START	The function is called before Q-Kernel is started.	0x1801
qERR_QUE_ISR	The function is called from an ISR.	0x1802
qERR_QUE_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.	0x1803
qERR_QUE_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.	0x1804
qERR_QUE_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.	0x1805
qERR_QUE_MEMORY	There is no memory available to handle the request.	0x1806
qERR_QUE_NAME_IN_USE	The name is already in use for another object.	0x1807
qERR_QUE_NO_NAME	Message queues without a name can't be opened.	0x1808
qERR_QUE_CRITICAL	This function cannot be called from within a critical section.	0x1809
qERR_QUE_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x180A
qERR_QUE_NO_RTCC	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x180B
qERR_QUE_IN_USE	Other thread(s) are waiting for the queue. The system can't detect if other threads are using the queue.	0x1810
qERR_QUE_NO_FLAGS	No flags in the EventFlags parameter is set.	0x1811
qERR_QUE_SIZE	The size of the queue is incorrect.	0x1812

Semaphore Errors

Error	Description	Error#
qERR_SEM_ID	The object is not a semaphore object, has not been created or points to no object at all.	0x1900
qERR_SEM_NO_START	The function is called before Q-Kernel is started.	0x1901
qERR_SEM_ISR	The function is called from an ISR.	0x1902
qERR_SEM_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.	0x1903
qERR_SEM_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.	0x1904
qERR_SEM_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.	0x1905
qERR_SEM_MEMORY	There is no memory available to handle the request.	0x1906
qERR_SEM_NAME_IN_USE	The name is already in use for another object.	0x1907
qERR_SEM_NO_NAME	Semaphores without a name can't be opened.	0x1908
qERR_SEM_CRITICAL	This function cannot be called from within a critical section.	0x1909
qERR_SEM_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x190A
qERR_SEM_NO_RTCC	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x190B
qERR_SEM_IN_USE	The semaphore object is in use by other threads or fibers. More specific other threads are waiting for it. The system can't detect if other thread or fibers are using this semaphore.	0x1910
qERR_SEM_OVERFLOW	The number of permits is greater than the maximum.	0x1911

Thread Errors

Error	Description	Error#
qERR_THR_ID	The thread control block is not a TCB, has not been created or points to no TCB at all.	0x1A00
qERR_THR_NO_START	The function is called before Q-Kernel is started.	0x1A01
qERR_THR_ISR	The function is called from an ISR.	0x1A02
qERR_THR_FBR	The function is called from a fiber which is not supported because fibers don't support blocking.	0x1A03
qERR_THR_LWT	The function is called from a lightweight thread which is not supported because lightweight threads don't support blocking.	0x1A04
qERR_THR_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.	0x1A05
qERR_THR_MEMORY	There is no memory available to handle the open request. All open functions return the same error.	0x1A06
qERR_THR_NAME_IN_USE	A thread with that name already exists	0x1A07
qERR_THR_NO_NAME	Threads without a name can't be opened.	0x1A08
qERR_THR_CRITICAL	This function cannot be called from within a critical section.	0x1A09
qERR_THR_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x1A0A
qERR_THR_NO_RTCC	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x1A0B
qERR_THR_IN_USE	The event object is in use by other threads. More specific other thread(s) are waiting for it. The system can't detect if other thread are using this event.	0x1A10
qERR_THR_PRIO	The thread priority is 0 or greater than 250	0x1A11
qERR_THR_STACK	The stack size is smaller than 16 bytes.	0x1A12

Error	Description	Error#
qERR_THR_CURRENT		0x1A13
qERR_THR_NO_FLAGS	Not one flag in EventFlags is set.	0x1A14
qERR_THR_WAIT_TYPE	The event type is incorrect.	0x1A15
qERR_THR_STAT_OFF	The system is linked with statistics but statistics are not enabled.	0x1A16
qERR_THR_NO_STAT	The system is linked without statistics.	0x1A17
qERR_THR_WAIT_STATE		0x1A18
qERR_THR_WAIT_OBJECT		0x1A19
qERR_THR_BIT_NBR		0x1A1A

