/\*

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

\* uC/OS-II

\* The Real-Time Kernel

\* TIMER MANAGEMENT

\*

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

\* All Rights Reserved

\*

\*

\* File : OS\_TMR.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.

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

\*/

#include <ucos\_ii.h>

/\*

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

\* NOTES

\*

\* 1) Your application MUST define the following #define constants:

\*

\* OS\_TASK\_TMR\_PRIO The priority of the Timer management task

\* OS\_TASK\_TMR\_STK\_SIZE The size of the Timer management task's stack

\*

\* 2) You must call OSTmrSignal() to notify the Timer management task that it's time to update the timers.

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

\*/

/\*

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

\* CONSTANTS

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

\*/

#define OS\_TMR\_LINK\_DLY 0u

#define OS\_TMR\_LINK\_PERIODIC 1u

/\*

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

\* LOCAL PROTOTYPES

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

\*/

#if OS\_TMR\_EN > 0u

static OS\_TMR \*OSTmr\_Alloc (void);

static void OSTmr\_Free (OS\_TMR \*ptmr);

static void OSTmr\_InitTask (void);

static void OSTmr\_Link (OS\_TMR \*ptmr, INT8U type);

static void OSTmr\_Unlink (OS\_TMR \*ptmr);

static void OSTmr\_Task (void \*p\_arg);

#endif

/\*

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

\* CREATE A TIMER

\*

\* Description: This function is called by your application code to create a timer.

\*

\* Arguments : dly Initial delay.

\* If the timer is configured for ONE-SHOT mode, this is the timeout used

\* If the timer is configured for PERIODIC mode, this is the first timeout to wait for

\* before the timer starts entering periodic mode

\*

\* period The 'period' being repeated for the timer.

\* If you specified 'OS\_TMR\_OPT\_PERIODIC' as an option, when the timer expires, it will

\* automatically restart with the same period.

\*

\* opt Specifies either:

\* OS\_TMR\_OPT\_ONE\_SHOT The timer counts down only once

\* OS\_TMR\_OPT\_PERIODIC The timer counts down and then reloads itself

\*

\* callback Is a pointer to a callback function that will be called when the timer expires. The

\* callback function must be declared as follows:

\*

\* void MyCallback (OS\_TMR \*ptmr, void \*p\_arg);

\*

\* callback\_arg Is an argument (a pointer) that is passed to the callback function when it is called.

\*

\* pname Is a pointer to an ASCII string that is used to name the timer. Names are useful for

\* debugging.

\*

\* perr Is a pointer to an error code. '\*perr' will contain one of the following:

\* OS\_ERR\_NONE

\* OS\_ERR\_TMR\_INVALID\_DLY you specified an invalid delay

\* OS\_ERR\_TMR\_INVALID\_PERIOD you specified an invalid period

\* OS\_ERR\_TMR\_INVALID\_OPT you specified an invalid option

\* OS\_ERR\_TMR\_ISR if the call was made from an ISR

\* OS\_ERR\_TMR\_NON\_AVAIL if there are no free timers from the timer pool

\*

\* Returns : A pointer to an OS\_TMR data structure.

\* This is the 'handle' that your application will use to reference the timer created.

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

\*/

#if OS\_TMR\_EN > 0u

OS\_TMR \*OSTmrCreate (INT32U dly,

INT32U period,

INT8U opt,

OS\_TMR\_CALLBACK callback,

void \*callback\_arg,

INT8U \*pname,

INT8U \*perr)

{

OS\_TMR \*ptmr;

#ifdef OS\_SAFETY\_CRITICAL

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

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

#ifdef OS\_SAFETY\_CRITICAL\_IEC61508

if (OSSafetyCriticalStartFlag == OS\_TRUE) {

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

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

switch (opt) { /\* Validate arguments \*/

case OS\_TMR\_OPT\_PERIODIC:

if (period == 0u) {

\*perr = OS\_ERR\_TMR\_INVALID\_PERIOD;

return ((OS\_TMR \*)0);

}

break;

case OS\_TMR\_OPT\_ONE\_SHOT:

if (dly == 0u) {

\*perr = OS\_ERR\_TMR\_INVALID\_DLY;

return ((OS\_TMR \*)0);

}

break;

default:

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

return ((OS\_TMR \*)0);

}

#endif

if (OSIntNesting > 0u) { /\* See if trying to call from an ISR \*/

\*perr = OS\_ERR\_TMR\_ISR;

return ((OS\_TMR \*)0);

}

OSSchedLock();

ptmr = OSTmr\_Alloc(); /\* Obtain a timer from the free pool \*/

if (ptmr == (OS\_TMR \*)0) {

OSSchedUnlock();

\*perr = OS\_ERR\_TMR\_NON\_AVAIL;

return ((OS\_TMR \*)0);

}

ptmr->OSTmrState = OS\_TMR\_STATE\_STOPPED; /\* Indicate that timer is not running yet \*/

ptmr->OSTmrDly = dly;

ptmr->OSTmrPeriod = period;

ptmr->OSTmrOpt = opt;

ptmr->OSTmrCallback = callback;

ptmr->OSTmrCallbackArg = callback\_arg;

#if OS\_TMR\_CFG\_NAME\_EN > 0u

ptmr->OSTmrName = pname;

#endif

OSSchedUnlock();

\*perr = OS\_ERR\_NONE;

return (ptmr);

}

#endif

/\*

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

\* DELETE A TIMER

\*

\* Description: This function is called by your application code to delete a timer.

