Q·Kernel Reference Guide

Version 6.0-3353

Q·Kernel is a product of Quasarsoft Ltd.

License

Q-Kernel-Free is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License (version 3) as published by the Free Software Foundation **and modified by** the QuasarSoft Ltd. exception.

> The QuasarSoft Ltd. EXCEPTION
>
> You may not exercise any of the rights granted to You in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Program for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any

Q-Kernel-Free is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the full license text at the following link <http://www.quasarsoft.com/license.html>

For the purpose of applying the license to this document, I consider "source code" to refer to this document source (.docx) and "object code" to refer to the generated file (.pdf).

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	Critical Section	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 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.

Critical Section

This indicator is one of the following colors:

- **Red** indicates that it is not allowed to call the function when in a critical section.
- **Orange** indicates that the function can be called critical section 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 critical section.

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.
- **Always** Preemption means that the function always preempts if called from a thread.
- **Possible** Preemption means that the function could preempts if called from a thread.

1. qBitClr

```
void qBitClr(
  unsigned* Ptr,    // Address
  unsigned Bit);    // The bit to clear
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function clears the bit in the unsigned integer pointed to by Ptr. This function is an inline function doing the following operation:

*Ptr &= ~(1<<(Bit));

The function is NOT atomic. Don't use qBitClrAtomic() if atomicy is not required.

Parameters and return value

Parameter	Description
unsigned* Ptr	Address
unsigned Bit	Bit Number (0->15 or 0->31)

Error conditions

None

2. qBitClrAtomic

```
void qBitClrAtomic(
  unsigned* Ptr,    // Address
  unsigned Bit);    // The bit to clear
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function clears the bit in the unsigned integer pointed to by Ptr. This function is a specific operation because it is atomic.

Use the function qBitClr() if atomicy is not required.

Parameters and return value

Parameter	Description
unsigned* Ptr	Address
unsigned Bit	Bit Number (0->15 or 0->31)

Error conditions

None

3. qBitMemClrAtomic

```
void qBitMemClrAtomic(
  unsigned BitNbr);    // The bit number to clear
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function clears a bit in special memory. This function is a specific operation because it is atomic and it can only operate on special memory. See the user guide for more information.

Parameters and return value

Parameter	Description
unsigned BitNbr	Bit Number

Error conditions

None

4. qBitMemSetAtomic

```
void qBitMemSetAtomic(
  unsigned BitNbr);    // The bit number to set
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function sets a bit in special memory. This function is a specific operation because it is atomic and it can only operate on special memory. See the user guide for more information.

Parameters and return value

Parameter	Description
unsigned BitNbr	Bit Number

Error conditions

None

5. qBitMemTglAtomic

```
void qBitMemTglAtomic(
  unsigned BitNbr);    // The bit number to toggle
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function toggles a bit in special memory. This function is a specific operation because it is atomic and it can only operate on special memory. See the user guide for more information.

Parameters and return value

Parameter	Description
unsigned Bit	Bit Number

Error conditions

None

6. qBitMemTst

```
bool qBitMemTst(
  unsigned BitNbr);    // The bit number to test
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function test a bit in special memory. This function is a specific operation because it is atomic and it can only operate on special memory. See the user guide for more information.

Parameters and return value

Parameter	Description
bool return	Returns true if bit was set and false if bit was clear
unsigned BitNbr	Bit Number

Error conditions

None

7. qBitSet

```
void qBitSet(
  unsigned* Ptr,    // Address
  unsigned Bit);    // The bit to set
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function set the bit in the unsigned integer pointed to by Ptr. This function is an inline function doing the following operation:

```
*Ptr |= (1<<(Bit));
```

The function is NOT atomic. Don't use qBitSetAtomic() if atomicy is not required.

Parameters and return value

Parameter	Description
unsigned* Ptr	Address
unsigned Bit	Bit Number (0->15 or 0->31)

Error conditions

None

8. qBitSetAtomic

```
void qBitSetAtomic(
  unsigned* Ptr,    // Address
  unsigned Bit);    // The bit to set
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function set the bit in the unsigned integer pointed to by Ptr. This function is a specific operation because it is atomic.

Use the function qBitSet() if atomicy is not required.

Parameters and return value

Parameter	Description
unsigned* Ptr	Address
unsigned Bit	Bit Number (0->15 or 0->31)

Error conditions

None

9. qBitTgl

```
void qBitTgl(
  unsigned* Ptr,    // Address
  unsigned Bit);    // The bit to toggle
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function toggles the bit in the unsigned integer pointed to by Ptr. This function is an inline function doing the following operation:

```
*Ptr ^= (1<<(Bit));
```

The function is NOT atomic. Don't use qBitTglAtomic() if atomicy is not required.

Parameters and return value

Parameter	Description
unsigned* Ptr	Address
unsigned Bit	Bit Number (0->15 or 0->31)

Error conditions

None

10. qBitTglAtomic

```
void qBitTglAtomic(
  unsigned* Ptr,    // Address
  unsigned Bit);    // The bit to toggle
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function toggles the bit in the unsigned integer pointed to by Ptr. This function is a specific operation because it is atomic.

Use the function qBitTgl() if atomicy is not required.

Parameters and return value

Parameter	Description
unsigned* Ptr	Address
unsigned Bit	Bit Number (0->15 or 0->31)

Error conditions

None

11. qBitTst

```
bool qBitTst(
  unsigned* Ptr,    // Address
  unsigned Bit);    // The bit to togle
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function test the bit in the unsigned integer pointed to by Ptr. This function is an inline function doing the following operation:

If (*Ptr & (1<<(Bit)) return true else return false;

Parameters and return value

Parameter	Description
return bool	Return true if bit set and false if bit is clear
unsigned* Ptr	Address
unsigned Bit	Bit Number (0->15 or 0->31)

Error conditions

None

12. qBytClr

```
void qBytClr(
  unsigned* From,    // From address
  unsigned Len);     // Length in number of bytes
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function clears (filled with all 0 bits) an array of unsigned bytes. These functions are optimized for the type of processor and operate much faster then the C functions.

Parameters and return value

Parameter	Description
uint8_t* From	From address
unsigned Len	The length in bytes

Error conditions

None

13. qBytDecAtomic

```
void qBytDecAtomic(
  uint8_t* p)      // Address for the operation
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function decrement the unsigned byte location pointed by p atomiccaly. An atomic operation is an operation that will always be executed without any other thread, fiber or interrupt being able to read or change the value during the operation.

Parameters and return value

Parameter	Description
uint8_t* p	Pointer to the location of the operation

Error conditions

None

14. qBytIncAtomic

```
void qBytIncAtomic(
  uint8_t* p)      // Address for the operation
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function increment the integer location pointed by p atomiccaly. An atomic operation is an operation that will always be executed without any other thread, fiber or interrupt being able to read or change the value during the operation.

Parameters and return value

Parameter	Description
uint8_t* p	Pointer to the location of the operation

Error conditions

None

15. qBytMov

```
void qBytMov(
  uint8_t* From,     // From address
  uint8_t* To,       // To address
  unsigned Len);     // Length in number of bytes
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function moves unsigned bytes from a From location to a To location. Both locations can not overlap. This is not tested and the responsibility of the developer.

These functions are optimized for the type of processor and operate much faster then the C functions.

Parameters and return value

Parameter	Description
uint8_t* From	From address
uint8_t* To	To address
unsigned Len	The length in bytes

Error conditions

None

16. qBytSet

```
void qBytSet(
  uint8_t* From,    // From address
  unsigned Len);    // Length in number of bytes
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function set (filled with all 1 bits) an array of unsigned bytes. These functions are optimized for the type of processor and operate much faster then the C functions.

Parameters and return value

Parameter	Description
uint8_t* From	From address
unsigned Len	The length in bytes

Error conditions

None

17. qCrcCalculate

```
uint32_t qCrcCalc(
  uint32_t Crc,    // The CRC to start with
  uint8_t* Buf,    // The buffer to get the CRC from
  unsigned Size);  // The size of the buffer
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function calculates the CRC of a number of characters. Because there is an option for a start value this function allows you to add different pieces together to create a combined CRC.

The polynomial is 0x1EDC6F41. This CRC is called in the Castagnoli or CRC32C test. See http://en.wikipedia.org/wiki/Cyclic_redundancy_check

Parameters and return value

Parameter	Description
Returns uint32_t	The calculated CRC
uint32_t CRC	The start value of 0xffffffff or previous calculated values.
uint8_t* Buf	The buffer.
unsigned Size	The size of the buffer

Error conditions

None

18. qCrcGet

```
uint32_t qCrcGet(
  uint8_t* Buf,     // The buffer to get the CRC from
  unsigned Size);   // The size of the buffer
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function calculates the CRC of a number of characters. The polynomial is 0x1EDC6F41. This CRC is called in the Castagnoli or CRC32C test. See http://en.wikipedia.org/wiki/Cyclic_redundancy_check

Parameters and return value

Parameter	Description
Returns uint32_t	The calculated CRC
uint8_t* Buf	The buffer.
unsigned Size	The size of the buffer

Error conditions

None

19. qCrtEnter

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

Before Start	Thread	Critical Section	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

20. qCrtExit

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

Before Start	Thread	Critical Section	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

21. qDivScaling

```
uint16_t qDivScaling(
  uint16_t v1,
  uint16_t v2,
  uint16_t v3);
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns (v1*v2)/v3. It can scale a variable based on 2 other variables. This is often the case when a variable needs to be changed a bit to accomplish trimming for small in-accuracies. If for example a reference voltage is defines as 2.500V but after calibration the refrenece voltage seems to be 2.510V a measured value need to be scaled. So if we measure 1.250V the real value is 1.255V. We need to calculate Voltage=(MeasureVoltage*2510)/2500. This is exacly what this function does. The process can be optimized in assembler.

Parameters and return value

Parameter	Description
Returns uint16_t	The result of the scaling
uint16_t v1	Measured value
uint16_t v2	Calibrated value
unit16_t v3	Reference value

Error conditions

None

22. qDivU3216

```
uint32_t qDivU3216(
  uint32_t Dividend,
  uint16_t Divisor);
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function divides the dividend by the divisor and returns the result. The remainder is stored at the address of the Remainder.

