Micriμm

New Features and Services since

μC/OS-II V2.00

(Current Version: V2.92.11)

www.Micrium.com

Introduction

This document describes all the features and services added to **µC/OS-II** since the introduction of the hard cover book *MicroC/OS-II, The Real-Time Kernel*, ISBN 0-87930-543-6. The software provided with the book was version 2.00 or V2.04. The version number of the change is shown when appropriate.

Added 'TLS' (V2.92.08)

µC/OS-II now supports 'Thread Local Storage' (TLS) for Analog Devices' CrossCore Embedded Studio (CCES) and IAR's Embedded Workbench.

µC/OS-II now provides built-in support for run-time library thread safety through the use of Task Local Storage (TLS) for storage of task-specific run-time library static data and mutual exclusion semaphores to protect accesses to shared resources.

The run-time environment consists of the run-time library, which contains the functions defined by the C and the C++ standards, and includes files that define the library interface (the system header files). Compilers provide complete libraries that are compliant with Standard C and C++. These libraries also supports floating-point numbers in IEEE 754 format and can be configured to include different levels of support for locale, file descriptors, multi-byte characters, etc. Most parts of the libraries are reentrant, but some functionality and parts are not reentrant because they require the use of static data. Different compilers provide different methods to add reentrancy to their libraries through an API defined by the tool chain supplier.

The current version supports TLS for Analog Devices' CrossCore Embedded Studio and IAR's Embedded Workbench.

In a multi-threaded environment the C/C++ library has to handle all library objects with a global state differently. Either an object is a true global object, then any updates of its state has to be guarded by some locking mechanism to make sure that only one task can update it at any one time, or an object is local to each task, then the static variables containing the objects state must reside in a variable area local for the task. This area is commonly named thread local storage or, TLS.

The run-time library may also need to use multiple types of locks. For example, a lock could be necessary to ensure exclusive access to the file stream, another one to the heap, etc. It is thus common to protect the following functions through one or more mutual exclusion semaphores (mutex):

- The heap through the usage of +, , +, , +, , and +, .
- The file system through the usage of +, , +, , +, , +, , and +, .
- The signal system through the usage of +).
- The tempfile system through the usage of +, .
- Initialization of static function objects.
- Thread-local storage is typically needed for the following library objects:

- Error functions through and
- Locale functions through the usage of +, and +,
- Time functions through the usage of +, , +, , +, , and +,
- Multibyte functions through the usage of +, , +, , +, , mbtowc(), +, , +, , and +,
- Random functions through the usage of +, and +,
- Other functions through the usage of +, and +,

Added ‘Task Registers’ (V2.92.07)

µC/OS-II now allows the user to store task-specific values in ‘task registers’. Task registers are different than CPU registers and are used to save such information as “errno” which is common in software components. Task registers can also store task-related data to be associated with the task at run time such as I/O register settings, configuration values, etc. A task may have as many as D N H E L H registers, and all registers have a data type of L 65 . A task register is changed by calling +, and read by calling +, . The desired task register is specified as an argument to these functions and can take a value between 3 and D N H E L H04.

```
+,#
LG +,#
+,#
```

+, now returns a version number scaled by 43/333. So, if +, returns 5<53 then it means V2.92.07.

Version number skipped (V2.91)

There is no version 2.91.

Added ‘OSSafetyCriticalStart()’ (V2.90)

µC/OS-II now allows your application to ‘tell’ **µC/OS-II** that your application is no longer allowed to delete kernel objects such as tasks, semaphores, queues, etc.

Delete task on incorrect return (V2.89)