\*

\* Arguments : ptmr Is a pointer to the timer to stop and delete.

\*

\* perr Is a pointer to an error code. '\*perr' will contain one of the following:

\* OS\_ERR\_NONE

\* OS\_ERR\_TMR\_INVALID 'ptmr' is a NULL pointer

\* OS\_ERR\_TMR\_INVALID\_TYPE 'ptmr' is not pointing to an OS\_TMR

\* OS\_ERR\_TMR\_ISR if the function was called from an ISR

\* OS\_ERR\_TMR\_INACTIVE if the timer was not created

\* OS\_ERR\_TMR\_INVALID\_STATE the timer is in an invalid state

\*

\* Returns : OS\_TRUE If the call was successful

\* OS\_FALSE If not

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

\*/

#if OS\_TMR\_EN > 0u

BOOLEAN OSTmrDel (OS\_TMR \*ptmr,

INT8U \*perr)

{

#ifdef OS\_SAFETY\_CRITICAL

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

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

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

if (ptmr == (OS\_TMR \*)0) {

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

return (OS\_FALSE);

}

#endif

if (ptmr->OSTmrType != OS\_TMR\_TYPE) { /\* Validate timer structure \*/

\*perr = OS\_ERR\_TMR\_INVALID\_TYPE;

return (OS\_FALSE);

}

if (OSIntNesting > 0u) { /\* See if trying to call from an ISR \*/

\*perr = OS\_ERR\_TMR\_ISR;

return (OS\_FALSE);

}

OSSchedLock();

switch (ptmr->OSTmrState) {

case OS\_TMR\_STATE\_RUNNING:

OSTmr\_Unlink(ptmr); /\* Remove from current wheel spoke \*/

OSTmr\_Free(ptmr); /\* Return timer to free list of timers \*/

OSSchedUnlock();

\*perr = OS\_ERR\_NONE;

return (OS\_TRUE);

case OS\_TMR\_STATE\_STOPPED: /\* Timer has not started or ... \*/

case OS\_TMR\_STATE\_COMPLETED: /\* ... timer has completed the ONE-SHOT time \*/

OSTmr\_Free(ptmr); /\* Return timer to free list of timers \*/

OSSchedUnlock();

\*perr = OS\_ERR\_NONE;

return (OS\_TRUE);

case OS\_TMR\_STATE\_UNUSED: /\* Already deleted \*/

OSSchedUnlock();

\*perr = OS\_ERR\_TMR\_INACTIVE;

return (OS\_FALSE);

default:

OSSchedUnlock();

\*perr = OS\_ERR\_TMR\_INVALID\_STATE;

return (OS\_FALSE);

}

}

#endif

/\*

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

\* GET THE NAME OF A TIMER

\*

\* Description: This function is called to obtain the name of a timer.

\*

\* Arguments : ptmr Is a pointer to the timer to obtain the name for

\*

\* pdest Is a pointer to pointer to where the name of the timer will be placed.

\*

\* perr Is a pointer to an error code. '\*perr' will contain one of the following:

\* OS\_ERR\_NONE The call was successful

\* OS\_ERR\_TMR\_INVALID\_DEST 'pdest' is a NULL pointer

\* OS\_ERR\_TMR\_INVALID 'ptmr' is a NULL pointer

\* OS\_ERR\_TMR\_INVALID\_TYPE 'ptmr' is not pointing to an OS\_TMR

\* OS\_ERR\_NAME\_GET\_ISR if the call was made from an ISR

\* OS\_ERR\_TMR\_INACTIVE 'ptmr' points to a timer that is not active

\* OS\_ERR\_TMR\_INVALID\_STATE the timer is in an invalid state

\*

\* Returns : The length of the string or 0 if the timer does not exist.