This function is provided to optimize divide operations with smaller values.

Parameters and return value

Parameter	Description
Returns uint32_t	The result of the divider
uint32_t Dividend	Dividend
uint16_t Divisor	Divisor

Error conditions

None

23. qDivU3216R

```
uint32_t qDivU3216R(
  uint32_t Dividend,
  uint16_t Divisor,
  uint16_t* Remainder);
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function divides the dividend by the divisor and returns the result. The remainder is stored at the address of the Remainder.

This function is provided to optimize divide operations with smaller values.

Parameters and return value

Parameter	Description
Returns uint32_t	The result of the divider
uint32_t Dividend	Dividend
uint16_t Divisor	Divisor
unit16_t* Remainder	Remainder

Error conditions

None

24. qDivU6416

```
uint64_t qDivU6416 (
  uint64_t Dividend,
  uint16_t Divisor);
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function divides the dividend by the divisor and returns the result. The remainder is stored at the address of the Remainder.

This function is provided to optimize divide operations with smaller values.

Parameters and return value

Parameter	Description
Returns uint64_t	The result of the divider
uint64_t Dividend	Dividend
uint16_t Divisor	Divisor

Error conditions

None

25. qDivU6416R

```
uint32_t qDivU6416 (
  uint64_t Dividend,
  uint16_t Divisor,
  uint16_t* Remainder);
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function divides the dividend by the divisor and returns the result. The remainder is stored at the address of the Remainder.

This function is provided to optimize divide operations with smaller values.

Parameters and return value

Parameter	Description
Returns uint64_t	The result of the divider
uint64_t Dividend	Dividend
uint16_t Divisor	Divisor
unit16_t* Remainder	Remainder

Error conditions

None

26. qDtmAddDays

```
uint32_t qDtmAddDays(
  uint32_t DatTim,  // The date-time to add to
  int Day);         // The number of days to add or
                    // subtract if negative
```

Before Start	Thread	Critical Section	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 uint32_t	The calculated new date
uint32_t DatTim	The date-time in internal format.
int Day	The number of days to add.

Error conditions

None

27. qDtmAddHours

```
uint32_t qDtmAddHours(
  uint32_t DatTim,  // The date-time to add to
  int Hour);        // The number of hours to add or
                    // subtract if negative
```

Before Start	Thread	Critical Section	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 uint32_t	The calculated new date
uint32_t DatTim	The date-time in internal format.
int Hour	The number of hours to add.

Error conditions

None

28. qDtmAddMinutes

```
uint32_t qDtmAddMinutes(
  uint32_t DatTim,  // The date-time to add to
  int Minute);      // The number of minutes to add or
                    // subtract if negative
```

Before Start	Thread	Critical Section	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 uint32_t	The calculated new date
uint32_t DatTim	The date-time in internal format.
int Minute	The number of minutes to add.

Error conditions

None

29. qDtmAddSeconds

```
uint32_t qDtmAddSeconds(
  uint32_t DatTim,  // The date-time to add to
  int Second);      // The number of seconds to add or
                    // subtract if negative
```

Before Start	Thread	Critical Section	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 uint32_t	The calculated new date
uint32_t DatTim	The date-time in internal format.
int Second	The number of seconds to add.

Error conditions

None

30. qDtmAddYears

```
uint32_t qDtmAddYears(
  uint32_t DatTim,  // The date-time to add to
  int Year);        // The number of Years to add or
                    // subtract if negative
```

Before Start	Thread	Critical Section	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 uint32_t	The calculated new date
uint32_t DatTim	The date-time in internal format.
int Year	The number of years to add.

Error conditions

None

31. qDtmDayOfWeek

```
unsigned qDtmDayOfWeek(
  uint32_t DatTim)    // The date-time to find day of
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

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

Parameters and return value

Parameter	Description
Returns unsigned	The function returns day of the week for a given date. Monday=1, Tuesday=2 … … … Sunday=7
uint32_t DatTim	The date-time in internal format.

Error conditions

None

32. qDtmFromBcdDT

```
uint32_t qDtmFromBcdDT(
  pDATETIME DateTime)    // The date-time to convert
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function converts a BCD date-time 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 uint32_t	The function returns the date-time in the internal format. If the date-time is incorrect the function returns zero.
pDATETIME DateTime	A pointer to the date-time structure.

Error conditions

None

33. qDtmFromYMDHMS

```
uint32_t qDtmFromYMDHMS(
  uint8_t Year,
  uint8_t Month,
  uint8_t Day,
  uint8_t Hour,
  uint8_t Minute,
  uint8_t Second);
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

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

Parameters and return value

Parameter	Description
Returns uint32_t	The function returns the date-time in the internal format. If the date-time is incorrect the function returns zero.
uint8_t Year	The number of years.
uint8_t Month	The month of the year from 1 to 12.
uint8_t Day	The day of the month.
uint8_t Hour	The hour of the day in 24 hour clock.
uint8_t Minute	The number of minutes.
uint8_t Second	The number of seconds.

Error conditions

None

34. qDtmToBcdDtm

```
void qDtmToBcdDtm(
  uint32_t DatTim
  pBCD_DTM bcdDatTim);
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

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

Parameters and return value

Parameter	Description
uint32_t DatTim	The date-time in internal format.
pBCD_DTM bcdDatTim	Pointer to the bcd datetime

Error conditions

None

35. qEvtClear

```
unsigned qEvtClear(
  pEVT pEvt,              // The event set to clear
  unsigned EventFlags);   // The flags to set
```

Before Start	Thread	Critical Section	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:

```
unsigned flags;          // variable to return the flags
pEVT 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 unsigned	Returns the event flags before the clear operation.
pEVT pEvt	A pointer to the event set object. Must be returned from qEvtCreate() or qEvtOpen() with the correct name.
unsigned 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.

36. qEvtClose

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

Before Start	Thread	Critical Section	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
pEVT 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.

37. qEvtCreate

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

Before Start	Thread	Critical Section	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 pEVT	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.

The developer must give every object a unique name.

38. qEvtOpen

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

Before Start	Thread	Critical Section	Fiber	ISR	Possible 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 exists the threads is suspended.

Parameters and return value

Parameter	Description
Returns pEVT	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_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.

39. qEvtOpenNB

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

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

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

Parameters and return value

Parameter	Description
Returns pEVT	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.

40. qEvtOpenTO

```
pEVT qEvtOpenTO(            // Retuns NULL when timed-out
  char* pName,              // The Name of the object
  int32_t TimeOut);         // The timeout
```

Before Start	Thread	Critical Section	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 exists the threads is suspended. The timeout specifies how long the thread is willing to wait.

Parameters and return value

Parameter	Description
Returns pEVT	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.
int32_t TimeOut	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. 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.

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_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 µSecond.
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.

41. qEvtSignal

```
void qEvtSignal(
  pEVT pEvt,                // The event set to signal
  unsigned EventFlags;      // The flags to set
```

Before Start	Thread	Critical Section	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.

If this function is called from an ISR 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
pEVT pEvt	A pointer to the event set object. Must be returned from qEvtCreate() or qEvtOpen() with the correct name.
unsigned 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_NO_FLAGS	Not one flag in EventFlags is set

42. qEvtWait

```
unsigned qEvtWait(
  pEVT pEvt,              // The event set to wait for
  unsigned EventFlags,    // The flags to wait for
  WAIT_TYPE WaitType);    // The type of wait (see below)
```

Before Start	Thread	Critical Section	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 unsigned	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.
pEVT pEvt	A pointer to the event set object. Must be returned from qEvtCreate() or qEvtOpen() with the correct name.
unsigned EventFlags	The event flags to wait for. At least one flag must be set.
WAIT_TYPE WaitType	The type of wait. The following are defined. WAIT_TYPE_ALL means wait until all flags are set. This is also called the AND scenario. WAIT_TYPE_ALL_CLEAR means wait until all flags are set and if this situation occurs reset the flags that the thread was waiting for. WAIT_TYPE_ANY means wait until one of the flags is set. This is also called the OR scenario. WAIT_TYPE_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_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.

43. qEvtWaitNB

```
unsigned qEvtWait(
  pEVT pEvt,              // The event set to wait for
  unsigned EventFlags,    // The flags to wait for
  WAIT_TYPE WaitType);    // The type of wait (see below)
```

Before Start	Thread	Critical Section	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 unsigned	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.
pEVT pEvt	A pointer to the event set object. Must be returned from qEvtCreate() or qEvtOpen() with the correct name.
unsigned EventFlags	The event flags to wait for. At least one flag must be set.

Parameter	Description
WAIT_TYPE WaitType	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. WAIT_TYPE_ALL means wait until all flags are set. This is also called the AND scenario. WAIT_TYPE_ALL_CLEAR means wait until all flags are set and if this situation occurs reset the flags that the thread was waiting for. WAIT_TYPE_ANY means wait until one of the flags is set. This is also called the OR scenario. WAIT_TYPE_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_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

44. qEvtWaitTO

```
unsigned qEvtWaitTO(
  pEVT pEvt,              // The event set to wait for
  unsigned EventFlags,    // The flags to wait for
  WAIT_TYPE WaitType,     // The type of wait (see below)
  int32_t TimeOut);        // The maximum wait time
```

Before Start	Thread	Critical Section	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 unsigned	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.
pEVT pEvt	A pointer to the event set object. Must be returned from qEvtCreate() or qEvtOpen() with the correct name.
unsigned EventFlags	The event flags to wait for. At least one flag must be set.