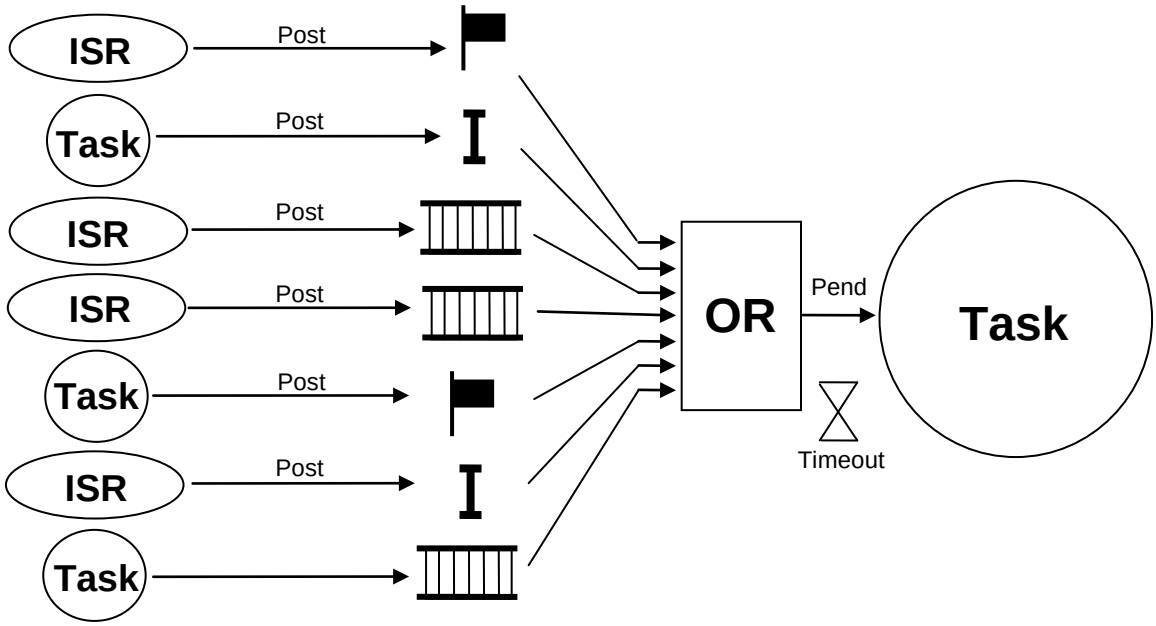
µC/OS-II now contains a new function called +, . All µC/OS-II tasks are not allowed to return. If a task returns by mistake, +, catches those and deletes the task.

+, calls K +, which in turn calls D K +, .

Pend on Multiple Events (V2.86)

µC/OS-II now contains a new function called H +, which allows a task to pend on multiple events (semaphores, mailboxes and queues) in any combination (see example diagram below). This new function is found in F H1F and is enabled by setting H H L H to 4 in FI 1K.

With H +, it's possible to pend on any number of semaphores, mailboxes and message queues at the same time (we don't support Mutex and Event Flags at this time). If a task pends on a combination of the above 'events' then, as soon an event is posted (and the pending task is the highest priority task pending on the event), the waiting task will wake up and be 'handed' the event. If events are present as the task pends then ALL the available events will be provided to the task.



Timer Manager (V2.81)

µC/OS-II now provides support for periodic as well as one-shot timers. This functionality is found in 1F and is enabled by setting H to 4 in FI 1K. Your application can have any number of timers (up to 65500). When a timer times out, an optional callback function can be called allowing you to perform any action (signal a task, turn on/off a light, etc.). Each timer has its own callback function.

IMPORTANT

The APIs for the Timer Manager were changed in V2.83 from what they were in V2.81 and V2.82. This was necessary to correct some issues with the Timer Manager. Please consult the Reference Manual for the new APIs.

When timer management is enabled, **µC/OS-II** creates a timer task (+,) which is responsible for updating all the timers. The priority of this task is determined by D N L which should be defined in your application's D FI 1K.

The timer manager provides a number of services to your applications. Specifically, you can call one of the following functions (see the **µC/OS-II** reference manual for a description of these functions) from your tasks:

F +,	Create a timer
G +,	Delete a timer
+,	Determine how much time before a timer expires
+,	Get the name of a timer
+,	Get the state of a timer (UNUSED, STOPPED, RUNNING, COMPLETED)
+,	Start a timer
+,	Stop a timer

