/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* uC/OS-II

\* The Real-Time Kernel

\* MESSAGE QUEUE MANAGEMENT

\*

\* (c) Copyright 1992-2009, Micrium, Weston, FL

\* All Rights Reserved

\*

\* File : OS\_Q.C

\* By : Jean J. Labrosse

\* Version : V2.91

\*

\* LICENSING TERMS:

\* ---------------

\* uC/OS-II is provided in source form for FREE evaluation, for educational use or for peaceful research.

\* If you plan on using uC/OS-II in a commercial product you need to contact Micri祄 to properly license

\* its use in your product. We provide ALL the source code for your convenience and to help you experience

\* uC/OS-II. The fact that the source is provided does NOT mean that you can use it without paying a

\* licensing fee.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#ifndef OS\_MASTER\_FILE

#include <ucos\_ii.h>

#endif

#if (OS\_Q\_EN > 0u) && (OS\_MAX\_QS > 0u)

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ACCEPT MESSAGE FROM QUEUE

\*

\* Description: This function checks the queue to see if a message is available. Unlike OSQPend(),

\* OSQAccept() does not suspend the calling task if a message is not available.

\*

\* Arguments : pevent is a pointer to the event control block

\*

\* perr is a pointer to where an error message will be deposited. Possible error

\* messages are:

\*

\* OS\_ERR\_NONE The call was successful and your task received a

\* message.

\* OS\_ERR\_EVENT\_TYPE You didn't pass a pointer to a queue

\* OS\_ERR\_PEVENT\_NULL If 'pevent' is a NULL pointer

\* OS\_ERR\_Q\_EMPTY The queue did not contain any messages

\*

\* Returns : != (void \*)0 is the message in the queue if one is available. The message is removed

\* from the so the next time OSQAccept() is called, the queue will contain

\* one less entry.

\* == (void \*)0 if you received a NULL pointer message

\* if the queue is empty or,

\* if 'pevent' is a NULL pointer or,

\* if you passed an invalid event type

\*

\* Note(s) : As of V2.60, you can now pass NULL pointers through queues. Because of this, the argument

\* 'perr' has been added to the API to tell you about the outcome of the call.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_Q\_ACCEPT\_EN > 0u

void \*OSQAccept (OS\_EVENT \*pevent,

INT8U \*perr)

{

void \*pmsg;

OS\_Q \*pq;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#ifdef OS\_SAFETY\_CRITICAL

if (perr == (INT8U \*)0) {

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

\*perr = OS\_ERR\_PEVENT\_NULL;

return ((void \*)0);

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_Q) {/\* Validate event block type \*/

\*perr = OS\_ERR\_EVENT\_TYPE;

return ((void \*)0);

}

OS\_ENTER\_CRITICAL();

pq = (OS\_Q \*)pevent->OSEventPtr; /\* Point at queue control block \*/

if (pq->OSQEntries > 0u) { /\* See if any messages in the queue \*/

pmsg = \*pq->OSQOut++; /\* Yes, extract oldest message from the queue \*/

pq->OSQEntries--; /\* Update the number of entries in the queue \*/

if (pq->OSQOut == pq->OSQEnd) { /\* Wrap OUT pointer if we are at the end of the queue \*/

pq->OSQOut = pq->OSQStart;

}

\*perr = OS\_ERR\_NONE;

} else {

\*perr = OS\_ERR\_Q\_EMPTY;

pmsg = (void \*)0; /\* Queue is empty \*/

}

OS\_EXIT\_CRITICAL();

return (pmsg); /\* Return message received (or NULL) \*/

}

#endif

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* CREATE A MESSAGE QUEUE

\*

\* Description: This function creates a message queue if free event control blocks are available.