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

\*/

#if OS\_TMR\_EN > 0u && OS\_TMR\_CFG\_NAME\_EN > 0u

INT8U OSTmrNameGet (OS\_TMR \*ptmr,

INT8U \*\*pdest,

INT8U \*perr)

{

INT8U len;

#ifdef OS\_SAFETY\_CRITICAL

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

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

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

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

\*perr = OS\_ERR\_TMR\_INVALID\_DEST;

return (0u);

}

if (ptmr == (OS\_TMR \*)0) {

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

return (0u);

}

#endif

if (ptmr->OSTmrType != OS\_TMR\_TYPE) { /\* Validate timer structure \*/

\*perr = OS\_ERR\_TMR\_INVALID\_TYPE;

return (0u);

}

if (OSIntNesting > 0u) { /\* See if trying to call from an ISR \*/

\*perr = OS\_ERR\_NAME\_GET\_ISR;

return (0u);

}

OSSchedLock();

switch (ptmr->OSTmrState) {

case OS\_TMR\_STATE\_RUNNING:

case OS\_TMR\_STATE\_STOPPED:

case OS\_TMR\_STATE\_COMPLETED:

\*pdest = ptmr->OSTmrName;

len = OS\_StrLen(\*pdest);

OSSchedUnlock();

\*perr = OS\_ERR\_NONE;

return (len);

case OS\_TMR\_STATE\_UNUSED: /\* Timer is not allocated \*/

OSSchedUnlock();

\*perr = OS\_ERR\_TMR\_INACTIVE;

return (0u);

default:

OSSchedUnlock();

\*perr = OS\_ERR\_TMR\_INVALID\_STATE;

return (0u);

}

}

#endif

/\*

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

\* GET HOW MUCH TIME IS LEFT BEFORE A TIMER EXPIRES

\*

\* Description: This function is called to get the number of ticks before a timer times out.

\*

\* Arguments : ptmr Is a pointer to the timer to obtain the remaining time from.

\*

\* perr Is a pointer to an error code. '\*perr' will contain one of the following:

\* OS\_ERR\_NONE

\* OS\_ERR\_TMR\_INVALID 'ptmr' is a NULL pointer

\* OS\_ERR\_TMR\_INVALID\_TYPE 'ptmr' is not pointing to an OS\_TMR

\* OS\_ERR\_TMR\_ISR if the call was made from an ISR

\* OS\_ERR\_TMR\_INACTIVE 'ptmr' points to a timer that is not active

\* OS\_ERR\_TMR\_INVALID\_STATE the timer is in an invalid state

\*

\* Returns : The time remaining for the timer to expire. The time represents 'timer' increments. In other words, if

\* OSTmr\_Task() is signaled every 1/10 of a second then the returned value represents the number of 1/10 of

\* a second remaining before the timer expires.

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

\*/

#if OS\_TMR\_EN > 0u

INT32U OSTmrRemainGet (OS\_TMR \*ptmr,

INT8U \*perr)

{

INT32U remain;

#ifdef OS\_SAFETY\_CRITICAL

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

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

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

if (ptmr == (OS\_TMR \*)0) {

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

return (0u);

}

#endif

if (ptmr->OSTmrType != OS\_TMR\_TYPE) { /\* Validate timer structure \*/

\*perr = OS\_ERR\_TMR\_INVALID\_TYPE;

return (0u);

}

if (OSIntNesting > 0u) { /\* See if trying to call from an ISR \*/

\*perr = OS\_ERR\_TMR\_ISR;

return (0u);

}

OSSchedLock();

switch (ptmr->OSTmrState) {

case OS\_TMR\_STATE\_RUNNING:

remain = ptmr->OSTmrMatch - OSTmrTime; /\* Determine how much time is left to timeout \*/

OSSchedUnlock();

\*perr = OS\_ERR\_NONE;

return (remain);

case OS\_TMR\_STATE\_STOPPED: /\* It's assumed that the timer has not started yet \*/

switch (ptmr->OSTmrOpt) {

case OS\_TMR\_OPT\_PERIODIC:

if (ptmr->OSTmrDly == 0u) {

remain = ptmr->OSTmrPeriod;

} else {

remain = ptmr->OSTmrDly;

}

OSSchedUnlock();

\*perr = OS\_ERR\_NONE;

break;

case OS\_TMR\_OPT\_ONE\_SHOT:

default:

remain = ptmr->OSTmrDly;

OSSchedUnlock();

\*perr = OS\_ERR\_NONE;

break;

}

return (remain);

case OS\_TMR\_STATE\_COMPLETED: /\* Only ONE-SHOT that timed out can be in this state \*/

OSSchedUnlock();

\*perr = OS\_ERR\_NONE;

return (0u);

case OS\_TMR\_STATE\_UNUSED:

OSSchedUnlock();

\*perr = OS\_ERR\_TMR\_INACTIVE;

return (0u);

default:

OSSchedUnlock();

\*perr = OS\_ERR\_TMR\_INVALID\_STATE;

return (0u);

}

}

#endif

/\*

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

\* FIND OUT WHAT STATE A TIMER IS IN

\*

\* Description: This function is called to determine what state the timer is in:

\*

\* OS\_TMR\_STATE\_UNUSED the timer has not been created

\* OS\_TMR\_STATE\_STOPPED the timer has been created but has not been started or has been stopped

\* OS\_TMR\_COMPLETED the timer is in ONE-SHOT mode and has completed it's timeout