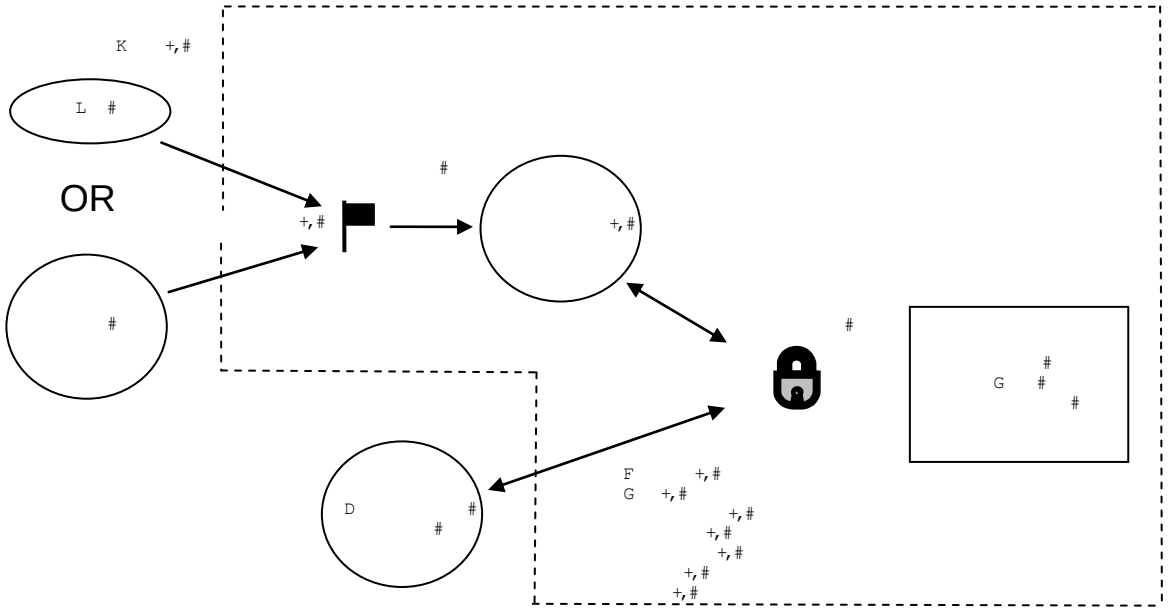
You should note that you **CANNOT** call these functions from ISRs.

The drawing below shows the task model of the Timer Manager. You should note that semaphore management needs to be enabled (you need to set H H to 4 in FI 1K) for the timer manager to work. The timer manager requires two (2) semaphores.

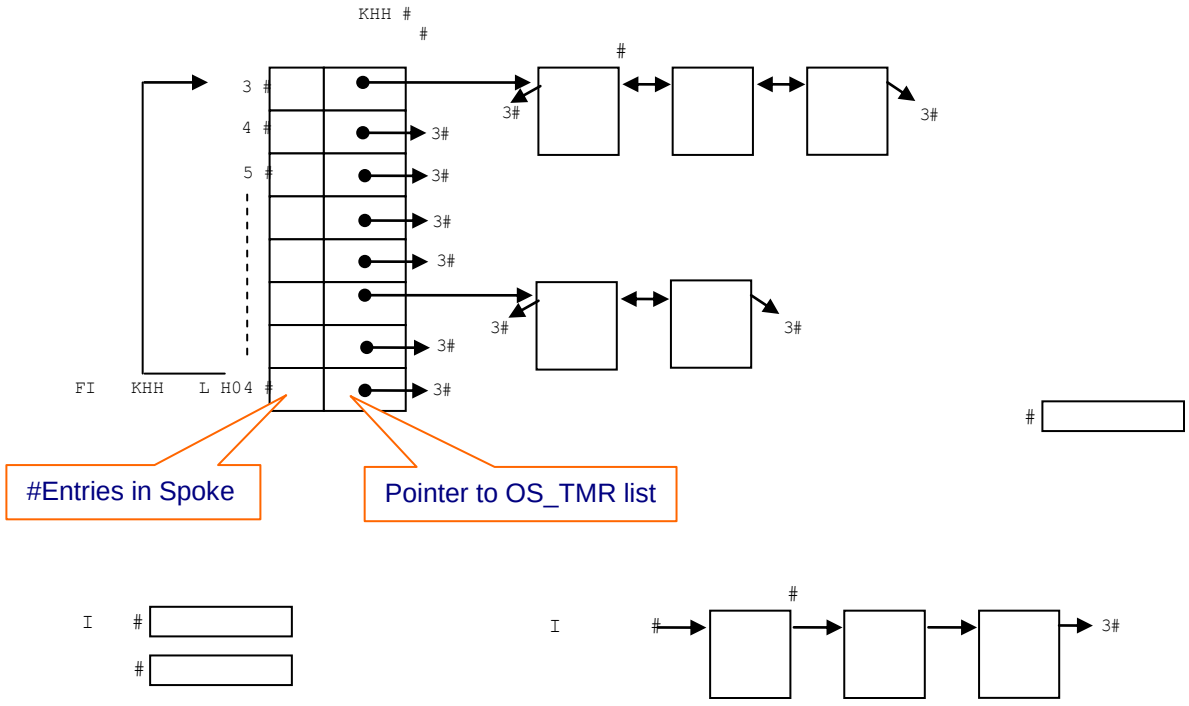
(1) An ISR or an application task needs to 'signal' a counting semaphore by calling +, at the rate specified in FI LFN D H (see FI 1K). The counting semaphore is called that is initialized to 3 by µC/OS-II when L +, is called. You should note that you should ONLY call +, and not worry about the semaphore; it's encapsulated by +, .