\*

\* Arguments : start is a pointer to the base address of the message queue storage area. The

\* storage area MUST be declared as an array of pointers to 'void' as follows

\*

\* void \*MessageStorage[size]

\*

\* size is the number of elements in the storage area

\*

\* Returns : != (OS\_EVENT \*)0 is a pointer to the event control clock (OS\_EVENT) associated with the

\* created queue

\* == (OS\_EVENT \*)0 if no event control blocks were available or an error was detected

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

OS\_EVENT \*OSQCreate (void \*\*start,

INT16U size)

{

OS\_EVENT \*pevent;

OS\_Q \*pq;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#ifdef OS\_SAFETY\_CRITICAL\_IEC61508

if (OSSafetyCriticalStartFlag == OS\_TRUE) {

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

if (OSIntNesting > 0u) { /\* See if called from ISR ... \*/

return ((OS\_EVENT \*)0); /\* ... can't CREATE from an ISR \*/

}

OS\_ENTER\_CRITICAL();

pevent = OSEventFreeList; /\* Get next free event control block \*/

if (OSEventFreeList != (OS\_EVENT \*)0) { /\* See if pool of free ECB pool was empty \*/

OSEventFreeList = (OS\_EVENT \*)OSEventFreeList->OSEventPtr;

}

OS\_EXIT\_CRITICAL();

if (pevent != (OS\_EVENT \*)0) { /\* See if we have an event control block \*/

OS\_ENTER\_CRITICAL();

pq = OSQFreeList; /\* Get a free queue control block \*/

if (pq != (OS\_Q \*)0) { /\* Were we able to get a queue control block ? \*/

OSQFreeList = OSQFreeList->OSQPtr; /\* Yes, Adjust free list pointer to next free\*/

OS\_EXIT\_CRITICAL();

pq->OSQStart = start; /\* Initialize the queue \*/

pq->OSQEnd = &start[size];

pq->OSQIn = start;

pq->OSQOut = start;

pq->OSQSize = size;

pq->OSQEntries = 0u;

pevent->OSEventType = OS\_EVENT\_TYPE\_Q;

pevent->OSEventCnt = 0u;

pevent->OSEventPtr = pq;

#if OS\_EVENT\_NAME\_EN > 0u

pevent->OSEventName = (INT8U \*)(void \*)"?";

#endif

OS\_EventWaitListInit(pevent); /\* Initalize the wait list \*/

} else {

pevent->OSEventPtr = (void \*)OSEventFreeList; /\* No, Return event control block on error \*/

OSEventFreeList = pevent;

OS\_EXIT\_CRITICAL();

pevent = (OS\_EVENT \*)0;

}

}

return (pevent);

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DELETE A MESSAGE QUEUE

\*

\* Description: This function deletes a message queue and readies all tasks pending on the queue.

\*

\* Arguments : pevent is a pointer to the event control block associated with the desired

\* queue.

\*

\* opt determines delete options as follows:

\* opt == OS\_DEL\_NO\_PEND Delete the queue ONLY if no task pending

\* opt == OS\_DEL\_ALWAYS Deletes the queue even if tasks are waiting.

\* In this case, all the tasks pending will be readied.

\*

\* perr is a pointer to an error code that can contain one of the following values:

\* OS\_ERR\_NONE The call was successful and the queue was deleted

\* OS\_ERR\_DEL\_ISR If you tried to delete the queue from an ISR

\* OS\_ERR\_INVALID\_OPT An invalid option was specified

\* OS\_ERR\_TASK\_WAITING One or more tasks were waiting on the queue

\* OS\_ERR\_EVENT\_TYPE If you didn't pass a pointer to a queue

\* OS\_ERR\_PEVENT\_NULL If 'pevent' is a NULL pointer.

\*

\* Returns : pevent upon error

\* (OS\_EVENT \*)0 if the queue was successfully deleted.

\*

\* Note(s) : 1) This function must be used with care. Tasks that would normally expect the presence of

\* the queue MUST check the return code of OSQPend().

\* 2) OSQAccept() callers will not know that the intended queue has been deleted unless

\* they check 'pevent' to see that it's a NULL pointer.

\* 3) This call can potentially disable interrupts for a long time. The interrupt disable

\* time is directly proportional to the number of tasks waiting on the queue.

\* 4) Because ALL tasks pending on the queue will be readied, you MUST be careful in

\* applications where the queue is used for mutual exclusion because the resource(s)

\* will no longer be guarded by the queue.