Parameter	Description
WAIT_TYPE WaitType	The type of wait. The following are defined. WAIT_TYPE_ALL means wait until all flags are set. This is also called the AND scenario. WAIT_TYPE_ALL_CLEAR means wait until all flags are set and if this situation occurs reset the flags that the thread was waiting for. WAIT_TYPE_ANY means wait until one of the flags is set. This is also called the OR scenario. WAIT_TYPE_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.
int32_t TimeOut	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. 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.

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 µSecond.

45. qErrNotify

```
unsigned qErrNotify(
  unsigned 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 this function. See the user guide for more information.

Parameters and return value

Parameter	Description
Return unsigned	The developer must return 0 if the function is called with 0.
unsigned 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.

46. qFbrCreate

```
void qFbrCreate(
  uint8_t Priority,      // The priority of the fiber
  void(*pFbr)(void));    // The fiber function itself
```

Before Start	Thread	Critical Section	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
uint8_t 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.

47. qFbrEnqueue0

```
void qFbrEnqueue0(      //
  void (*pFbr)());      // The function to queue
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function queues a fiber for processing without arguments.

Parameters and return value

Parameter	Description
void (*pFbr)()	The function to queue as fiber.

Error conditions

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

48. qFbrEnqueue1

```
void qFbrEnqueue1(       //
  void (*pFbr)(void*),   // The function to queue
  void *pPar);           // Par for the function
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function queues a fiber for processing with one argument

Parameters and return value

Parameter	Description
void (*pFbr)(void*)	The function to queue as fiber.
void *pPar	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.

Error conditions

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

49. qFbrEnqueue2

```
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	Critical Section	Fiber	ISR	No preemption

Description

This function queues a fiber for processing with two arguments.

Parameters and return value

Parameter	Description
void (*pFbr)(void*,void*)	The function to queue as fiber.
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 to small the function will loop.

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

```
void qTskSpawn1();
```

Before Start	Thread	Critical Section	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

51. qFbrSpawnRtcc

```
void qTskSpawnRtcc();
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function spawns a fiber to check if something has expired in the RTCC. This function is available so developers can write their own RTCC clock functions. See for an example the function yRtcc4.c

Parameters and return value

None

Error conditions

None

52. qFbrStatCycles

```
uint64_t qFbrStatTotalCycles();
```

Before Start	Thread	Critical Section	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 uint64_t	The number of cycles

Error conditions

Error	Description
qERR_FBR_ISR	The function is called from an ISR.
qERR_FBR_STAT_OFF	Statistics is not running

53. qFixAlloc

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

Before Start	Thread	Critical Section	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.

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.
pFPL pFpl	The fixed memory pool that has been created with qFixCreate().

Error conditions

None

54. qFixAllocClr

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

Before Start	Thread	Critical Section	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.
pFPL pFpl	The fixed memory pool that has been created with qFixCreate().

Error conditions

None

55. qFixCreate

```
pFPL qFixCreate(              // The function creates
  unsigned Size,              // a fixed memory pool
  unsigned NbrBlocks);        //
```

Before Start	Thread	Critical Section	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 pFPL	A pointer to the new fixed pool. The function returns null if there is no memory available.
unsigned Size	The size of the memory block to allocate.
unsigned 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
qFPL_ISR	The function is called from an ISR

56. qFixClose

```
void qFixClose(          // The function closes a fixed
  pFPL pFpl);            // memory pool
```

Before Start	Thread	Critical Section	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
pFPL pFpl	The pool to close

Error conditions

Error	Description
qFPL_ISR	The function is called from an ISR

57. qFixFree

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

Before Start	Thread	Critical Section	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

58. qHeaAlloc

```
void* qHeaAlloc(          // Allocates heap memory
   unsigned Size);        // Size of the memory to
                          // allocate
```

Before Start	Thread	Critical Section	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 of the memory is guaranteed zero.

There is no qHeaFree() 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.
unsigned 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.

59. qHeaSize

```
unsigned qHeaSize(); // Returns free heap size
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function returns the free heap size.

Parameters and return value

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

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.

60. qKrnError

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

Before Start	Thread	Critical Section	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
unsigned Error	The error that will be signaled.

Error conditions

None

61. qKrnInCritical

```
unsigned qKrnInCritical();
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns not 0 if the kernel is in a critical section and 0 if not.

Parameters and return value

Parameter	Description
return unsigned	Return not 0 if in critical section

Error conditions

None

62. qKrnInFiber

```
unsigned qKrnInFiber();
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns not 0 if the kernel is in a fiber and 0 if not.

Parameters and return value

Parameter	Description
return unsigned	Return not 0 if in fiber

Error conditions

None

63. qKrnInCritical

```
unsigned qKrnInCritical();
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns not 0 if the kernel is in a critical section and 0 if not.

Parameters and return value

Parameter	Description
return unsigned	Return not 0 if in critical section

Error conditions

None

64. qKrnInit

```
pERR qKrnInit(
  uint32_t ClockFrequency,       // MIPS
  unsigned IdleThreadStackSize, // 64 good start value
  unsigned InterruptStackSize,  // 256 good start value
  unsigned FiberQueueSize);     // 4 good start value
```

Before Start	Thread	Critical Section	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.
- Creates memory for the interrupt stack.
- If statistics is enabled it will setup statistics.
- Setup the timer and RTCC
- 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 pERR	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.
uint32_t ClockFrequency	Speed of the processor in MIPS
unsigned IdleTreadStackSize	Normally a value of 64 bytes should be enough. If the qKrnIdleNotification() is implemented it should be more
unsigned InterruptStackSize	Size of interrupt stack. See the user manual for more information about the interrupt stack
unsigned FiberQueueSize	Size in elements. Minimum of 4 is in most case enough

Error conditions

Error	Description
qERR_KRN_MEM_SIZE	There is not enough memory available to initialize the system and the defined objects.
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.

65. qKrnInitEds

```
pERR qKrnInit(
  uint32_t ClockFrequency,       // MIPS
  unsigned IdleThreadStackSize, // 64 good start value
  unsigned InterruptStackSize,  // 256 good start value
  unsigned FiberQueueSize,      // 4 good start value
  unsigned MaxEdsStackSize);    // See user manual
```

Before Start	Thread	Critical Section	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.
- Creates memory for the interrupt stack.
- If statistics is enabled it will setup statistics.
- Setup the timer and RTCC
- 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.)
- Will allocate memory to copy the maximum EDS thread stack into.
- 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. This 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 pERR	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.
uint32_t ClockFrequency	Speed of the processor in MIPS
unsigned IdleTreadStackSize	Normally a value of 64 bytes should be enough. If the qKrnIdleNotification() is implemented it should be more
unsigned InterruptStackSize	Size of interrupt stack. See the user manual for more information about the interrupt stack
unsigned FiberQueueSize	Size in elements. Minimum of 4 is in most case enough
unsigned MaxEdsStackSize	This defines the maxiumum EDS thread stack size that can be used.

Error conditions

Error	Description
qERR_KRN_MEM_SIZE	There is not enough memory available to initialize the system and the defined objects.
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.

66. qKrnNtfIdle

```
void qKrnNtfIdle();         // 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 xIdleThread (void* p) {  // The idle thread
    while(1) {
        … … …
        qNtfIdle();             // qNftIdle() called
    }
}
```

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

67. qKrnNtfSwitch

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

Description

The function will be called when switch notification is enabled and ***Q·Kernel*** switches from one thread to another thread. There is a “weak” function that will be called when the developer does not provides one.

The developer can use the pTCB 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 thread related information. 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.

68. qKrnStack

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

Before Start	Thread	Critical Section	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 unsigned	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.

69. qKrnStart

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

Before Start	Thread	Critical Section	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
- 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.

70. qKrnStatOff

```
void qKrnStatOff();
```

Before Start	Thread	Critical Section	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.

71. qKrnStatOn

```
void qKrnStatOn();
```

Before Start	Thread	Critical Section	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.

72. qKrnSwitchNotificationOff

```
void qKrnSwitchNotificationOff();
```

Before Start	Thread	Critical Section	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 created unpredictable results.

73. qKrnSwitchNotificationOn

```
void qKrnSwitchNotificationOn();
```

Before Start	Thread	Critical Section	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 created unpredictable results.

74. qKrnTrackingIdle

```
void qKrnTrackingIdle(
  unsigned* pIdleAdr,  // Tracking Addres (Idle)
  unsigned  IdleBit);  // Bit number (Idle)
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function set the tracking for the kernel power mode idle. The developer has to specify an address of the external port and the bit number in the port.

If the address is NULL tracking is disabled. (This is the default situation). Enabling or disabling tracking does not influence the performance, therefore tracking is non-

Parameters and return value

Parameter	Description
pIdleAdr	Addres for idle tracking. NULL disables idle tracking
IdleBit	Bit Number for idle tracking

Error conditions

Error	Description
qERR_KRN_ISR	The function is called from an ISR.
qERR_KRN_IDLE_BIT_NBR	The run bit number is larger then the number of bits in an interger-1

75. qKrnTrackingRun

```
void qKrnTrackingSleep(
  unsigned* pRunAdr,   // Tracking Addres (Running)
  unsigned  RunBit);   // Bit number (Running)
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function set the tracking for the kernel power mode Run. The developer has to specify an address of the external port and the bit number in the port.

If the address is NULL tracking is disabled. (This is the default situation). Enabling or disabling tracking does not influence the performance, therefore tracking is non-

Parameters and return value

Parameter	Description
pRunAdr	Addres for run tracking. NULL disables run tracking
RunBit	Bit Number for run tracking

Error conditions

Error	Description
qERR_KRN_ISR	The function is called from an ISR.
qERR_KRN_RUN_BIT_NBR	The run bit number is larger then the number of bits in an interger-1

76. qKrnTrackingSleep

```
void qKrnTrackingSleep(
  unsigned* pSleepAdr, // Tracking Addres (Sleeping)
  unsigned  SleepBit); // Bit number (Sleeping)
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function set the tracking for the kernel power mode sleep. The developer has to specify an address of the external port and the bit number in the port.