(2) The timer management task (+,) pends forever on the counting semaphore waiting for it to be signaled. When the semaphore is signaled, +, acquires another semaphore (a binary semaphore in this case,) to gain exclusive access to timer data structures. When +, is the owner of the semaphore it updates all the timers created by your application.

(3) Your application accesses timer data structure via interface functions. These functions allow you to create, delete, start and stop timers as well as examine the amount of time remaining before a timer times out.



The drawing below shows the data structures used in the timer manager.



(4) Each timer is characterized by a data structure of type (see 1). Each timer contains the 'period' of the timer (if the timer is to operate in periodic mode), the name of the timer, a timer 'match' value (described later) and other fields used to link the timer. Free timers are placed in a singly linked list of 'unused' timers pointed to by I .

(5) The number of free timers is held in I and the number of used (or allocated) timers is held in . Of course, the total number of timers is the sum of these two fields and, unless you don't properly use the timer management services, that sum should always equal FI D .

(6) Every time +, is called, the unsigned 32-bit variable is incremented by one and used to see if timers have expired.

(7) The timer manager keeps track of which timer it needs to update using a 'timer wheel'. The wheel is basically an array of structures of type KHH (see 1) that wraps around. This structure contains two fields: a pointer to a doubly-linked list of structures and, the number of entries in that list.

(8) The 'wheel' contains FI KHH L H entries or spokes.
#
#
structures are inserted in the wheel when you call +, . The position (i.e. spoke) in for a specific timer is given by:

```
# #          #.#        #
# #      #(#        FI   KHH   L H #
```

The 'match' corresponds to the value that needs to reach before the timer expires. For example, let's say that is 3 (just initialized) and we want to create a timer that will expire every second (assuming FI LFN H HF is set to 43). Also, let's assume that FI KHH L H is ; (as shown in the diagram above).

```
# #          #.#        #
# #3#.#43 #
# #43 #
#
# #      #(#        FI   KHH   L H #
# #43#(#; #
# #5 #
```

This means that +, will obtain a free data structure from the free list of timers and the place this data structure in at position #2 in the table. +, will then store the 'match' value in the data structure.

Every time is incremented by +, , +, goes through ALL the structures placed at spoke (#(# FI KHH L H) to see if 'matches' the value store in the structure. If a match occurs, the timer is removed from the list. If the timer was started by +, with a 'periodic' option then, the structure is placed in the by calculating its new position, again using #.# . In our example, the new 'spoke' would be:

```
          # #          #.#        #
          # #43#.#43 #
          # #53 #
     #
          # #      #(#        FI    KHH    L H #
          # #53#(#; #
          # #   #
```

The use of a timer wheel basically reduces the execution time of the timer task so that it only handles a few of the timers. Of course, the worst case is such that all timers are placed in the same spoke of the timer wheel. However, statistically, this will occur rarely. It's generally recommended to keep the size of the wheel a fraction of the total number of times. In other words, you should set:

```
              FI    KHH    L H#   #I        #   # +        FI    D ,#
```

A fraction of 2 to 8 should work well.