\* 5) If the storage for the message queue was allocated dynamically (i.e. using a malloc()

\* type call) then your application MUST release the memory storage by call the counterpart

\* call of the dynamic allocation scheme used. If the queue storage was created statically

\* then, the storage can be reused.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_Q\_DEL\_EN > 0u

OS\_EVENT \*OSQDel (OS\_EVENT \*pevent,

INT8U opt,

INT8U \*perr)

{

BOOLEAN tasks\_waiting;

OS\_EVENT \*pevent\_return;

OS\_Q \*pq;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#ifdef OS\_SAFETY\_CRITICAL

if (perr == (INT8U \*)0) {

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

\*perr = OS\_ERR\_PEVENT\_NULL;

return (pevent);

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_Q) { /\* Validate event block type \*/

\*perr = OS\_ERR\_EVENT\_TYPE;

return (pevent);

}

if (OSIntNesting > 0u) { /\* See if called from ISR ... \*/

\*perr = OS\_ERR\_DEL\_ISR; /\* ... can't DELETE from an ISR \*/

return (pevent);

}

OS\_ENTER\_CRITICAL();

if (pevent->OSEventGrp != 0u) { /\* See if any tasks waiting on queue \*/

tasks\_waiting = OS\_TRUE; /\* Yes \*/

} else {

tasks\_waiting = OS\_FALSE; /\* No \*/

}

switch (opt) {

case OS\_DEL\_NO\_PEND: /\* Delete queue only if no task waiting \*/

if (tasks\_waiting == OS\_FALSE) {

#if OS\_EVENT\_NAME\_EN > 0u

pevent->OSEventName = (INT8U \*)(void \*)"?";

#endif

pq = (OS\_Q \*)pevent->OSEventPtr; /\* Return OS\_Q to free list \*/

pq->OSQPtr = OSQFreeList;

OSQFreeList = pq;

pevent->OSEventType = OS\_EVENT\_TYPE\_UNUSED;

pevent->OSEventPtr = OSEventFreeList; /\* Return Event Control Block to free list \*/

pevent->OSEventCnt = 0u;

OSEventFreeList = pevent; /\* Get next free event control block \*/

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

pevent\_return = (OS\_EVENT \*)0; /\* Queue has been deleted \*/

} else {

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_TASK\_WAITING;

pevent\_return = pevent;

}

break;

case OS\_DEL\_ALWAYS: /\* Always delete the queue \*/

while (pevent->OSEventGrp != 0u) { /\* Ready ALL tasks waiting for queue \*/

(void)OS\_EventTaskRdy(pevent, (void \*)0, OS\_STAT\_Q, OS\_STAT\_PEND\_OK);

}

#if OS\_EVENT\_NAME\_EN > 0u

pevent->OSEventName = (INT8U \*)(void \*)"?";

#endif

pq = (OS\_Q \*)pevent->OSEventPtr; /\* Return OS\_Q to free list \*/

pq->OSQPtr = OSQFreeList;

OSQFreeList = pq;

pevent->OSEventType = OS\_EVENT\_TYPE\_UNUSED;

pevent->OSEventPtr = OSEventFreeList; /\* Return Event Control Block to free list \*/

pevent->OSEventCnt = 0u;

OSEventFreeList = pevent; /\* Get next free event control block \*/

OS\_EXIT\_CRITICAL();

if (tasks\_waiting == OS\_TRUE) { /\* Reschedule only if task(s) were waiting \*/

OS\_Sched(); /\* Find highest priority task ready to run \*/

}

\*perr = OS\_ERR\_NONE;

pevent\_return = (OS\_EVENT \*)0; /\* Queue has been deleted \*/

break;

default:

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_INVALID\_OPT;

pevent\_return = pevent;

break;

}

return (pevent\_return);

}

#endif

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FLUSH QUEUE

\*

\* Description : This function is used to flush the contents of the message queue.