If the address is NULL tracking is disabled. (This is the default situation). Enabling or disabling tracking does not influence the performance, therefore tracking is non-

Parameters and return value

Parameter	Description
pSleepAdr	Addres for sleep tracking, NULL disables sleep tracking
SleepBit	Bit Number for sleep tracking

Error conditions

Error	Description
qERR_KRN_ISR	The function is called from an ISR.
qERR_KRN_SLEEP_BIT_NBR	The run bit number is larger then the number of bits in an interger-1

77. qKrnUSecOff

```
void qKrnUSecOff();
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function disables µSecond gathering.

Parameters and return value

None

Error conditions

Error	Description
qERR_KRN_ISR	The function is called from an ISR.

78. qKrnUSecOn

```
void qKrnStatOn();
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function enables µSecond 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.

79. qKrnVersion

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

Before Start	Thread	Critical Section	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:

```
typedef struct sVER{
    uint8_t major;
    uint8_t minor;
    uint16_t build;
} VER;
typedef const VER* pVER;
```

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 pVER	The function returns a pointer to the version structure.

Error conditions

None

80. qMemAlloc

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

Before Start	Thread	Critical Section	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.
unsigned Size	The size of the memory block to allocate.

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.

81. qMemAllocClr

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

Before Start	Thread	Critical Section	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.
unsigned Size	The size of the memory to allocate.

Error conditions

Error	Description
qERR_MEM_ISR	The function is called from an ISR.

82. qMemAllocFromPool

```
void* qMemAllocFromPool(// Allocates memory from a
   pMPL pMpl);          // memory pool
```

Before Start	Thread	Critical Section	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.
pMPL 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.

83. qMemAllocFromPoolClr

```
void* qMemAllocFromPoolClr(// Allocates memory from…
   pMPL pMpl);             // …a memory pool and…
                           // …clears the memory.
```

Before Start	Thread	Critical Section	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 of 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.
pMPL 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.

84. qMemAllocFromPoolFast

```
void* qMemAllocFromPoolFast(// Allocates memory from a
   pMPL pMpl);              // memory pool
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function allocates memory from the pool without parameter checking 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.
pMPL pMpl	The address of a memory pool.

Error conditions

None

85. qMemFree

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

Before Start	Thread	Critical Section	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 one of the allocation functions. (qMemAlloc.....)

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.

86. qMemFreeFast

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

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function frees memory and returns the memory to one of the variable pools without parameter checking

Parameters and return value

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

Error conditions

None

87. qMemPool

```
pMPL qMemPool(       // Returns the pool with the
                     // specified blocksize
   unsigned Size);   // Size of the blocks
```

Before Start	Thread	Critical Section	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 pMPL	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.
unsigned 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.

88. qMemPoolAdd

```
unsigned qMemPoolAdd(          //
   pMPL pMpl,                  //
   unsigned NbrBlocks);        //
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function allocates memory from the heap and adds blocks to the memory pool.

Parameters and return value

Parameter	Description
Returns unsigned	The function returns the number memory blocks that where added.
pMPL pMpl	The memory pool.
unsigned 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.

89. qMemPoolNext

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

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function returns a pointer to the next pool. To get the first memory pool specify pMpl as NULL.

Parameters and return value

Parameter	Description
Returns pMPL	The function returns the next memory pool.
pMPL 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.

90. qMemPoolSize

```
unsigned qMemPoolSize(          //
   pMPL pMpl);                  //
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function returns the size of the pool.

Parameters and return value

Parameter	Description
Returns unsigned	The function returns the size of the memory pool.
pMPL 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.

91. qMemRealloc

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

Before Start	Thread	Critical Section	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 fits 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 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.
void *p	Pointer to an existing memory block
unsigned 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.

92. qMsgAlloc

```
pMSG qMsgAlloc(       // Allocates a message
  uint16_t Size,      // with the specified size
  uint16_t MsgType); // and a specified message type
```

Before Start	Thread	Critical Section	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 pMSG	A pointer to the message. The function returns a null pointer if there is no memory available.
uint16_t Size	The maximum size of the message. The Size must be between 8 and 32000 inclusive.
uint16_t MsgType	The type of message specified by the user

Error conditions

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

93. qMsgAllocFast

```
pMSG qMsgAlloc(       // Allocates a message
  pMPL pMpl,          // pool to use
  uint16_t MaxSize,   // maximim size
  uint16_t MsgType); // and a specified message type
```

Before Start	Thread	Critical Section	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 specified variable memory pool. The maximum size is to specify the size of the message. This must be smaller or equal to the pool size larger then the pool size – 16. So the pool and max size must be not too far out.

Parameters and return value

Parameter	Description
Returns pMSG	A pointer to the message. The function returns a null pointer if there is no memory available.
uint16_t Size	The maximum size of the message. The MaxSize must be between poolSize - 15 and poolSize inclusive.
uint16_t MsgType	The type of message specified by the user

Error conditions

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

94. qMsgCopy

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

Before Start	Thread	Critical Section	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 pMSG	A pointer to the message. The function returns a null pointer if there is no memory available.
unsigned pMSG	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().

95. qMsgDataSize

```
uint16_t qMsgDataSize(      // returns the real size of
  pMSG p);                  // the message
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function returns the real message size.

Parameters and return value

Parameter	Description
uint16_t	Returns the maximum message size
pMSG p	The message pool

Error conditions

Error	Description
qERR_MSG_ID	This is not a valid message pool

96. qMsgFixAlloc

```
pMSG qMsgFixAlloc(    // Allocates a message
  pFIX pFix);         // From fixed memory
```

Before Start	Thread	Critical Section	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 pMSG	A pointer to the message. The function returns a null pointer if there is no memory available.
pFIX pFix	A pointer to a fixed memory block returned from qMsgFixCreate()

Error conditions

No errors are thrown

97. qMsgFixCreate

```
pMSG qMsgFixCreate(     // Create a fixed message pool
  uint16_t MsgSize,     // Size of the message
  uint16_t NbrMessages);// Number of messages to hold
```

Before Start	Thread	Critical Section	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 pMSG	A pointer to the fixed memory pool message. The function returns a null pointer if there is no memory available.
uint16_t MsgSize	The size of the message.
uint16_t NbrMessages	The maximum number of messages the pool can hold.

Error conditions

Error	Description
qFIX_ISR	The function is called from an ISR

98. qMsgFree

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

Before Start	Thread	Critical Section	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.

If this function is called from an ISR 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
pMSG pMsg	A pointer to the message.

Error conditions

Error	Description
qERR_MSG_ID	The message is not allocated by qMsgAlloc().

99. qMsgMaxSize

```
uint16_t qMsgMaxSize(       // returns the max size of
  pMSG p);                  // the message
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

The function returns the maximum message size.

Parameters and return value

Parameter	Description
uint16_t	Returns the maximum message size
pMSG p	The message pool

Error conditions

Error	Description
qERR_MSG_ID	This is not a valid message pool

100. qMsgPublish

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

Before Start	Thread	Critical Section	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.

If this function is called from an ISR 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
pQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.
pMSG pMsg	The message to be published to all subscribers.

Error conditions

Error	Description
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().

101. qMsgRead

```
pMSG qMsgRead(          // returns pointer to message
  pPIP pPip);           // The pipe to read from
```

Before Start	Thread	Critical Section	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.

If this function is called from an ISR the system will spawn the notification of the writer, so the writer will never be notified in an ISR.

Parameters and return value

Parameter	Description
Returns pMSG	The message read from the pipe or NULL
pPIP 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.

102. qMsgReceive

```
pMSG qMsgReceive(        // Returns a message
  pQUE pQue);            // The message queue to read
```

Before Start	Thread	Critical Section	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 pMSG	Returns a pointer to the message. If a time-out occurs the function will return a null pointer.
pQUE 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_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.

103. qMsgReceiveNB

```
pMSG qMsgReceiveNB(
  pQUE pQue);              // The message queue to read
```

Before Start	Thread	Critical Section	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 pMSG	Returns a pointer to the message. The function will return a null pointer if no message is available.
pQUE 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.

104. qMsgReceiveTO

```
pMSG qMsgReceiveTO(      // Returns a message
  pQUE pQue,             // The message queue to read
  int32_t TimeOut);      // The Timeout
```

Before Start	Thread	Critical Section	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 pMSG	Returns a pointer to the message. If a time-out occurs the function will return a null pointer.
pQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.
int32_t TimeOut	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. 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.

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_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 µSecond.
qERR_QUE_CRITICAL	This function cannot be called from within a critical section.

105. qMsgSend

```
unsigned qMsgSend(
  pQUE pQue,            // The queue to send to
  pMSG pMsg);           // The message to send
```

Before Start	Thread	Critical Section	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 unsigned	A zero value indicates that the function timed-out. A value of 1 indicates success.
pQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.
pMSG 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_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.

106. qMsgSendNB

```
unsigned qMsgSend(
  pQUE pQue,          // The queue to send to
  pMSG pMsg);         // The message to send
```

Before Start	Thread	Critical Section	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 unsigned	A zero value indicates that the queue is full. A value of 1 indicates success.
pQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.
pMSG 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().

107. qMsgSendTO

```
unsigned qMsgSendTO(
  pQUE pQue,           // The queue to send to
  pMSG pMsg,           // The message to send
  int32_t TimeOut);     // The timeout
```

Before Start	Thread	Critical Section	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 unsigned	A zero value indicates that the function timed-out. A value of 1 indicates success.
pQUE pQue	A pointer to the queue object. Must be returned from qQueCreate() or qQueOpen() with the correct name.
pMSG pMsg	The message to send to the queue or a waiting thread.
int32_t TimeOut	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. 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.

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_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 µSecond.

108. qMsgWrite