RAM usage (in bytes) for the timer manager is shown below:

```
5#-#       +L  49 ,##.#
4#-#       +L  65 ,##.#
6#-#       +  L  H ,#.#
    D N        N  L H#-#       +     N,#.#
       FI   KHH   L H#-# +       +L  49 ,#.#       +  L  H ,,#.#
       FI   D #-# + #-#       +  L  H ,#.##
###################5#-#       +L  65 ,##.##
###################6#-#       +L  ; ,###.##
##################       FI   D H  L H#-#       +L  ; ,,#
```

Because `L ; s` and `E HD s` are typically 1 byte, `L 49 s` are 2 bytes and `L 65 s` are 4 bytes, we can simplify the above equation as follows:

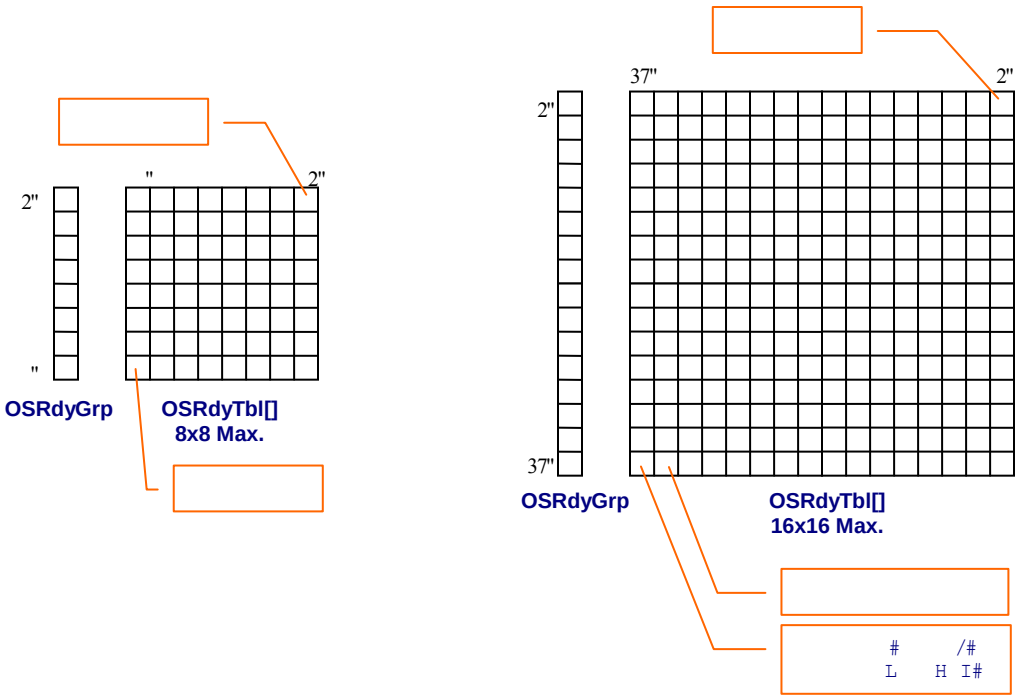
```
5#-#5##.#
4#-# ##.#
6#-#       +  L  H ,#.#
    D N        N  L H#-#       +     N,#.#
       FI   KHH   L H#-# +5#.#       +  L  H ,,#.#
       FI   D #-# + #-#       +  L  H ,#.##
###################5#-# ##.##
###################6######.##
##################       FI   D H  L H,#
```

Or,

```
;##############################################.#
6#-#       +  L  H ,#############################.#
                    #-#       +     N,#########.#
                     #-# +5#.#       +  L  H ,,#.#
              #-# + #-#       +  L  H ,#.#44#.#                    ,#
#
```

Support for 255 tasks (V2.80)

µC/OS-II can now support up to **255** tasks. To support up to 255 tasks, we simply increased the ready list and event wait lists to a matrix of 16x16 instead of 8x8. In fact, the actual size of the matrix (whether 8x8 or 16x16) depends on the value of H L in FI 1K. If H L is less than or equal to 96, we use an 8x8 matrix and thus **µC/OS-II** behaves exactly the same as it used to. If you specify a value for H L to be greater than 96, we use the 16x16 matrix as show below.