\*

\* Arguments : none

\*

\* Returns : OS\_ERR\_NONE upon success

\* OS\_ERR\_EVENT\_TYPE If you didn't pass a pointer to a queue

\* OS\_ERR\_PEVENT\_NULL If 'pevent' is a NULL pointer

\*

\* WARNING : You should use this function with great care because, when to flush the queue, you LOOSE

\* the references to what the queue entries are pointing to and thus, you could cause

\* 'memory leaks'. In other words, the data you are pointing to that's being referenced

\* by the queue entries should, most likely, need to be de-allocated (i.e. freed).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_Q\_FLUSH\_EN > 0u

INT8U OSQFlush (OS\_EVENT \*pevent)

{

OS\_Q \*pq;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

return (OS\_ERR\_PEVENT\_NULL);

}

if (pevent->OSEventType != OS\_EVENT\_TYPE\_Q) { /\* Validate event block type \*/

return (OS\_ERR\_EVENT\_TYPE);

}

#endif

OS\_ENTER\_CRITICAL();

pq = (OS\_Q \*)pevent->OSEventPtr; /\* Point to queue storage structure \*/

pq->OSQIn = pq->OSQStart;

pq->OSQOut = pq->OSQStart;

pq->OSQEntries = 0u;

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

#endif

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PEND ON A QUEUE FOR A MESSAGE

\*

\* Description: This function waits for a message to be sent to a queue

\*

\* Arguments : pevent is a pointer to the event control block associated with the desired queue

\*

\* timeout is an optional timeout period (in clock ticks). If non-zero, your task will

\* wait for a message to arrive at the queue up to the amount of time

\* specified by this argument. If you specify 0, however, your task will wait

\* forever at the specified queue or, until a message arrives.

\*

\* perr is a pointer to where an error message will be deposited. Possible error

\* messages are:

\*

\* OS\_ERR\_NONE The call was successful and your task received a

\* message.

\* OS\_ERR\_TIMEOUT A message was not received within the specified 'timeout'.

\* OS\_ERR\_PEND\_ABORT The wait on the queue was aborted.

\* OS\_ERR\_EVENT\_TYPE You didn't pass a pointer to a queue

\* OS\_ERR\_PEVENT\_NULL If 'pevent' is a NULL pointer

\* OS\_ERR\_PEND\_ISR If you called this function from an ISR and the result

\* would lead to a suspension.

\* OS\_ERR\_PEND\_LOCKED If you called this function with the scheduler is locked

\*

\* Returns : != (void \*)0 is a pointer to the message received

\* == (void \*)0 if you received a NULL pointer message or,

\* if no message was received or,

\* if 'pevent' is a NULL pointer or,

\* if you didn't pass a pointer to a queue.

\*

\* Note(s) : As of V2.60, this function allows you to receive NULL pointer messages.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void \*OSQPend (OS\_EVENT \*pevent,

INT32U timeout,

INT8U \*perr)

{

void \*pmsg;

OS\_Q \*pq;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#ifdef OS\_SAFETY\_CRITICAL

if (perr == (INT8U \*)0) {

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

\*perr = OS\_ERR\_PEVENT\_NULL;

return ((void \*)0);

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_Q) {/\* Validate event block type \*/

\*perr = OS\_ERR\_EVENT\_TYPE;

return ((void \*)0);

}

if (OSIntNesting > 0u) { /\* See if called from ISR ... \*/

\*perr = OS\_ERR\_PEND\_ISR; /\* ... can't PEND from an ISR \*/

return ((void \*)0);

}

if (OSLockNesting > 0u) { /\* See if called with scheduler locked ... \*/

\*perr = OS\_ERR\_PEND\_LOCKED; /\* ... can't PEND when locked \*/

return ((void \*)0);

}

OS\_ENTER\_CRITICAL();

pq = (OS\_Q \*)pevent->OSEventPtr; /\* Point at queue control block \*/

if (pq->OSQEntries > 0u) { /\* See if any messages in the queue \*/

pmsg = \*pq->OSQOut++; /\* Yes, extract oldest message from the queue \*/

pq->OSQEntries--; /\* Update the number of entries in the queue \*/

if (pq->OSQOut == pq->OSQEnd) { /\* Wrap OUT pointer if we are at the end of the queue \*/

pq->OSQOut = pq->OSQStart;

}

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

return (pmsg); /\* Return message received \*/

}

OSTCBCur->OSTCBStat |= OS\_STAT\_Q; /\* Task will have to pend for a message to be posted \*/

OSTCBCur->OSTCBStatPend = OS\_STAT\_PEND\_OK;

OSTCBCur->OSTCBDly = timeout; /\* Load timeout into TCB \*/

OS\_EventTaskWait(pevent); /\* Suspend task until event or timeout occurs \*/

OS\_EXIT\_CRITICAL();