Timer Errors

Error	Description	Error#
qERR_TMR_ID	The object is not a timer object, has not been created or points to no object at all.	0x1B00
qERR_TMR_NO_START	The function is called before Q-Kernel is started.	0x1B01
qERR_TMR_ISR	The function is called from an ISR.	0x1B02
qERR_TMR_FBR	The function is called from a fiber with the option qTMR_SIGNAL_NOW. This option can only be used from a thread. Fibers includes all functions called from fibers like qNtfSwitch() or expired timer and alarm functions.	0x1B03
qERR_TMR_LWT	The function is called from a lightweight thread with the option qTMR_SIGNAL_NOW. This option can only be used from a thread.	0x1B04
qERR_TMR_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.	0x1B05
qERR_TMR_MEMORY	There is no memory available to handle the open request. All open functions return the same error.	0x1B06
qERR_TMR_NAME_IN_USE	The name is already in use for another timer object.	0x1B07
qERR_TMR_NO_NAME	Timers without a name can't be opened.	0x1B08
qERR_TMR_CRITICAL	This function cannot be called from within a critical section.	0x1B09
qERR_TMR_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x1B0A
qERR_TMR_NO_RTCC	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x1B0B
qERR_TMR_IN_USE		0x1B10
qERR_TMR_SIGNAL_OPT	The SignalOptions are invalid.	0x1B11
qERR_TMR_TYPE	The type is not valid.	0x1B12
qERR_TMR_FUNCTION	No function specified.	0x1B13

Error	Description	Error#
qERR_TMR_TIME	The time is incorrect.	0x1B14
qERR_TMR_NO_USEC	The code that keeps track of μ Seconds is not enabled. Use qKrnUsecOn() to start the μ Second tracking.	0x1B15

Real Time Clock Errors

Error	Description	Error#
qERR_RTC_ID	The object is not a timer object, has not been created or points to no object at all.	0x1C10
qERR_RTC_NO_START	The function is called before Q-Kernel is started.	0x1C11
qERR_RTC_ISR	The function is called from an ISR.	0x1C12
qERR_RTC_FBR	The function is called from a fiber with the option qTMR_SIGNAL_NOW. This option can only be used from a thread. Fibers includes all functions called from fibers like qNtfSwitch() or expired timer and alarm functions.	0x1C13
qERR_RTC_LWT	The function is called from a lightweight thread with the option qTMR_SIGNAL_NOW. This option can only be used from a thread.	0x1C14
qERR_RTC_1_1_2010	Dates before 1-JAN-2010 are not allowed.	0x1C15
qERR_RTC_MEMORY	There is no memory available to handle the open request. All open functions return the same error.	0x1C16
qERR_RTC_NAME_IN_USE	The name is already in use for another timer object.	0x1C17
qERR_RTC_NO_NAME	Timers without a name can't be opened.	0x1C18
qERR_RTC_CRITICAL	This function cannot be called from within a critical section.	0x1C19
qERR_RTC_EMULATION		0x1C1A
qERR_RTC_NO_RTCC	This function requires a timer which is not available. (qTIMER=0)	0x1C1B

Publish/subscribe Errors

Error	Description	Error#
qERR_PUB_ID	The object is not a publisher object, has not been created or points to no object at all.	0x1D00
qERR_PUB_NO_START	The function is called before Q-Kernel is started.	0x1D01
qERR_PUB_ISR	The function is called from an ISR.	0x1D02
qERR_PUB_FBR	The function is called from a fiber which is not allowed. Fibers includes all functions called from fibers like qNtfSwitch() or expired timer and alarm functions.	0x1D03
qERR_PUB_LWT	The function is called from a lightweight thread which is not allowed. This option can only be used from a thread.	0x1D04
qERR_PUB_TIMEOUT	The specified TimeOut is smaller than 1 μ Second.	0x1D05
qERR_PUB_MEMORY	There is no memory available to handle the open request. All open functions return the same error.	0x1D06
qERR_PUB_NAME_IN_USE	The name is already in use for another publisher object.	0x1D07
qERR_PUB_NO_NAME	Publishers without a name can't be opened.	0x1D08
qERR_PUB_CRITICAL	This function cannot be called from within a critical section.	0x1D09
qERR_PUB_NO_TIMER	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x1D0A
qERR_PUB_NO_RTCC	The specified TimeOut requires a timer which is not available. (qTIMER=0)	0x1D0B

Memory Errors

Error	Description	Error#
qERR_MEM_ID	The object is not a memory object, has not been created or points to no object at all.	0x1E00
qERR_MEM_ISR	The function is called from an ISR.	0x1E02
qERR_MEM_DAMAGE		0x1E10