\* OS\_TMR\_RUNNING the timer is currently running

\*

\* Arguments : ptmr Is a pointer to the desired timer

\*

\* perr Is a pointer to an error code. '\*perr' will contain one of the following:

\* OS\_ERR\_NONE

\* OS\_ERR\_TMR\_INVALID 'ptmr' is a NULL pointer

\* OS\_ERR\_TMR\_INVALID\_TYPE 'ptmr' is not pointing to an OS\_TMR

\* OS\_ERR\_TMR\_ISR if the call was made from an ISR

\* OS\_ERR\_TMR\_INACTIVE 'ptmr' points to a timer that is not active

\* OS\_ERR\_TMR\_INVALID\_STATE if the timer is not in a valid state

\*

\* Returns : The current state of the timer (see description).

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

\*/

#if OS\_TMR\_EN > 0u

INT8U OSTmrStateGet (OS\_TMR \*ptmr,

INT8U \*perr)

{

INT8U state;

#ifdef OS\_SAFETY\_CRITICAL

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

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

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

if (ptmr == (OS\_TMR \*)0) {

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

return (0u);

}

#endif

if (ptmr->OSTmrType != OS\_TMR\_TYPE) { /\* Validate timer structure \*/

\*perr = OS\_ERR\_TMR\_INVALID\_TYPE;

return (0u);

}

if (OSIntNesting > 0u) { /\* See if trying to call from an ISR \*/

\*perr = OS\_ERR\_TMR\_ISR;

return (0u);

}

OSSchedLock();

state = ptmr->OSTmrState;

switch (state) {

case OS\_TMR\_STATE\_UNUSED:

case OS\_TMR\_STATE\_STOPPED:

case OS\_TMR\_STATE\_COMPLETED:

case OS\_TMR\_STATE\_RUNNING:

\*perr = OS\_ERR\_NONE;

break;

default:

\*perr = OS\_ERR\_TMR\_INVALID\_STATE;

break;

}

OSSchedUnlock();

return (state);

}

#endif

/\*

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

\* START A TIMER

\*

\* Description: This function is called by your application code to start a timer.

\*

\* Arguments : ptmr Is a pointer to an OS\_TMR

\*

\* perr Is a pointer to an error code. '\*perr' will contain one of the following:

\* OS\_ERR\_NONE

\* OS\_ERR\_TMR\_INVALID

\* OS\_ERR\_TMR\_INVALID\_TYPE 'ptmr' is not pointing to an OS\_TMR

\* OS\_ERR\_TMR\_ISR if the call was made from an ISR

\* OS\_ERR\_TMR\_INACTIVE if the timer was not created

\* OS\_ERR\_TMR\_INVALID\_STATE the timer is in an invalid state

\*

\* Returns : OS\_TRUE if the timer was started

\* OS\_FALSE if an error was detected

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

\*/

#if OS\_TMR\_EN > 0u

BOOLEAN OSTmrStart (OS\_TMR \*ptmr,

INT8U \*perr)

{

#ifdef OS\_SAFETY\_CRITICAL

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

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

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

if (ptmr == (OS\_TMR \*)0) {

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

return (OS\_FALSE);

}

#endif

if (ptmr->OSTmrType != OS\_TMR\_TYPE) { /\* Validate timer structure \*/

\*perr = OS\_ERR\_TMR\_INVALID\_TYPE;

return (OS\_FALSE);

}

if (OSIntNesting > 0u) { /\* See if trying to call from an ISR \*/

\*perr = OS\_ERR\_TMR\_ISR;

return (OS\_FALSE);

}

OSSchedLock();

switch (ptmr->OSTmrState) {

case OS\_TMR\_STATE\_RUNNING: /\* Restart the timer \*/

OSTmr\_Unlink(ptmr); /\* ... Stop the timer \*/

OSTmr\_Link(ptmr, OS\_TMR\_LINK\_DLY); /\* ... Link timer to timer wheel \*/

OSSchedUnlock();

\*perr = OS\_ERR\_NONE;

return (OS\_TRUE);

case OS\_TMR\_STATE\_STOPPED: /\* Start the timer \*/

case OS\_TMR\_STATE\_COMPLETED:

OSTmr\_Link(ptmr, OS\_TMR\_LINK\_DLY); /\* ... Link timer to timer wheel \*/

OSSchedUnlock();

\*perr = OS\_ERR\_NONE;

return (OS\_TRUE);

case OS\_TMR\_STATE\_UNUSED: /\* Timer not created \*/

OSSchedUnlock();

\*perr = OS\_ERR\_TMR\_INACTIVE;

return (OS\_FALSE);

default:

OSSchedUnlock();

\*perr = OS\_ERR\_TMR\_INVALID\_STATE;

return (OS\_FALSE);

}

}

#endif

/\*

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

\* STOP A TIMER

\*

\* Description: This function is called by your application code to stop a timer.