OS\_Sched(); /\* Find next highest priority task ready to run \*/

OS\_ENTER\_CRITICAL();

switch (OSTCBCur->OSTCBStatPend) { /\* See if we timed-out or aborted \*/

case OS\_STAT\_PEND\_OK: /\* Extract message from TCB (Put there by QPost) \*/

pmsg = OSTCBCur->OSTCBMsg;

\*perr = OS\_ERR\_NONE;

break;

case OS\_STAT\_PEND\_ABORT:

pmsg = (void \*)0;

\*perr = OS\_ERR\_PEND\_ABORT; /\* Indicate that we aborted \*/

break;

case OS\_STAT\_PEND\_TO:

default:

OS\_EventTaskRemove(OSTCBCur, pevent);

pmsg = (void \*)0;

\*perr = OS\_ERR\_TIMEOUT; /\* Indicate that we didn't get event within TO \*/

break;

}

OSTCBCur->OSTCBStat = OS\_STAT\_RDY; /\* Set task status to ready \*/

OSTCBCur->OSTCBStatPend = OS\_STAT\_PEND\_OK; /\* Clear pend status \*/

OSTCBCur->OSTCBEventPtr = (OS\_EVENT \*)0; /\* Clear event pointers \*/

#if (OS\_EVENT\_MULTI\_EN > 0u)

OSTCBCur->OSTCBEventMultiPtr = (OS\_EVENT \*\*)0;

#endif

OSTCBCur->OSTCBMsg = (void \*)0; /\* Clear received message \*/

OS\_EXIT\_CRITICAL();

return (pmsg); /\* Return received message \*/

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ABORT WAITING ON A MESSAGE QUEUE

\*

\* Description: This function aborts & readies any tasks currently waiting on a queue. This function

\* should be used to fault-abort the wait on the queue, rather than to normally signal

\* the queue via OSQPost(), OSQPostFront() or OSQPostOpt().

\*

\* Arguments : pevent is a pointer to the event control block associated with the desired queue.

\*

\* opt determines the type of ABORT performed:

\* OS\_PEND\_OPT\_NONE ABORT wait for a single task (HPT) waiting on the

\* queue

\* OS\_PEND\_OPT\_BROADCAST ABORT wait for ALL tasks that are waiting on the

\* queue

\*

\* perr is a pointer to where an error message will be deposited. Possible error

\* messages are:

\*

\* OS\_ERR\_NONE No tasks were waiting on the queue.

\* OS\_ERR\_PEND\_ABORT At least one task waiting on the queue was readied

\* and informed of the aborted wait; check return value

\* for the number of tasks whose wait on the queue

\* was aborted.

\* OS\_ERR\_EVENT\_TYPE If you didn't pass a pointer to a queue.

\* OS\_ERR\_PEVENT\_NULL If 'pevent' is a NULL pointer.

\*

\* Returns : == 0 if no tasks were waiting on the queue, or upon error.

\* > 0 if one or more tasks waiting on the queue are now readied and informed.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_Q\_PEND\_ABORT\_EN > 0u

INT8U OSQPendAbort (OS\_EVENT \*pevent,

INT8U opt,

INT8U \*perr)

{

INT8U nbr\_tasks;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#ifdef OS\_SAFETY\_CRITICAL

if (perr == (INT8U \*)0) {

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

\*perr = OS\_ERR\_PEVENT\_NULL;

return (0u);

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_Q) { /\* Validate event block type \*/

\*perr = OS\_ERR\_EVENT\_TYPE;

return (0u);

}

OS\_ENTER\_CRITICAL();

if (pevent->OSEventGrp != 0u) { /\* See if any task waiting on queue? \*/

nbr\_tasks = 0u;

switch (opt) {

case OS\_PEND\_OPT\_BROADCAST: /\* Do we need to abort ALL waiting tasks? \*/

while (pevent->OSEventGrp != 0u) { /\* Yes, ready ALL tasks waiting on queue \*/

(void)OS\_EventTaskRdy(pevent, (void \*)0, OS\_STAT\_Q, OS\_STAT\_PEND\_ABORT);

nbr\_tasks++;