```
unsigned qMsgWrite( //
  pPIP pPip,        // The pipe to write into
  pMSG pMsg);       // A pointer to the message
```

Before Start	Thread	Critical Section	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.

If this function is called from an ISR the system will spawn the notification of the writer so the writer will never be notified in an ISR.

Parameters and return value

Parameter	Description
Returns unsigned	Zero if no space and 1 if the message is written.
pPIP pPip	The pipe object to write to.
pMSG 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_MSG_ID	The object is not a message object, has not been created or points to no object at all.

109. qMtxClose

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

Before Start	Thread	Critical Section	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
pMTX 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.

110. qMtxCreate

```
pMTX qMtxCreate(
  char *pName,              // The Name of the mutex
  bool Lock);               // if true create it locked
```

Before Start	Thread	Critical Section	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 pMTX	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.
uint8_t 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 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.

111. qMtxLock

```
unsigned qMtxLock(
  pMTX pMtx);              // The mutex to lock
```

Before Start	Thread	Critical Section	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 unsigned	Returns 0 if timed out and 1 if mutex is locked
pMTX 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 can own mutexes.
qERR_MTX_NO_START	The function is called before ***Q·Kernel*** is started.

112. qMtxLockNB

```
unsigned qMtxLockNB(
  pMTX pMtx;)              // The mutex to lock
```

Before Start	Thread	Critical Section	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 unsigned	Returns 0 if timed out and 1 if mutex is locked
pMTX 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 can own mutexes.
qERR_MTX_NO_START	The function is called before ***Q-Kernel*** is started.

113. qMtxLockTO

```
unsigned qMtxLockTO(
  pMTX pMtx,              // The mutex to lock
  int32_t TimeOut);        //
```

Before Start	Thread	Critical Section	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 unsigned	Returns 0 if timed out and 1 if mutex is locked
pMTX pMtx	A pointer to the mutex object. Must be returned from qMtxCreate() or qMtxOpen() with the correct name.
int32_t TimeOut	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. 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.

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 µSecond.
qERR_MTX_FBR	The function is called from a fiber. Only threads can own mutexes.

114. qMtxOpen

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

Before Start	Thread	Critical Section	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 exists the threads is suspended.

Parameters and return value

Parameter	Description
Returns pMTX	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 can own mutexes.
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.

115. qMtxOpenNB

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

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns a pointer to an existing mutex object.

Parameters and return value

Parameter	Description
Returns pMTX	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.

116. qMtxOpenTO

```
pMTX qMtxOpenTO(          // Retuns NULL when timed-out
  char *pName,            // The Name of the object
  int32_t TimeOut);        // The timeout
```

Before Start	Thread	Critical Section	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 exists the threads is suspended. The timeout specifies how long the thread is willing to wait.

Parameters and return value

Parameter	Description
Returns pMTX	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.
int32_t TimeOut	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. 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.

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 can own mutexes.
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 µSecond.
qERR_MTX_MEMORY	There is no memory available to handle the open request.

117. qMtxOwner

```
pTCB qMtxOwner(
  pMTX pMtx);              // The mutex
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns the owner of the specified mutex or null if the mutex is not owned by anybody.

Parameters and return value

Parameter	Description
Returns pTCB	Returns 0 if there is no owner and otherwise the TCB of the thread
pMTX 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.

118. qMtxUnlock

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

Before Start	Thread	Critical Section	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
pMTX 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 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.

119. qPipBlockSize

```
unsigned qPipBlockSize(   // returns block size
  pPIP pPip);
```

Before Start	Thread	Critical Section	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 unsigned	The block size of the pipe. This is the same as the parameter BlockSize in qPipCreate()
pPIP 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. qPipClose

```
void qPipClose(
  pPIP pPip);              // The Pipe to close
```

Before Start	Thread	Critical Section	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
pPIP 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.

121. qPipCreate

```
pPIP qPipCreate(
  char *pName,                          // The Name of the pipe
  uint16_t BlockSize,                   // Size of one block
  uint16_t MaxBlocks,                   // Maximum blocks
  void (*pNtfReader)(pPIP,unsigned),// Notify reader
  void (*pNtfWriter)(pPIP,unsigned));
                                        // Notify writer
```

Before Start	Thread	Critical Section	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(pMSG).

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 pPIP	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.
uint8_t 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(pMSG) for messages.
uint16_t 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) (pPIP, unsigned)	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 in the pipe available for reading.
void (*pNtfWriter) (pPIP, unsigned)	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 space available in the pipe. (Number of blocks that can be written until it is full)

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.

122. qPipEntries

```
unsigned qPipEntries(     // returns number of block …
  pPIP pPip);             // … currently in the pipe
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns the number of entries in the pipe.

Parameters and return value

Parameter	Description
Returns unsigned	The number of entries in the pipe. If the pipe is empty it returns 0.
pPIP 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.

123. qPipFreeBlocks

```
unsigned qPipFreeBlocks(// returns free number of …
  pPIP pPip);           // … blocks in the pipe
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns the free number of blocks that fit in the pipe.

Parameters and return value

Parameter	Description
Returns unsigned	The free number of blocks in the pipe.
pPIP 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.

124. qPipGet

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

Before Start	Thread	Critical Section	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()for synchronized reading from a pipe.

Parameters and return value

Parameter	Description
Returns unsigned	The number of elements read.
pPIP 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.
unsigned NbrBlocks	The number of blocks that the buffer can contain.

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_NBR_BLOCKS	Number of blocks specified as 0

125. qPipGetBytFast

```
bool qPipGetBytFast(  // returns true if byte is read
  pPIP pPip,          // The pipe to read from
  uint16_t *pBuffer); // A pointer to the data
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function reads a byte from a pipe without parameter checking. The function returns true if a byte was read and false when there is no information in the pipe.

It is the developer's responsibility to check that the pipe was created for bytes.

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()for synchronized reading from a pipe.

Parameters and return value

Parameter	Description
Returns unsigned	The number of elements read.
pPIP pPip	The pipe object to read from.
uint8_t *pBuffer	The buffer to read the information in.

Error conditions

None

126. qPipGetFast

```
unsigned qPipGetFast( // returns number of elements read
  pPIP pPip,          // The pipe to read from
  void *pBuffer,      // A pointer to the data
  unsigned NbrBlocks);// Size of the buffer in blocks
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function reads information from a pipe without parameter checking. 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()for synchronized reading from a pipe.

Parameters and return value

Parameter	Description
Returns unsigned	The number of elements read.
pPIP 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.
unsigned 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

None

127. qPipGetWordFast

```
bool qPipGetFast(      // returns true if byte is written
  pPIP pPip,           // The pipe to read from
  void *pBuffer);      // A pointer to the data
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function reads a 16 or 32 bit word from a pipe without parameter checking. The function returns true if a byte was read and false when there is no information in the pipe.

It is the developer's responsibility to check that the pipe was created for words.

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()for synchronized reading from a pipe.

Parameters and return value

Parameter	Description
Returns unsigned	The number of elements read.
pPIP pPip	The pipe object to read from.
unsgined *pBuffer	The buffer to read the information in. The buffer MUST be word aligned.

Error conditions

None

128. qPipMaxBlocks

```
unsigned qPipMaxBlocks( // returns maximum number of …
  pPIP pPip);           // … blocks in the pipe
```

Before Start	Thread	Critical Section	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 unsigned	The maximum number of blocks in the pipe. This is the same as the parameter MaxBlocks in qPipCreate()
pPIP 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.

129. qPipOpen

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

Before Start	Thread	Critical Section	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 pPIP	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_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.

130. qPipOpenNB

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

Before Start	Thread	Critical Section	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 exists the threads 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 pPIP	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.

131. qPipOpenTO

```
pPIP qPipOpenTO(          // Retuns NULL when timed-out
  char *pName,            // The Name of the object
  int32_t TimeOut);        // The timeout
```

Before Start	Thread	Critical Section	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 exists the threads 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 pPIP	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.
int32_t TimeOut	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. 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.

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 µSecond.
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.

132. qPipPut

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

Before Start	Thread	Critical Section	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()for synchronized writing to a pipe.

Parameters and return value

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

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_NBR_BLOCKS	Number of blocks specified as 0

133. qPipPutBytFast

```
bool qPipPutFast(      // returns true if byte is written
  pPIP pPip,           // The pipe to write into
  uint8_t *pBuffer);   // A pointer to the data
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function writes a byte into the pipe without parameter checking. The function returns true is the byte is written.

It is the developer's responsibility to check that the pipe was created for bytes.

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()for synchronized writing to a pipe.

Parameters and return value

Parameter	Description
Returns bool	Returns true if byte successfully written
pPIP pPip	The pipe object to write to.
uint8_t *pBuffer	The buffer that contains the information.

Error conditions

This function does not throw errors.

134. qPipPutWrdFast

```
bool qPipPutFast(     // returns true if word is written
  pPIP pPip,          // The pipe to write into
  unsigned *pBuffer); // A pointer to the data
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function writes a word into the pipe without parameter checking. The function returns true is the word has been written.

It is the developer's responsibility to check that the pipe was created for words.

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()for synchronized writing to a pipe.

Parameters and return value

Parameter	Description
Returns bool	Returns true id word is successfully written.
pPIP pPip	The pipe object to write to.
void *pBuffer	The buffer that contains the information. The developer must specify the correct buffer type.

Error conditions

This function does not throw errors.

135. qPipPutFast

```
unsigned qPipPutFast( // returns nbr of elements written
  pPIP pPip,          // The pipe to write into
  void *pBuffer,      // A pointer to the data
  unsigned NbrBlocks);// Size of buffer in number of …
                      // … elements
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function writes information into the pipe without parameter checking. 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()for synchronized writing to a pipe.