OS_LOWEST_PRIO <= 63 **OS_LOWEST_PRIO > 63**

You should note that the actual size of the matrix depends on H L . For example, if H L is 43 then the matrix is actually 2x8 (two bytes of 8 bits). Similarly, if H L is set to , the matrix will be 6x8. When H L is above 96, we use 16-bit wide entries. For example, if you specify H L to be 433 then the matrix will be 7x16 (7 entries of 16 bits each). You **CANNOT** set H L to 588 because this value is reserved for L H I.

New Files

"

D FI 1K# # # # " " 40 2 #

" " " " " " " " "D FI 1K" " " " " " " 0" "
" " " " " " " " " ." " " " " " " " " " "
0"

"

FI 1K# # # # " " 40 2 #

" " " " " " " " " " " " " " " " " " 0" FI 1K" " "
" " 40 2" " " " " " " " " " " " " " " " " " " "
µC/OS-II0" " " " " " " FI 1K" " FI 1K" " " " " 0"

"

1F# # # # " " 40 3." " " 40 5 #

" " " " " " " "µC/OS-II0" " " " " " " " " " 0" " "
" " " " / 0" " " " " " " " " " " " " " 0" " "
" " " " " " " " " 0

"

New Port Files

GE 1F# # # # " " 40840" " " " GHE 1F" " 40 2 #
GE 1F# # # # " " 40 2 #

" " " " " " " " " " F F 1F." F 1K" " F D 1D " "
" " " " 0" GE 1F" " " " " " " " " " " " " " 0"
" " " " " " " " " " " " " " " " "µC/OS-II" " "
0" "
" 0" " " " " " " 0" GE 1F" " " " " GHE 1F"
" 40840

GE 1F" " " " " " " " " " " " " " " " " 0" GE 1F"
" " " " " 40 2" " " " " " " " " " " " " " " " "
"µC/OS-II0

"

Changes

F LL1K# # # # # # " "" 40 2." 40 2" " 40 6 #

" " " "& " " " "D FI 1K." F 1K" " FI 1K0" "
" " " "µC/OS-II" " " " " " " " 0

" " " " " " 0" ." " " " " " H 0" "
" "
" 0

"

" "" 40

" " " " " " " " " " " "54/ " 0" " "
" G K +," " " G +," " " " " 0" " ."
" " " " " " " " " " FE " 4" " " " " " " " " " "
" " " 0" " " " " " " 2 " " 1 / " " "54/ " 0

"

" "" 40

" " " " " " " " " " " " " 0 ." "
" " " " " ." " ." " " " " " " " 0 " " 40 ." " " "
" " " " 0 " " " " " " " " " " " " " "
" 0
" " " " " " " " " " " " " " " " " " " "
" " " " " " " " " " " " " 0

"

" "" 40 4

" " " " F +," " F H +,." F +," "
" " " " " " " " " " " " " " " " " 0 " " " " "
" " " " ." " " " " + N,0

"

New #define Constants and Macros

D K N H # # + FI 1K/# 51;8,#

" " " " µC/OS-II " " " " " " " 0"
." " " "3" "

"

The µC/OS-II hook …	Calls the Application-define hook …
F K +,#	D F K +,#
G K +,#	D G K +,#
L K +,#	D L K +,#
K +,#	D K +,#
K +,#	D K +,#
FEL K +,#	D FEL K +,#
K +,#	D K +,#

"

D FKN H # # + FI 1K/# 513 ,#

" " " " " " " " " " " " " " " " " "
µC/OS-II" 0" " " " " " " " " " " " "4 " " " " "
" " " " " " " "µC/OS-II" ." " " " " " 0

"

#

F L LFD H K G#&6# + F 1K/# 513 ,#

" " " " " " " " " " " " " " " " " " 0"
" " 4026." F L LFD H K G" " " " " "4" "50" " 4026." " " " " "
0 0" " " " " " " " " " " " " " " " " " " "
" " " 0" " " " " " " " " " " " " " " "

"

& # H H F L LFD +,## + # # F +,,#
& # H L F L LFD +,### + F + ,,#

Note that the functions F +, and F +, would be written in assembly language and would typically be found in F D 1D (or equivalent).