\*

\* Arguments : ptmr Is a pointer to the timer to stop.

\*

\* opt Allows you to specify an option to this functions which can be:

\*

\* OS\_TMR\_OPT\_NONE Do nothing special but stop the timer

\* OS\_TMR\_OPT\_CALLBACK Execute the callback function, pass it the callback argument

\* specified when the timer was created.

\* OS\_TMR\_OPT\_CALLBACK\_ARG Execute the callback function, pass it the callback argument

\* specified in THIS function call

\*

\* callback\_arg Is a pointer to a 'new' callback argument that can be passed to the callback function

\* instead of the timer's callback argument. In other words, use 'callback\_arg' passed in

\* THIS function INSTEAD of ptmr->OSTmrCallbackArg

\*

\* perr Is a pointer to an error code. '\*perr' will contain one of the following:

\* OS\_ERR\_NONE

\* OS\_ERR\_TMR\_INVALID 'ptmr' is a NULL pointer

\* OS\_ERR\_TMR\_INVALID\_TYPE 'ptmr' is not pointing to an OS\_TMR

\* OS\_ERR\_TMR\_ISR if the function was called from an ISR

\* OS\_ERR\_TMR\_INACTIVE if the timer was not created

\* OS\_ERR\_TMR\_INVALID\_OPT if you specified an invalid option for 'opt'

\* OS\_ERR\_TMR\_STOPPED if the timer was already stopped

\* OS\_ERR\_TMR\_INVALID\_STATE the timer is in an invalid state

\* OS\_ERR\_TMR\_NO\_CALLBACK if the timer does not have a callback function defined

\*

\* Returns : OS\_TRUE If we stopped the timer (if the timer is already stopped, we also return OS\_TRUE)

\* OS\_FALSE If not

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

\*/

#if OS\_TMR\_EN > 0u

BOOLEAN OSTmrStop (OS\_TMR \*ptmr,

INT8U opt,

void \*callback\_arg,

INT8U \*perr)

{

OS\_TMR\_CALLBACK pfnct;

#ifdef OS\_SAFETY\_CRITICAL

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

OS\_SAFETY\_CRITICAL\_EXCEPTION();

}

#endif

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

if (ptmr == (OS\_TMR \*)0) {

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

return (OS\_FALSE);

}

#endif

if (ptmr->OSTmrType != OS\_TMR\_TYPE) { /\* Validate timer structure \*/

\*perr = OS\_ERR\_TMR\_INVALID\_TYPE;

return (OS\_FALSE);

}

if (OSIntNesting > 0u) { /\* See if trying to call from an ISR \*/

\*perr = OS\_ERR\_TMR\_ISR;

return (OS\_FALSE);

}

OSSchedLock();

switch (ptmr->OSTmrState) {

case OS\_TMR\_STATE\_RUNNING:

OSTmr\_Unlink(ptmr); /\* Remove from current wheel spoke \*/

\*perr = OS\_ERR\_NONE;

switch (opt) {

case OS\_TMR\_OPT\_CALLBACK:

pfnct = ptmr->OSTmrCallback; /\* Execute callback function if available ... \*/

if (pfnct != (OS\_TMR\_CALLBACK)0) {

(\*pfnct)((void \*)ptmr, ptmr->OSTmrCallbackArg); /\* Use callback arg when timer was created \*/

} else {

\*perr = OS\_ERR\_TMR\_NO\_CALLBACK;

}

break;

case OS\_TMR\_OPT\_CALLBACK\_ARG:

pfnct = ptmr->OSTmrCallback; /\* Execute callback function if available ... \*/

if (pfnct != (OS\_TMR\_CALLBACK)0) {

(\*pfnct)((void \*)ptmr, callback\_arg); /\* ... using the 'callback\_arg' provided in call \*/

} else {

\*perr = OS\_ERR\_TMR\_NO\_CALLBACK;

}

break;

case OS\_TMR\_OPT\_NONE:

break;

default:

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

break;

}

OSSchedUnlock();

return (OS\_TRUE);

case OS\_TMR\_STATE\_COMPLETED: /\* Timer has already completed the ONE-SHOT or ... \*/

case OS\_TMR\_STATE\_STOPPED: /\* ... timer has not started yet. \*/

OSSchedUnlock();

\*perr = OS\_ERR\_TMR\_STOPPED;

return (OS\_TRUE);

case OS\_TMR\_STATE\_UNUSED: /\* Timer was not created \*/

OSSchedUnlock();

\*perr = OS\_ERR\_TMR\_INACTIVE;

return (OS\_FALSE);

default:

OSSchedUnlock();

\*perr = OS\_ERR\_TMR\_INVALID\_STATE;

return (OS\_FALSE);

}

}

#endif

/\*

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

\* SIGNAL THAT IT'S TIME TO UPDATE THE TIMERS

\*

\* Description: This function is typically called by the ISR that occurs at the timer tick rate and is used to signal to

\* OSTmr\_Task() that it's time to update the timers.