Parameters and return value

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

Error conditions

This function does not throw errors.

136. qPipRead

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

Before Start	Thread	Critical Section	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.

If this function is called from an ISR the system will notify the writer also in the ISR.

Parameters and return value

Parameter	Description
Returns unsigned	The number of elements read.
pPIP pPip	The pipe object to read from.
void *pBuffer	The buffer to read the information in. The developer must specify the correct buffer type.
unsigned 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_NBR_BLOCKS	The buffer size is incorrect. Must be 1 or higher.

137. qPipReadFast

```
unsigned qPipReadFast(// returns number of elements read
  pPIP pPip,          // The pipe to read from
  void *pBuffer,      // A pointer to the data
  unsigned NbrBlocks);// Size of the buffer in blocks
```

Before Start	Thread	Critical Section	Fiber	ISR	Possible preemption

Description

This function reads information from a pipe without parameter checking. 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.

If this function is called from an ISR the system will notify the writer also in the ISR.

Parameters and return value

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

Error conditions

None

138. qPipWrite

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

Before Start	Thread	Critical Section	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.

If this function is called from an ISR the system will notify the writer also in the ISR.

Parameters and return value

Parameter	Description
Returns unsigned	The number of elements written.
pPIP pPip	The pipe object to write to.
void *pBuffer	The buffer that contains the information. The developer must specify the correct buffer type.
unsigned 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_NBR_BLOCKS	The buffer size is incorrect. Must be 1 or higher.

139. qPipWriteFast

```
unsigned qPipWriteFast(// returns nbr elements written
  pPIP pPip,           // The pipe to write into
  void *pBuffer,       // A pointer to the data
  unsigned NbrBlocks); // Size of buffer in number of …
                       // … blocks
```

Before Start	Thread	Critical Section	Fiber	ISR	Possible preemption

Description

This function writes information into the pipe without parameter checking. 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.

If this function is called from an ISR the system will notify the writer also in the ISR.

Parameters and return value

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

Error conditions

None

140. qPubClose

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

Before Start	Thread	Critical Section	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
pPUB 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.

141. qPubCreate

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

Before Start	Thread	Critical Section	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 pPUB	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.

142. qPubOpen

```
pPUB qPubOpen(              // Retuns NULL when timed-out
  char *pName);             // The Name of the semphore
```

Before Start	Thread	Critical Section	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 pPUB	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_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.

143. qPubOpenNB

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

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns a pointer to an existing publish object.

Parameters and return value

Parameter	Description
Returns pPUB	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.

144. qPubOpenTO

```
pPUB qPubOpenTO(          // Retuns NULL when timed-out
  char *pName,            // The Name of the object
  int32_t TimeOut);        // The timeout
```

Before Start	Thread	Critical Section	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 exists the threads is suspended. The timeout specifies how long the thread is willing to wait.

Parameters and return value

Parameter	Description
Returns pPUB	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.
int32_t TimeOut	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. 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.

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 µSecond.
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.

145. qPubSubscribeFun

```
void qPubSubscribeFun(
  pPUB pPub,                // The publisher to subscribe
  void (*pFun)(void*, pMSG);
```

Before Start	Thread	Critical Section	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
pPUB pPub	A pointer to the publish object. Must be returned from qPubCreate() or qPubOpen() with the correct name.
void (*pFun)(void*, pMSG)	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.

146. qPubSubscribePip

```
void qPubSubscribePip(
  pPUB pPub,      // The publisher to subscribe
  pPIP pPip);
```

Before Start	Thread	Critical Section	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
pPUB pPub	A pointer to the publish object. Must be returned from qPubCreate() or qPubOpen() with the correct name.
pPIP 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.

147. qPubSubscribeQue

```
void qPubSubscribeQue(
  pPUB pPub,      // The publisher to subscribe
  pQUE pQue);
```

Before Start	Thread	Critical Section	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
pPUB pPub	A pointer to the publish object. Must be returned from qPubCreate() or qPubOpen() with the correct name.
pQUE 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.

148. qPwrPermitIdle

```
void qPwrPermitIdle()
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function allows the kernel to switch the system into idle mode.

Parameters and return value

None

Error conditions

None

149. qPwrPermitSleep

```
void qPwrPermitSleep()
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function allows the kernel to switch the system into sleep mode.

Parameters and return value

None

Error conditions

None

150. qPwrPreventIdle

```
void qPwrPreventIdle()
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function prevents the kernel to switch the system into idle mode.

Parameters and return value

None

Error conditions

None

151. qPwrPreventSleep

```
void qPwrPreventSleep()
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function prevents the kernel to switch the system into sleep mode.

Parameters and return value

None

Error conditions

None

152. qQueClose

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

Before Start	Thread	Critical Section	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
pQUE 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 of other threads are using the queue.

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

153. qQueCreate

```
pQUE qQueCreate(
  char *pName,              // The Name of the queue
  uint16_t Size);           // Nbr of message in queue
```

Before Start	Thread	Critical Section	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 pQUE	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.
unsigned 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.

154. qQueOpen

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

Before Start	Thread	Critical Section	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 pQUE	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_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.

155. qQueOpenNB

```
pQUE qQueOpenNB(            //
  char *pName);             // The Name of the EventSet
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns a pointer to an existing queue object.

Parameters and return value

Parameter	Description
Returns pQUE	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_QUE_ISR	The function is called from an ISR.
qERR_QUE_NO_NAME	Queues without a name can't be opened.

156. qQueOpenTO

```
pQUE qQueOpenTO(          // Retuns NULL when timed-out
  char *pName,            // The Name of the object
  int32_t TimeOut);        // The timeout
```

Before Start	Thread	Critical Section	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 exists the threads 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 pQUE	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.
int32_t TimeOut	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. 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.

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 µSecond.
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.

157. qRanGet

```
uint32_t qRanGet()
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns a random value based on George Marsaglia (multiply with carry) algorithm random number generation. It's a simple algorithms that nevertheless produce high quality output. The function will notify the system by calling qRanNtfSeed() the first time the function is used. After that the function operates on its own. See the user guide for more information.

Parameters and return value

Parameter	Description
Retrun uint32_t	New random value

Error conditions

None

158. qRanNtfSeed

```
void qRanNtfSeed(
  uint32_t X,        // Seed number X
  uint32_t Y);       // Seed number Y
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

If the function qRanGet() will be used the developer has to implement this function. The default function throws an error (qERR_NTF_SEED). Please see the user guide for more information

159. qRtcAlarm

```
void qRtcAlarm(
  uint32_t DatTim,       // date-time in internal format
  void (*pFbr)(void *p));// The function to call
  void (*pFbr)());       // The parameter
```

Before Start	Thread	Critical Section	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
unsigned DatTim	The DatTim in internal format
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

Error conditions

Error	Description
qERR_RTC_ISR	The function is called from an ISR.
qERR_RTC_FUNCTION	No function specified.
qERR_RTC_1_1_2010	The date time is before 1/1/2010

160. qRtcGetDatTim

```
uint32_t qRtcGetDatTim();    //
```

Before Start	Thread	Critical Section	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 uint32_t	Date time in internal format.

Error conditions

None

161. qRtcGetUptime

```
uint32_t qRtcGetUptime();    //
```

Before Start	Thread	Critical Section	Fiber	ISR	Possible preemption

Description

This function returns the uptime.

Parameters and return value

Parameter	Description
Returns uint32_t	The uptime in seconds

Error conditions

None

162. qRtcSetDatTim

```
void qRtcSetDatTim(
   uint32_t DatTim);          //
```

Before Start	Thread	Critical Section	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
uint32_t 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_CRITICAL	The function is called from a critical section
qERR_RTC_1_1_2010	The specified date time is before 1/1/2010

163. qSemAcquire

```
unsigned qSemAcquire(
  pSEM pSem);         // Semaphore to acquire the permit
```

Before Start	Thread	Critical Section	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 functions 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 unsigned	Returns 0 if timed out and 1 if permit is granted
pSEM 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_NO_NAME	Semaphores without a name can't be opened.
qERR_SEM_CRITICAL	This function cannot be called from within a critical section.

164. qSemAcquireFast

```
unsigned qSemAcquireFast(
  pSEM pSem);         // Semaphore to acquire the permit
```

Before Start	Thread	Critical Section	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 functions 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.

This function operates without parameter checking.

Parameters and return value

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

Error conditions

None

165. qSemAcquireNB

```
unsigned qSemAcquireNB(
  pSEM pSem);         // Semaphore to acquire the permit
```

Before Start	Thread	Critical Section	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 qSemAcquire() is already used in another place in the application because using both functions doubles the flash footprint.

Parameters and return value

Parameter	Description
Returns unsigned	Returns 0 if timed out and 1 if permit is granted
pSEM 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.

166. qSemAcquireTO

```
unsigned qSemAcquireTO(
  pSEM pSem,         // Semaphore to acquire the permit
  int32_t TimeOut);  // The timeout
```

Before Start	Thread	Critical Section	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 functions 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 unsigned	Returns 0 if timed out and 1 if permit is granted
pSEM pSem	A pointer to the semaphore object. Must be returned from qSemCreate() or qSemOpen() with the correct name.
int32_t TimeOut	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. 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.

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_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 µSecond.

167. qSemClose

```
void qSemClose(
  pSEM pSem);          // The semphore to close EventSet
```

Before Start	Thread	Critical Section	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
pSEM 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.

168. qSemCreate

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

Before Start	Thread	Critical Section	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 pSEM	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.
unsigned 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.

169. qSemOpen

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

Before Start	Thread	Critical Section	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 pSEM	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_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.

170. qSemOpenNB

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

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns a pointer to an existing semaphore object.

Parameters and return value

Parameter	Description
Returns pSEM	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.

171. qSemOpenTO