GHE H ## # + FI 1K/# 5193,#

" " " " " " " " " " " " " " " " 0" "
" " " " F H1F0""

#

#

H H L H ## + FI 1K/# 51;9,#

" " " " " " " " " " " " " " " " " " " 3" " "
2 0" " " " " " " " " " H +,0" "& " "
" " 40 80

H H D H H # + FI 1K/# 5193# # # # 51; ,#

I D H # # # + FI 1K/# 5184,#

I D D H H # # + FI 1K/# 5193# # # # 51; ,#

I D DL F H # + FI 1K/# 5184,#

D I D # # + FI 1K/# 5184,#

This constant is used to determine how many event flags your application will support.

E H G DE H # + FI 1K/# 51; ,#

This constant is used to determine whether you will enable (when 1) code generation for the D +, function.

H D H H # # + FI 1K/# 5193# # # # 51; ,#

H H ## # + FI 1K/# 513 ,#

H G DE H # + FI 1K/# 51; ,#

This constant is used to determine whether you will enable (when 1) code generation for the D +, function.

H H G DE H # + FI 1K/# 51; ,#

This constant is used to determine whether you will enable (when 1) code generation for the D +, function.

#

µC/OS-View µC/OS-View µC/OS-II µC/OS-View

H # # # # + FI 1K/# 51;4,#

"& " " " " "3" " " " " "2" " " " " 0

FI D # # # + FI 1K/# 51;4,#

"& " " " " " " " " " " " " " " " " 0" "

H " " " "4." " " "**AT LEAST**" "**2**" 0

FI D H H # # + FI 1K/# 51;4# # # # 51; ,#

"& " " " " " " " " " " 0"

FI KHH L H# # + FI 1K/# 51;4,#

"& " " " " " " " " " " " 0" " " " " " " " " " " "

"5" "435 0" " " " " " " " " " " " " " " " " 0" "

" 0"

" " " " " " " " " " 38/ " 0" " " " " " " " " " "

" " " 0" " " " " " " " " " " " " " " " " " 0" "

" " ." " " " FI D #2# 0

FI LFN H HF# + FI 1K/# 51;4,#

"& " " " " " " " " " " 0" " " " " " " " " "

" " " " " 0 0 LFN H HF 0 " " " " " " " "

FI LFN H HF" "43" 0 032" 0"

"

"

The following table summarizes some of the new & constants in FI 1K which were all added in since V2.00.

#

name in	... to enable the function(s):
D K N H #	D F K +,# D G K +,# D L K +,# D K +,# D K +,# D FEL K +,# D K +,
GHE H #	Enable debug constants in F H1F. If you are using a kernel aware debugger, you should enable this feature.
#	#
H H D H H #	H +,# H +,# And, to allow naming semaphores, mutexes, mailboxes and message queues.
H H L H #	H +,#
#	#
I D DFFH H #	I D +,#
I D GH H #	I G +,#
I D D H H #	I +,# I +,# And, to allow naming event flag groups.
I D H H #	I +,#
#	#
E DFFH H #	D +,#
E GH H #	G +,#
E H G DE H #	D +,#
E H #	+,#
E H #	+,#
E H H #	E +,#
#	#
H D H H #	+,# +,#
H H H #	+,#
#	#
H DFFH H #	D +,#
H GH H #	G +,#
H H H #	+,#

"

"

#	#
DFFH H #	D +,#
GH H #	G +,#
I K H #	I +,#
H G DE H #	D +,#
H #	+,#
I H #	I +,#
H #	+,#
H H #	+,#
#	#
H DFFH H #	D +,#
H GH H #	G +,#
H H G DE H #	D +,#
H H H #	+,#
H H H #	+,#
#	#
D N D H H #	+,# +,# And, to allow naming tasks.#
D N IL H H #	To allocate variables in FE for performance monitoring of each task at run-time.#
D N H H #	+,#
D N D N FKN H #	F +,#
D N K N H #	K +,#
D N N L H#	Size in N elements of the Timer Management task.#
#	#
LFN H H #	To support the stepping feature of **µC/OS-View**.#
#	#
L H G K H #	G K +,#
L H G H H H #	G +,#
L H H H H #	+, and +,#
L H LFN K N H #	K +,#
#	#
H #	Enables +4, or Disables +3, timer management functions.#
FI D #	Determines the maximum number of timers in your application.#
FI D H H #	Determines whether names can be assigned to timers.#
FI KHH L H#	Determines the size of the timer wheel (in number of entries).#
FI LFN H HF#	Rate at which timers will be updated (Hz)
#	#
FKHG FN H #	+, and +,#