\*

\* Arguments : none

\*

\* Returns : OS\_ERR\_NONE The call was successful and the timer task was signaled.

\* OS\_ERR\_SEM\_OVF If OSTmrSignal() was called more often than OSTmr\_Task() can handle the timers.

\* This would indicate that your system is heavily loaded.

\* OS\_ERR\_EVENT\_TYPE Unlikely you would get this error because the semaphore used for signaling is created

\* by uC/OS-II.

\* OS\_ERR\_PEVENT\_NULL Again, unlikely you would ever get this error because the semaphore used for signaling

\* is created by uC/OS-II.

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

\*/

#if OS\_TMR\_EN > 0u

INT8U OSTmrSignal (void)

{

INT8U err;

err = OSSemPost(OSTmrSemSignal);

return (err);

}

#endif

/\*

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

\* ALLOCATE AND FREE A TIMER

\*

\* Description: This function is called to allocate a timer.

\*

\* Arguments : none

\*

\* Returns : a pointer to a timer if one is available

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

\*/

#if OS\_TMR\_EN > 0u

static OS\_TMR \*OSTmr\_Alloc (void)

{

OS\_TMR \*ptmr;

if (OSTmrFreeList == (OS\_TMR \*)0) {

return ((OS\_TMR \*)0);

}

ptmr = (OS\_TMR \*)OSTmrFreeList;

OSTmrFreeList = (OS\_TMR \*)ptmr->OSTmrNext;

ptmr->OSTmrNext = (OS\_TCB \*)0;

ptmr->OSTmrPrev = (OS\_TCB \*)0;

OSTmrUsed++;

OSTmrFree--;

return (ptmr);

}

#endif

/\*

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

\* RETURN A TIMER TO THE FREE LIST

\*

\* Description: This function is called to return a timer object to the free list of timers.

\*

\* Arguments : ptmr is a pointer to the timer to free

\*

\* Returns : none

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

\*/

#if OS\_TMR\_EN > 0u

static void OSTmr\_Free (OS\_TMR \*ptmr)

{

ptmr->OSTmrState = OS\_TMR\_STATE\_UNUSED; /\* Clear timer object fields \*/

ptmr->OSTmrOpt = OS\_TMR\_OPT\_NONE;

ptmr->OSTmrPeriod = 0u;

ptmr->OSTmrMatch = 0u;

ptmr->OSTmrCallback = (OS\_TMR\_CALLBACK)0;

ptmr->OSTmrCallbackArg = (void \*)0;

#if OS\_TMR\_CFG\_NAME\_EN > 0u

ptmr->OSTmrName = (INT8U \*)(void \*)"?";

#endif

ptmr->OSTmrPrev = (OS\_TCB \*)0; /\* Chain timer to free list \*/

ptmr->OSTmrNext = OSTmrFreeList;

OSTmrFreeList = ptmr;

OSTmrUsed--; /\* Update timer object statistics \*/

OSTmrFree++;

}

#endif

/\*

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

\* INITIALIZATION

\* INITIALIZE THE FREE LIST OF TIMERS

\*

\* Description: This function is called by OSInit() to initialize the free list of OS\_TMRs.

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

#if OS\_TMR\_EN > 0u

void OSTmr\_Init (void)

{

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

INT8U err;

#endif

INT16U ix;

INT16U ix\_next;

OS\_TMR \*ptmr1;

OS\_TMR \*ptmr2;

OS\_MemClr((INT8U \*)&OSTmrTbl[0], sizeof(OSTmrTbl)); /\* Clear all the TMRs \*/

OS\_MemClr((INT8U \*)&OSTmrWheelTbl[0], sizeof(OSTmrWheelTbl)); /\* Clear the timer wheel \*/

for (ix = 0u; ix < (OS\_TMR\_CFG\_MAX - 1u); ix++) { /\* Init. list of free TMRs \*/

ix\_next = ix + 1u;

ptmr1 = &OSTmrTbl[ix];

ptmr2 = &OSTmrTbl[ix\_next];

ptmr1->OSTmrType = OS\_TMR\_TYPE;

ptmr1->OSTmrState = OS\_TMR\_STATE\_UNUSED; /\* Indicate that timer is inactive \*/

ptmr1->OSTmrNext = (void \*)ptmr2; /\* Link to next timer \*/

#if OS\_TMR\_CFG\_NAME\_EN > 0u

ptmr1->OSTmrName = (INT8U \*)(void \*)"?";

#endif

}

ptmr1 = &OSTmrTbl[ix];

ptmr1->OSTmrType = OS\_TMR\_TYPE;

ptmr1->OSTmrState = OS\_TMR\_STATE\_UNUSED; /\* Indicate that timer is inactive \*/

ptmr1->OSTmrNext = (void \*)0; /\* Last OS\_TMR \*/

#if OS\_TMR\_CFG\_NAME\_EN > 0u

ptmr1->OSTmrName = (INT8U \*)(void \*)"?";