}

break;

case OS\_PEND\_OPT\_NONE:

default: /\* No, ready HPT waiting on queue \*/

(void)OS\_EventTaskRdy(pevent, (void \*)0, OS\_STAT\_Q, OS\_STAT\_PEND\_ABORT);

nbr\_tasks++;

break;

}

OS\_EXIT\_CRITICAL();

OS\_Sched(); /\* Find HPT ready to run \*/

\*perr = OS\_ERR\_PEND\_ABORT;

return (nbr\_tasks);

}

OS\_EXIT\_CRITICAL();

\*perr = OS\_ERR\_NONE;

return (0u); /\* No tasks waiting on queue \*/

}

#endif

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* POST MESSAGE TO A QUEUE

\*

\* Description: This function sends a message to a queue

\*

\* Arguments : pevent is a pointer to the event control block associated with the desired queue

\*

\* pmsg is a pointer to the message to send.

\*

\* Returns : OS\_ERR\_NONE The call was successful and the message was sent

\* OS\_ERR\_Q\_FULL If the queue cannot accept any more messages because it is full.

\* OS\_ERR\_EVENT\_TYPE If you didn't pass a pointer to a queue.

\* OS\_ERR\_PEVENT\_NULL If 'pevent' is a NULL pointer

\*

\* Note(s) : As of V2.60, this function allows you to send NULL pointer messages.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_Q\_POST\_EN > 0u

INT8U OSQPost (OS\_EVENT \*pevent,

void \*pmsg)

{

OS\_Q \*pq;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

return (OS\_ERR\_PEVENT\_NULL);

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_Q) { /\* Validate event block type \*/

return (OS\_ERR\_EVENT\_TYPE);

}

OS\_ENTER\_CRITICAL();

if (pevent->OSEventGrp != 0u) { /\* See if any task pending on queue \*/

/\* Ready highest priority task waiting on event \*/

(void)OS\_EventTaskRdy(pevent, pmsg, OS\_STAT\_Q, OS\_STAT\_PEND\_OK);

OS\_EXIT\_CRITICAL();

OS\_Sched(); /\* Find highest priority task ready to run \*/

return (OS\_ERR\_NONE);

}

pq = (OS\_Q \*)pevent->OSEventPtr; /\* Point to queue control block \*/

if (pq->OSQEntries >= pq->OSQSize) { /\* Make sure queue is not full \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_Q\_FULL);

}

\*pq->OSQIn++ = pmsg; /\* Insert message into queue \*/

pq->OSQEntries++; /\* Update the nbr of entries in the queue \*/

if (pq->OSQIn == pq->OSQEnd) { /\* Wrap IN ptr if we are at end of queue \*/

pq->OSQIn = pq->OSQStart;

}

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

#endif

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* POST MESSAGE TO THE FRONT OF A QUEUE

\*

\* Description: This function sends a message to a queue but unlike OSQPost(), the message is posted at

\* the front instead of the end of the queue. Using OSQPostFront() allows you to send

\* 'priority' messages.

\*

\* Arguments : pevent is a pointer to the event control block associated with the desired queue

\*

\* pmsg is a pointer to the message to send.

\*

\* Returns : OS\_ERR\_NONE The call was successful and the message was sent

\* OS\_ERR\_Q\_FULL If the queue cannot accept any more messages because it is full.

\* OS\_ERR\_EVENT\_TYPE If you didn't pass a pointer to a queue.

\* OS\_ERR\_PEVENT\_NULL If 'pevent' is a NULL pointer

\*

\* Note(s) : As of V2.60, this function allows you to send NULL pointer messages.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_Q\_POST\_FRONT\_EN > 0u

INT8U OSQPostFront (OS\_EVENT \*pevent,

void \*pmsg)

{

OS\_Q \*pq;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

return (OS\_ERR\_PEVENT\_NULL);

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_Q) { /\* Validate event block type \*/

return (OS\_ERR\_EVENT\_TYPE);

}

OS\_ENTER\_CRITICAL();

if (pevent->OSEventGrp != 0u) { /\* See if any task pending on queue \*/

/\* Ready highest priority task waiting on event \*/

(void)OS\_EventTaskRdy(pevent, pmsg, OS\_STAT\_Q, OS\_STAT\_PEND\_OK);

OS\_EXIT\_CRITICAL();