#

#

New Data Types

F ## # # + F 1K/# 513 ,#

F L LFD H K G"%δ" " Ø" " ." " " " " "38/ " " " "

Ø

#

I D ## # # + F LL1K/# 5184,#

" " "L ; ."L 49 " "L 65 " " " " " ."38" "54" ." Ø

#

+ F LL1K/# 51;4,#

1F Ø

"

New Hook Functions

L K E + ,# # + F 1F/# 513 ,#

L +," " " " " " " "

µC/OS-II" " Ø

"

L K H + ,## # + F 1F/# 513 ,#

L +," " " " " " "µC/OS-II"

" Ø

"

FEL K + FE#- ,# + F 1F/# 513 ,#

FEL +," " " " " " " " Ø"

" " " " " " Ø

"

L K + ,# # + F 1F/# 5184,#

L +,Ø" " " " " " " " " " " " " "

" " " " " " Ø

"

"

New Functions

The following table provides a list of all the new functions (i.e. services) that YOUR application can call. The list also includes functions that used to exist but, if these are in this list, it's because their API may have changed.

Refer to the ***Reference Manual*** of the current release for a description of these functions.

Function Name	File	Enabled By …
H +,#	F H1F#	H H D H H #
H +,#	F H1F#	H H D H H #
H +,#	F H1F#	H H L H #
I D +,#	I D 1F#	I D H #))# I D DFFH H #
I F +,#	I D 1F#	I D H #
I G +,#	I D 1F#	I D H #))# I D GH H #
I +,#	I D 1F#	I D D H H #
I +,#	I D 1F#	I D D H H #
I +,#	I D 1F#	I D H #
I I +,#	I D 1F#	I D H #
I +,#	I D 1F#	I D H #
I +,#	I D 1F#	I D H #
G +,#	E 1F#	E H #))# E GH H #
D +,#	E 1F#	E H #))# E H G DE H #
+,#	E 1F#	E H #))# E H #
D +,#	H 1F#	H H #))# H DFFH H #
F +,#	H 1F#	H H #
G +,#	H 1F#	H H #))# H GH H #
+,#	H 1F#	H H #
+,#	H 1F#	H H #
+,#	H 1F#	H H #))# H H H #
D +,#	1F#	H #))# DFFH H #
G +,#	1F#	H #))# GH H #
I +,#	1F#	H #))# I K H #
+,#	1F#	H #
D +,#	1F#	H #))# H G DE H #
+,#	1F#	H #
I +,#	1F#	H #))# I H #
+,#	1F#	H #))# H #
F +,#	F H1F#	DIH F L LFD LHF9483;#
G +,#	H 1F#	H H #))# H GH H #
D +,#	H 1F#	H H #))# H H G DE H #
+,#	H 1F#	H H #))# H H H #
+,#	D N1F#	D N D H H #
+,#	D N1F#	D N D H H #
+,#	1F#	H #
+,#	1F#	H #
+,#	1F#	H #
+,#	1F#	H #
+,#	1F#	H #

"

References

µC/OS-II, The Real-Time Kernel, 2nd Edition
Jean J. Labrosse
CMP Books, 2002
ISBN 1-57820-103-9

Contacts

Micriμm
949 Crestview Circle
Weston, FL 33327
954-217-2036
954-217-2037 (FAX)
e-mail: 0 0
WEB: 0 0

CMP Books, Inc.
1601 W. 23rd St., Suite 200
Lawrence, KS 66046-9950
(785) 841-1631
(785) 841-2624 (FAX)
WEB: ll 0 0
e-mail: 0

"