#endif

OSTmrTime = 0u;

OSTmrUsed = 0u;

OSTmrFree = OS\_TMR\_CFG\_MAX;

OSTmrFreeList = &OSTmrTbl[0];

OSTmrSem = OSSemCreate(1u);

OSTmrSemSignal = OSSemCreate(0u);

#if OS\_EVENT\_NAME\_EN > 0u /\* Assign names to semaphores \*/

OSEventNameSet(OSTmrSem, (INT8U \*)(void \*)"uC/OS-II TmrLock", &err);

OSEventNameSet(OSTmrSemSignal, (INT8U \*)(void \*)"uC/OS-II TmrSignal", &err);

#endif

OSTmr\_InitTask();

}

#endif

/\*

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

\* INITIALIZE THE TIMER MANAGEMENT TASK

\*

\* Description: This function is called by OSTmrInit() to create the timer management task.

\* \* Arguments : none

\*

\* Returns : none

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

\*/

#if OS\_TMR\_EN > 0u

static void OSTmr\_InitTask (void)

{

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

INT8U err;

#endif

#if OS\_TASK\_CREATE\_EXT\_EN > 0u

#if OS\_STK\_GROWTH == 1u

(void)OSTaskCreateExt(OSTmr\_Task,

(void \*)0, /\* No arguments passed to OSTmrTask() \*/

&OSTmrTaskStk[OS\_TASK\_TMR\_STK\_SIZE - 1u], /\* Set Top-Of-Stack \*/

OS\_TASK\_TMR\_PRIO,

OS\_TASK\_TMR\_ID,

&OSTmrTaskStk[0], /\* Set Bottom-Of-Stack \*/

OS\_TASK\_TMR\_STK\_SIZE,

(void \*)0, /\* No TCB extension \*/

OS\_TASK\_OPT\_STK\_CHK | OS\_TASK\_OPT\_STK\_CLR); /\* Enable stack checking + clear stack \*/

#else

(void)OSTaskCreateExt(OSTmr\_Task,

(void \*)0, /\* No arguments passed to OSTmrTask() \*/

&OSTmrTaskStk[0], /\* Set Top-Of-Stack \*/

OS\_TASK\_TMR\_PRIO,

OS\_TASK\_TMR\_ID,

&OSTmrTaskStk[OS\_TASK\_TMR\_STK\_SIZE - 1u], /\* Set Bottom-Of-Stack \*/

OS\_TASK\_TMR\_STK\_SIZE,

(void \*)0, /\* No TCB extension \*/

OS\_TASK\_OPT\_STK\_CHK | OS\_TASK\_OPT\_STK\_CLR); /\* Enable stack checking + clear stack \*/

#endif

#else

#if OS\_STK\_GROWTH == 1u

(void)OSTaskCreate(OSTmr\_Task,

(void \*)0,

&OSTmrTaskStk[OS\_TASK\_TMR\_STK\_SIZE - 1u],

OS\_TASK\_TMR\_PRIO);

#else

(void)OSTaskCreate(OSTmr\_Task,

(void \*)0,

&OSTmrTaskStk[0],

OS\_TASK\_TMR\_PRIO);

#endif

#endif

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

OSTaskNameSet(OS\_TASK\_TMR\_PRIO, (INT8U \*)(void \*)"uC/OS-II Tmr", &err);

#endif

}

#endif

/\*

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

\* INSERT A TIMER INTO THE TIMER WHEEL

\*

\* Description: This function is called to insert the timer into the timer wheel. The timer is always inserted at the

\* beginning of the list.

\*

\* Arguments : ptmr Is a pointer to the timer to insert.

\*

\* type Is either:

\* OS\_TMR\_LINK\_PERIODIC Means to re-insert the timer after a period expired

\* OS\_TMR\_LINK\_DLY Means to insert the timer the first time

\*

\* Returns : none

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

\*/

#if OS\_TMR\_EN > 0u

static void OSTmr\_Link (OS\_TMR \*ptmr,

INT8U type)

{

OS\_TMR \*ptmr1;

OS\_TMR\_WHEEL \*pspoke;

INT16U spoke;

ptmr->OSTmrState = OS\_TMR\_STATE\_RUNNING;

if (type == OS\_TMR\_LINK\_PERIODIC) { /\* Determine when timer will expire \*/

ptmr->OSTmrMatch = ptmr->OSTmrPeriod + OSTmrTime;

} else {

if (ptmr->OSTmrDly == 0u) {

ptmr->OSTmrMatch = ptmr->OSTmrPeriod + OSTmrTime;

} else {

ptmr->OSTmrMatch = ptmr->OSTmrDly + OSTmrTime;