OS\_Sched(); /\* Find highest priority task ready to run \*/

return (OS\_ERR\_NONE);

}

pq = (OS\_Q \*)pevent->OSEventPtr; /\* Point to queue control block \*/

if (pq->OSQEntries >= pq->OSQSize) { /\* Make sure queue is not full \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_Q\_FULL);

}

if (pq->OSQOut == pq->OSQStart) { /\* Wrap OUT ptr if we are at the 1st queue entry \*/

pq->OSQOut = pq->OSQEnd;

}

pq->OSQOut--;

\*pq->OSQOut = pmsg; /\* Insert message into queue \*/

pq->OSQEntries++; /\* Update the nbr of entries in the queue \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

#endif

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* POST MESSAGE TO A QUEUE

\*

\* Description: This function sends a message to a queue. This call has been added to reduce code size

\* since it can replace both OSQPost() and OSQPostFront(). Also, this function adds the

\* capability to broadcast a message to ALL tasks waiting on the message queue.

\*

\* Arguments : pevent is a pointer to the event control block associated with the desired queue

\*

\* pmsg is a pointer to the message to send.

\*

\* opt determines the type of POST performed:

\* OS\_POST\_OPT\_NONE POST to a single waiting task

\* (Identical to OSQPost())

\* OS\_POST\_OPT\_BROADCAST POST to ALL tasks that are waiting on the queue

\* OS\_POST\_OPT\_FRONT POST as LIFO (Simulates OSQPostFront())

\* OS\_POST\_OPT\_NO\_SCHED Indicates that the scheduler will NOT be invoked

\*

\* Returns : OS\_ERR\_NONE The call was successful and the message was sent

\* OS\_ERR\_Q\_FULL If the queue cannot accept any more messages because it is full.

\* OS\_ERR\_EVENT\_TYPE If you didn't pass a pointer to a queue.

\* OS\_ERR\_PEVENT\_NULL If 'pevent' is a NULL pointer

\*

\* Warning : Interrupts can be disabled for a long time if you do a 'broadcast'. In fact, the

\* interrupt disable time is proportional to the number of tasks waiting on the queue.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_Q\_POST\_OPT\_EN > 0u

INT8U OSQPostOpt (OS\_EVENT \*pevent,

void \*pmsg,

INT8U opt)

{

OS\_Q \*pq;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

return (OS\_ERR\_PEVENT\_NULL);

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_Q) { /\* Validate event block type \*/

return (OS\_ERR\_EVENT\_TYPE);

}

OS\_ENTER\_CRITICAL();

if (pevent->OSEventGrp != 0x00u) { /\* See if any task pending on queue \*/

if ((opt & OS\_POST\_OPT\_BROADCAST) != 0x00u) { /\* Do we need to post msg to ALL waiting tasks ? \*/

while (pevent->OSEventGrp != 0u) { /\* Yes, Post to ALL tasks waiting on queue \*/

(void)OS\_EventTaskRdy(pevent, pmsg, OS\_STAT\_Q, OS\_STAT\_PEND\_OK);

}

} else { /\* No, Post to HPT waiting on queue \*/

(void)OS\_EventTaskRdy(pevent, pmsg, OS\_STAT\_Q, OS\_STAT\_PEND\_OK);

}

OS\_EXIT\_CRITICAL();

if ((opt & OS\_POST\_OPT\_NO\_SCHED) == 0u) { /\* See if scheduler needs to be invoked \*/

OS\_Sched(); /\* Find highest priority task ready to run \*/

}

return (OS\_ERR\_NONE);

}

pq = (OS\_Q \*)pevent->OSEventPtr; /\* Point to queue control block \*/

if (pq->OSQEntries >= pq->OSQSize) { /\* Make sure queue is not full \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_Q\_FULL);

}

if ((opt & OS\_POST\_OPT\_FRONT) != 0x00u) { /\* Do we post to the FRONT of the queue? \*/

if (pq->OSQOut == pq->OSQStart) { /\* Yes, Post as LIFO, Wrap OUT pointer if we ... \*/

pq->OSQOut = pq->OSQEnd; /\* ... are at the 1st queue entry \*/

}

pq->OSQOut--;

\*pq->OSQOut = pmsg; /\* Insert message into queue \*/

} else { /\* No, Post as FIFO \*/

\*pq->OSQIn++ = pmsg; /\* Insert message into queue \*/

if (pq->OSQIn == pq->OSQEnd) { /\* Wrap IN ptr if we are at end of queue \*/

pq->OSQIn = pq->OSQStart;

}

}

pq->OSQEntries++; /\* Update the nbr of entries in the queue \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

#endif

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* QUERY A MESSAGE QUEUE

\*

\* Description: This function obtains information about a message queue.