```
pSEM qSemOpenTO(              // Retuns NULL when timed-out
  char *pName,                // The Name of the object
  int32_t TimeOut);            // The timeout
```

Before Start	Thread	Critical Section	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 exists the threads 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 pSEM	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.
int32_t TimeOut	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. 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.

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 µSecond.
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.

172. qSemPermits

```
unsigned qSemPermits(
  pSEM pSem);      // The semaphore to get permits for
```

Before Start	Thread	Critical Section	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 unsigned	The function returns the number of permits available.
pSEM 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.

173. qSemRelease

```
void qSemRelease(
  pSEM pSem);           // The semaphore that contains
                        // the permit to release
```

Before Start	Thread	Critical Section	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
pSEM 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.

174. qSemReleaseFast

```
void qSemReleaseFast(
  pSEM pSem);           // The semaphore that contains
                        // the permit to release
```

Before Start	Thread	Critical Section	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.

This function operates without parameter checking.

Permits can be released from Interrupt Service Routines (ISR)

Parameters and return value

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

Error conditions

None

175. qThrClose

```
void qThrClose(
  pTCB pTcb)              // The thread to close
```

Before Start	Thread	Critical Section	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 it 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
pTCB 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_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.

176. qThrCreate

```
pTCB qThrCreate(
  char *pName,              // The Name of the thread
  void(*pFun)(void *pPar), // The thread function itself
  void *pPar,               // The parameter data
  unsigned StackSize,       // The size of the stack
  uint8_t Priority);        // The priority
```

Before Start	Thread	Critical Section	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 pTCB	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.
unsigned StackSize	The size of the stack. This memory is allocated from variable memory.
uint8_t 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 then 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.

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

177. qThrCreateEds (Only 16bit PIC's with EDS)

```
pTCB qThrCreateEds(
  char *pName,              // The Name of the thread
  void(*pFun)(void *pPar), // The thread function itself
  void *pPar,               // The parameter data
  unsigned StackSize,       // The size of the stack
  uint8_t Priority,         // The priority
  __eds__ unsigned* Stack);// Pointer to the stack
```

Before Start	Thread	Critical Section	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 pTCB	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.
unsigned StackSize	The size of the stack. This memory is allocated from variable memory.
uint8_t Priority	The priority from 1 to 250. The higher the number the higher the priority.
__eds__ unsigned* Stack	A pointer to the stack. Its the developer responsibility to manage the EDS memory and define the correct stack

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_STACK	The stack size is smaller then 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

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

178. qThrCreateSuspended

```
pTCB qThrCreate(
  char *pName,              // The Name of the thread
  void(*pFun)(void *pPar), // The thread function itself
  void *pPar,               // The parameter data
  unsigned StackSize,       // The size of the stack
  uint8_t Priority);        // The priority
```

Before Start	Thread	Critical Section	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 pTCB	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.
unsigned StackSize	The size of the stack. This memory is allocated from variable memory.
uint8_t 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 then 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.

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

179. qThrCreateSuspendedEds (PIC's with EDS)

```
pTCB qThrCreateSuspendedEds(
  char *pName,              // The Name of the thread
  void(*pFun)(void *pPar), // The thread function itself
  void *pPar,               // The parameter data
  unsigned StackSize,       // The size of the stack
  uint8_t Priority);        // The priority
  __eds__ unsigned* Stack);// Pointer to the stack
```

Before Start	Thread	Critical Section	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 pTCB	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.
unsigned StackSize	The size of the stack. This memory is allocated from variable memory.
uint8_t Priority	The priority from 1 to 250. The higher the number the higher the priority.
__eds__ unsigned* Stack	A pointer to the stack. Its the developer responsibility to manage the EDS memory and define the correct stack

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_STACK	The stack size is smaller then 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.

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

180. qThrCurrent

```
pTCB qThrCurrent();   // Return the current TCB
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function will return a pointer to the current TCB.

Parameters

Parameter	Description
Returns pTCB	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.

181. qThrEvtClear

```
unsigned qThrEvtClear(
  pTCB pTcb,              // The thread event to clear
  unsigned EventFlags); // The flags to clear
```

Before Start	Thread	Critical Section	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:

```
unsigned flags;                          // variable to return the
flags
pTCB p;                        // Set by create or open
flags = qThrEvtClear(p,0)     // Null does not clear anything. It
```

Parameters and return value

Parameter	Description
Returns unsigned	Returns the event flags before the clear operation.
pTCB 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.
unsigned 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.

182. qThrEvtSignal

```
void qThrEvtSignal(
  qtTHR pThr,           // The thread event set to signal
  unsigned EventFlags);// The flags to set
```

Before Start	Thread	Critical Section	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.

If this function is called from an ISR 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
pTCB 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.
unsigned 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

183. qThrEvtWait

```
unsigned qThrEvtWait(
  unsigned EventFlags,  // The flags to wait for
  uint8_t WaitType);    // The type of wait (see below)
```

Before Start	Thread	Critical Section	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 unsigned	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.
unsigned EventFlags	The event flags to wait for. At least one flag must be set.
uint8_t WaitType	The type of wait. The following are defined. WAIT_TYPE_ALL means wait until all flags are set. This is also called the AND scenario. WAIT_TYPE_ALL_CLEAR means wait until all flags are set and if this situation occurs reset the flags that the thread was waiting for. WAIT_TYPE_ANY means wait until one of the flags is set. This is also called the OR scenario. WAIT_TYPE_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_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_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.

184. qThrEvtWaitNB

```
unsigned qThrEvtWaitNB(
  unsigned EventFlags,    // The flags to wait for
  uint8_t WaitType);      // The type of wait (see below)
```

Before Start	Thread	Critical Section	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 unsigned	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.
unsigned EventFlags	The event flags to wait for. At least one flag must be set.
uint8_t WaitType	The type of wait. The following are defined. WAIT_TYPE_ALL means wait until all flags are set. This is also called the AND scenario. WAIT_TYPE_ALL_CLEAR means wait until all flags are set and if this situation occurs reset the flags that the thread was waiting for. WAIT_TYPE_ANY means wait until one of the flags is set. This is also called the OR scenario. WAIT_TYPE_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_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

185. qThrEvtWaitTO

```
unsigned qThrEvtWaitTO(
  unsigned EventFlags, // The flags to wait for
  uint8_t WaitType,    // The type of wait (see below)
  int32_t TimeOut);     // The time out
```

Before Start	Thread	Critical Section	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 unsigned	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.
unsigned EventFlags	The event flags to wait for. At least one flag must be set.
uint8_t WaitType	The type of wait. The following are defined. WAIT_TYPE_ALL means wait until all flags are set. This is also called the AND scenario. WAIT_TYPE_ALL_CLEAR means wait until all flags are set and if this situation occurs reset the flags that the thread was waiting for. WAIT_TYPE_ANY means wait until one of the flags is set. This is also called the OR scenario. WAIT_TYPE_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.

Parameter	Description
int32_t TimeOut	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. 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.

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_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 µSecond.

186. qThrOpen

```
pTCB qThrOpen(          // Retuns NULL when timed-out
  char *pName);         // The Name of the thread
```

Before Start	Thread	Critical Section	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 pTCB	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_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.

187. qThrOpenNB

```
pTCB qThrOpenNB(          //
  char *pName);           // The name of the thread
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns a pointer to an existing thread object.

Parameters and return value

Parameter	Description
Returns pTCB	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.

188. qThrOpenTO

```
pTCB qThrOpenTO(        // Retuns NULL when timed-out
  char *pName,          // The Name of the object
  int32_t TimeOut);      // The timeout
```

Before Start	Thread	Critical Section	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 exists the threads 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 pTCB	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.
int32_t TimeOut	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. 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.

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 µSecond.
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.

189. qThrResume

```
void qThrResume(     // Wake-up a thread
  pTCB pTcb,         // The TCB of the thread to wake-up
  int reason);       // The reason why the thread should
                     // be resumed
```

Before Start	Thread	Critical Section	Fiber	ISR	Possible preemption

Description

This function resumes a thread that waits or is suspended. If the thread is not sleeping this function has no effect and does not set an error condition.

If this function is called from an ISR 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
pTCB 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.
int reason	The reason why the thread will be resumed. The qThrSuspend() and the qThrSleep() function will return the reason after they have been resumed. The value must be unequal to -1.

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_REASON	The reason is -1

190. qThrResumeV4

```
void qThrResumeV4(      // Wake-up a thread
  pTCB pTcb);
```

Before Start	Thread	Critical Section	Fiber	ISR	Possible preemption

Description

This function resumes a suspended thread from an interrupt, fiber or thread. This function does not has the ability to communicate a value to the resume function. You have to use qThrSuspendV4() in conjection with this function

Parameters and return value

Parameter	Description
pTCB 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

This function does not test for any errors

191. qThrSetPriority

```
void qThrSetPriority(  // Set the priority for a thread
  pTCB pTcb,           // The TCB of the thread
  uint8_t Priority);   // The new priority
```

Before Start	Thread	Critical Section	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
pTCB 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.
uint8_t 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_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.

192. qThrSleep

```
int qThrSleep(        // Let the thread sleep
  int32_t Time);       // Sleeptime in milliseconds
```

Before Start	Thread	Critical Section	Fiber	ISR	Always preemption

Description

This function lets the current thread sleep for a specified time and returns an indication if and why it was resumed.

Parameters and return value

Parameter	Description
Returns unsigned	The reason why the thread was resumed. The function returns -1 if it was not resumed. All other values means that the thread was resumed.
int32_t Time	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. 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.

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_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 µSecond.
qERR_THR_CRITICAL	The function is called within a critical section.

193. qThrStack

```
unsigned qThrStack(// Returns the stack-size in use
  pTCB pTcb);      // The TCB of the thread
```

Before Start	Thread	Critical Section	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 used in production systems. This function is normally used in debug sessions.

Parameters and return value

Parameter	Description
Returns unsigned	This function returns the maximum number of bytes that has been in use on the stack until now.
pTCB 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.

194. qThrStatCycles

```
uint64_t qThrStatCycles(
  pTCB pTcb);                    // The TCB of the thread
```

Before Start	Thread	Critical Section	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 uint64_t	The number of cycles
pTCB 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

195. qThrSuspend

```
int qThrSuspend();    // Returns reason
```

Before Start	Thread	Critical Section	Fiber	ISR	Always 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
Returns int	The reason why the thread was resumed.

Error conditions

Error	Description
qERR_THR_NO_START	The Kernel is not yet started.
qERR_THR_ISR	The function is called from an ISR.
qERR_THR_FIBER	The function tries to suspend while in a fiber.
qERR_THR_CRITICAL	The function is called while in a critical section.

196. qThrSuspendV4

```
void qThrSuspendV4();
```

Before Start	Thread	Critical Section	Fiber	ISR	Always preemption

Description

This function suspend a running. The thread can only be activated by the qThrResumeV4() function.

Parameters and return value

No parameters and no return value

Error conditions

No parameters or state is tested

197. qThrTagGet

```
unsigned qThrTagGet(
  unsigned Tag);       // The tag to set
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns the tag form the thread control block of the current tread.