}

}

spoke = (INT16U)(ptmr->OSTmrMatch % OS\_TMR\_CFG\_WHEEL\_SIZE);

pspoke = &OSTmrWheelTbl[spoke];

if (pspoke->OSTmrFirst == (OS\_TMR \*)0) { /\* Link into timer wheel \*/

pspoke->OSTmrFirst = ptmr;

ptmr->OSTmrNext = (OS\_TMR \*)0;

pspoke->OSTmrEntries = 1u;

} else {

ptmr1 = pspoke->OSTmrFirst; /\* Point to first timer in the spoke \*/

pspoke->OSTmrFirst = ptmr;

ptmr->OSTmrNext = (void \*)ptmr1;

ptmr1->OSTmrPrev = (void \*)ptmr;

pspoke->OSTmrEntries++;

}

ptmr->OSTmrPrev = (void \*)0; /\* Timer always inserted as first node in list \*/

}

#endif

/\*

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

\* REMOVE A TIMER FROM THE TIMER WHEEL

\*

\* Description: This function is called to remove the timer from the timer wheel.

\*

\* Arguments : ptmr Is a pointer to the timer to remove.

\*

\* Returns : none

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

\*/

#if OS\_TMR\_EN > 0u

static void OSTmr\_Unlink (OS\_TMR \*ptmr)

{

OS\_TMR \*ptmr1;

OS\_TMR \*ptmr2;

OS\_TMR\_WHEEL \*pspoke;

INT16U spoke;

spoke = (INT16U)(ptmr->OSTmrMatch % OS\_TMR\_CFG\_WHEEL\_SIZE);

pspoke = &OSTmrWheelTbl[spoke];

if (pspoke->OSTmrFirst == ptmr) { /\* See if timer to remove is at the beginning of list \*/

ptmr1 = (OS\_TMR \*)ptmr->OSTmrNext;

pspoke->OSTmrFirst = (OS\_TMR \*)ptmr1;

if (ptmr1 != (OS\_TMR \*)0) {

ptmr1->OSTmrPrev = (void \*)0;

}

} else {

ptmr1 = (OS\_TMR \*)ptmr->OSTmrPrev; /\* Remove timer from somewhere in the list \*/

ptmr2 = (OS\_TMR \*)ptmr->OSTmrNext;

ptmr1->OSTmrNext = ptmr2;

if (ptmr2 != (OS\_TMR \*)0) {

ptmr2->OSTmrPrev = (void \*)ptmr1;

}

}

ptmr->OSTmrState = OS\_TMR\_STATE\_STOPPED;

ptmr->OSTmrNext = (void \*)0;

ptmr->OSTmrPrev = (void \*)0;

pspoke->OSTmrEntries--;

}

#endif

/\*

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

\* TIMER MANAGEMENT TASK

\*

\* Description: This task is created by OSTmrInit().

\*

\* Arguments : none

\*

\* Returns : none

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

\*/

#if OS\_TMR\_EN > 0u

static void OSTmr\_Task (void \*p\_arg)

{

INT8U err;

OS\_TMR \*ptmr;

OS\_TMR \*ptmr\_next;

OS\_TMR\_CALLBACK pfnct;

OS\_TMR\_WHEEL \*pspoke;

INT16U spoke;

p\_arg = p\_arg; /\* Prevent compiler warning for not using 'p\_arg' \*/

for (;;) {

OSSemPend(OSTmrSemSignal, 0u, &err); /\* Wait for signal indicating time to update timers \*/

OSSchedLock();

OSTmrTime++; /\* Increment the current time \*/

spoke = (INT16U)(OSTmrTime % OS\_TMR\_CFG\_WHEEL\_SIZE); /\* Position on current timer wheel entry \*/

pspoke = &OSTmrWheelTbl[spoke];

ptmr = pspoke->OSTmrFirst;

while (ptmr != (OS\_TMR \*)0) {

ptmr\_next = (OS\_TMR \*)ptmr->OSTmrNext; /\* Point to next timer to update because current ... \*/

/\* ... timer could get unlinked from the wheel. \*/

if (OSTmrTime == ptmr->OSTmrMatch) { /\* Process each timer that expires \*/

OSTmr\_Unlink(ptmr); /\* Remove from current wheel spoke \*/

if (ptmr->OSTmrOpt == OS\_TMR\_OPT\_PERIODIC) {

OSTmr\_Link(ptmr, OS\_TMR\_LINK\_PERIODIC); /\* Recalculate new position of timer in wheel \*/

} else {

ptmr->OSTmrState = OS\_TMR\_STATE\_COMPLETED; /\* Indicate that the timer has completed \*/

}

pfnct = ptmr->OSTmrCallback; /\* Execute callback function if available \*/

if (pfnct != (OS\_TMR\_CALLBACK)0) {

(\*pfnct)((void \*)ptmr, ptmr->OSTmrCallbackArg);

}

}

ptmr = ptmr\_next;

}

OSSchedUnlock();

}

}

#endif