\*

\* Arguments : pevent is a pointer to the event control block associated with the desired queue

\*

\* p\_q\_data is a pointer to a structure that will contain information about the message

\* queue.

\*

\* Returns : OS\_ERR\_NONE The call was successful and the message was sent

\* OS\_ERR\_EVENT\_TYPE If you are attempting to obtain data from a non queue.

\* OS\_ERR\_PEVENT\_NULL If 'pevent' is a NULL pointer

\* OS\_ERR\_PDATA\_NULL If 'p\_q\_data' is a NULL pointer

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_Q\_QUERY\_EN > 0u

INT8U OSQQuery (OS\_EVENT \*pevent,

OS\_Q\_DATA \*p\_q\_data)

{

OS\_Q \*pq;

INT8U i;

OS\_PRIO \*psrc;

OS\_PRIO \*pdest;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

#if OS\_ARG\_CHK\_EN > 0u

if (pevent == (OS\_EVENT \*)0) { /\* Validate 'pevent' \*/

return (OS\_ERR\_PEVENT\_NULL);

}

if (p\_q\_data == (OS\_Q\_DATA \*)0) { /\* Validate 'p\_q\_data' \*/

return (OS\_ERR\_PDATA\_NULL);

}

#endif

if (pevent->OSEventType != OS\_EVENT\_TYPE\_Q) { /\* Validate event block type \*/

return (OS\_ERR\_EVENT\_TYPE);

}

OS\_ENTER\_CRITICAL();

p\_q\_data->OSEventGrp = pevent->OSEventGrp; /\* Copy message queue wait list \*/

psrc = &pevent->OSEventTbl[0];

pdest = &p\_q\_data->OSEventTbl[0];

for (i = 0u; i < OS\_EVENT\_TBL\_SIZE; i++) {

\*pdest++ = \*psrc++;

}

pq = (OS\_Q \*)pevent->OSEventPtr;

if (pq->OSQEntries > 0u) {

p\_q\_data->OSMsg = \*pq->OSQOut; /\* Get next message to return if available \*/

} else {

p\_q\_data->OSMsg = (void \*)0;

}

p\_q\_data->OSNMsgs = pq->OSQEntries;

p\_q\_data->OSQSize = pq->OSQSize;

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_NONE);

}

#endif /\* OS\_Q\_QUERY\_EN \*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* QUEUE MODULE INITIALIZATION

\*

\* Description : This function is called by uC/OS-II to initialize the message queue module. Your

\* application MUST NOT call this function.

\*

\* Arguments : none

\*

\* Returns : none

\*

\* Note(s) : This function is INTERNAL to uC/OS-II and your application should not call it.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void OS\_QInit (void)

{

#if OS\_MAX\_QS == 1u

OSQFreeList = &OSQTbl[0]; /\* Only ONE queue! \*/

OSQFreeList->OSQPtr = (OS\_Q \*)0;

#endif

#if OS\_MAX\_QS >= 2u

INT16U ix;

INT16U ix\_next;

OS\_Q \*pq1;

OS\_Q \*pq2;

OS\_MemClr((INT8U \*)&OSQTbl[0], sizeof(OSQTbl)); /\* Clear the queue table \*/

for (ix = 0u; ix < (OS\_MAX\_QS - 1u); ix++) { /\* Init. list of free QUEUE control blocks \*/

ix\_next = ix + 1u;

pq1 = &OSQTbl[ix];

pq2 = &OSQTbl[ix\_next];

pq1->OSQPtr = pq2;

}

pq1 = &OSQTbl[ix];

pq1->OSQPtr = (OS\_Q \*)0;

OSQFreeList = &OSQTbl[0];

#endif

}

#endif /\* OS\_Q\_EN \*/