Parameters and return value

Parameter	Description
Returns unsigned	The existing tag.

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR

198. qThrTagSet

```
unsigned qThrTagSet(
  unsigned Tag);       // The tag to set
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function set a tag in the thread control block of the current tread and returns the existing tag.

Parameters and return value

Parameter	Description
Returns unsigned	The existing tag before it was over-written by the new tag.
unsigned Tag	The tag to set

Error conditions

Error	Description
qERR_THR_ISR	The function is called from an ISR

199. qThrTracking

```
void qThrTrack(
  pTCB pTcb,        // The TCB of the thread
  unsigned *pAddr, // The address of the bit to set/clr
  unsigned BitNbr);// The bit number
```

Before Start	Thread	Critical Section	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
pTCB 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.
unsigned *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.
unsigned 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

200. qThrYield

```
void qThrYield();      // Yield the thread
```

Before Start	Thread	Critical Section	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.

201. qTimCycles

```
uint64_t qTimCycles();// Returns up-time in cycles
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns the number of cycles that the system is up. All time funcions are based on cycles and an MCU that run at 100MHz overflows after 5,840 years so this function has a large range.

This function requires the kernel keep track off individual cycles which it can then convert to uSec. See the function qKrnUSecOn().

Parameters and return value

Parameter	Description
Returns uint64_t	The number of cycles 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 µSeconds is not enabled. Use qKrnUSecOn() to start the µSecond tracking.

202. qTimMSec

```
uint64_t qTimMSec();// Returns system up-time in mSec
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns the number of mSec that the system is up. All time funcions are based on cycles and an MCU that run at 100MHz overflows after 5,840 years so this function has a large range.

This function requires the kernel keep track off individual cycles which it can then convert to uSec. See the function qKrnUSecOn().

Parameters and return value

Parameter	Description
Returns uint64_t	The number of mSec 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 µSeconds is not enabled. Use qKrnUSecOn() to start the µSecond tracking.

203. qTimUSec

```
uint64_t qTimUSec();// Returns system up-time in µSec
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns the number of µSec that the system is up. All time funcions are based on cycles and an MCU that run at 100MHz overflows after 5,840 years so this function has a large range.

This function requires the kernel keep track off individual cycles which it can then convert to uSec. See the function qKrnUSecOn().

Parameters and return value

Parameter	Description
Returns uint64_t	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 µSeconds is not enabled. Use qKrnUSecOn() to start the µSecond tracking.

204. qTmrClose

```
void qTmrClose(
  pTMR pTmr);                  // The timer to close
```

Before Start	Thread	Critical Section	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.

Parameters and return value

Parameter	Description
pTMR pTmr	A pointer to the object. Must be returned from the qTmrCreate() or qTmrOpen() function 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.

205. qTmrCreate

```
pTMR qTmrCreate(
  char *pName,              // The Name of the timer
  void (*pFbr)(void *p),    // The function to call
  void *pParam,             // The parameter when called
  TMR_TYPE TimerType,       // Timer options
  int32_t Time);            // The number of seconds
                            // or cycles before the
                            // timer signals the event
```

Before Start	Thread	Critical Section	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 pTMR	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
pTMR_TYPE TimerType	The timer type options. • qTMR_TYPE_PERIODIC specifies a periodic timer. • qTMR_TYPE_ONE_SHOT specifies a one shot timer. • qTMR_TYPE_PERIODIC_MANUAL_START specifies a periodic timer that must be started manual. • qTMR_TYPE_ONE_SHOT_MANUAL_START specifies a one shot timer that must be started manual.
int32_t Time	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. 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.

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.

206. qTmrOpen

```
pTMR qTmrOpen(                  // Retuns NULL when timed-out
  char *pName);                 // The Name of the timer
```

Before Start	Thread	Critical Section	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 exists the threads 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 pTMR	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, 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_NO_NAME	Timers without a name can't be opened.
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.

207. qTmrOpenNB

```
pTMR qTmrOpenNB(          //
  char *pName);           // The Name of the timer
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function returns a pointer to an existing timer object.

Parameters and return value

Parameter	Description
Returns pTMR	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.

208. qTmrOpenTO

```
pTMR qTmrOpenTO(              // Retuns NULL when timed-out
  char *pName,                 // The Name of the object
  int32_t TimeOut);             // The timeout
```

Before Start	Thread	Critical Section	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 exists the threads 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 pTMR	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.
int32_t TimeOut	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. 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.

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 µSecond.
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.

209. qTmrStart

```
void qTmrStart(
  pTMR pTmr);                  // The timer to start
```

Before Start	Thread	Critical Section	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.

If this function is called from an ISR 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
pTMR pTmr	A pointer to the timer object. Must be returned from qTmrCreate() or qTmrOpen() with the correct name.

Error conditions

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

210. qTmrStop

```
void qTmrStop(
  pTMR pTmr);                    // The timer to stop
```

Before Start	Thread	Critical Section	Fiber	ISR	Possible preemption

Description

This function stops a timer after it expires.

Parameters and return value

Parameter	Description
pTMR qTmr	A pointer to the timer object. Must be returned from qTmrCreate() or qTmrOpen() with the correct name.

Error conditions

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

211. qWrdClr

```
void qWrdClr(
  unsigned* From,    // From address
  unsigned Len);     // Length in number of bytes
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function clears (filled with all 0 bits) an array of unsigned intergers. These functions are optimized for the type of processor and operate much faster then the C functions.

Parameters and return value

Parameter	Description
unsigned* From	From address
unsigned Len	The length in bytes

Error conditions

None

212. qWrdDecAtomic

```
void qWrdDecAtomic(
  unsigned* p)      // Address for the operation
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function decrement the integer location pointed by p atomiccaly. An atomic operation is an operation that will always be executed without any other thread, fiber or interrupt being able to change the value during the operation.

Parameters and return value

Parameter	Description
unsigned* p	Pointer to the location of the operation

Error conditions

None

213. qWrdIncAtomic

```
void qWrdIncAtomic(
  unsigned* p)      // Address for the operation
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function increment the integer location pointed by p atomiccaly. An atomic operation is an operation that will always be executed without any other thread, fiber or interrupt being able to read or change the value during the operation.

Parameters and return value

Parameter	Description
unsigned* p	Pointer to the location of the operation

Error conditions

None

214. qWrdMov

```
void qWrdMov(
  unsigned* From,    // From address
  unsigned* To,      // To address
  unsigned Len);     // Length in number of bytes
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function moves unsigned intergers from a From location to a To location. Both locations can not overlap. This is not tested and the responsibility of the developer.

These functions are optimized for the type of processor and operate much faster then the C functions.

Parameters and return value

Parameter	Description
unsigned* From	From address
unsigned* To	To address
unsigned Len	The length in bytes

Error conditions

None

215. qWrdSet

```
void qWrdSet(
  unsigned* From,    // From address
  unsigned Len);     // Length in number of bytes
```

Before Start	Thread	Critical Section	Fiber	ISR	No preemption

Description

This function set (filled with all 1 bits) an array of unsigned intergers. These functions are optimized for the type of processor and operate much faster then the C functions.

Parameters and return value

Parameter	Description
unsigned* From	From address
unsigned Len	The length in bytes

Error conditions

None

216. 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_TIMEOUT	The specified TimeOut is smaller than 1 µSecond.	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
qERR_EVT_NO_FLAGS	Not one flag in EventFlags is set.	0x1111
qERR_EVT_STACK_LIMIT_1		0x1112
qERR_EVT_STACK_LIMIT_2		0x1113

Error	Description	Error#
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 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_MEMORY	There is no variable memory available to handle the request.	0x131B
qERR_KRN_MHZ		0x131C

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 can own mutexes.	0x1603
qERR_MTX_TIMEOUT	The specified TimeOut is smaller than 1 µSecond.	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_TIMEOUT	The specified TimeOut is smaller than 1 µSecond.	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_TIMEOUT	The specified TimeOut is smaller than 1 µSecond.	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 of 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_TIMEOUT	The specified TimeOut is smaller than 1 µSecond.	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 of 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_TIMEOUT	The specified TimeOut is smaller than 1 µSecond.	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
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

Error	Description	Error#
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_TIMEOUT	The specified TimeOut is smaller than 1 µSecond.	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
qERR_TMR_TIME	The time is incorrect.	0x1B14

Error	Description	Error#
qERR_TMR_NO_USEC	The code that keeps track of µSeconds is not enabled. Use qKrnUSecOn() to start the µSecond 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_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_TIMEOUT	The specified TimeOut is smaller than 1 µSecond.	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