µC3/Compact USERS GUIDE

5th Edition eForce Co., Ltd.

Introduction

μC3(Micro C cube)is a RTOS(Real Time Operating System)with kernel in accordance with specification of μITRON4.0, which is specified by TRON Association Corporation as Open Real Time kernel, in core.

The C3 of μC3 describes 3 concepts of **Compact , Connectivity, Capability.** Besides, the name of cube shows possibility of generating 3 effects by the concept above.

The position of this Document

This document is used as common manual for kernel function of μC3/Compact. However, another manual will be used for middle-ware or kernel function existing in each CPU. In case of necessity, please refer to these manuals. And, Have different functions depending on the version of the specification. Description of this case, the old kernel version is described as "Ver.1.x kernel", and the old configurator version is described as "Ver.2.x configurator".

TRON is abbreviation of "The Real-time Operation system Nucleus".
μITRON is abbreviation of "Micro Industrial TRON".
Specification of μITRON4.0 is available in homepage of TRON Association(http://www.assoc.tron.org/).
μC3 is a registered trademark of eForce Co., Ltd.
The contents of this document may be changed without prior notice.

Revision History

Modified items in 2nd Edition

Page	Content
	Changed layout

Modified items in 3rd Edition

Page	Content
	Changed layout
32	Added explanation regarding to selection of CPU
32,34,35,37,39, 41,42,4445,46, 47,49,50,52	Changed images together with version-up of configurator
35,37,39	Changed “Common part of kernel” to “Common kernel”
37	Changed explanation of Common kernel

Modified items in 4th Edition

Page	Content
23	Modified description of resource "Acquisition and return" of semaphore
28	Added explanation of System time update and Time unit used at kernel, Change Timing for, time-out
116	Corrected type name of parameter of COM port initialization function
123	Corrected “Receiving one character” to “Receiving character string” in title of Standard COM port driver system call
125,126	Modified “Time unit is mounting dependence” to “Time unit is 1 mili-second” in explanation of data type “TMO” “RELTIM” “SYSTIM”

Modified items in 5th Edition

Page	Content
60-95	Added a explanation of Version3 or later Configurator.

Table of Content

Chapter 1　What is μC3/Compact?

1. 1　Features

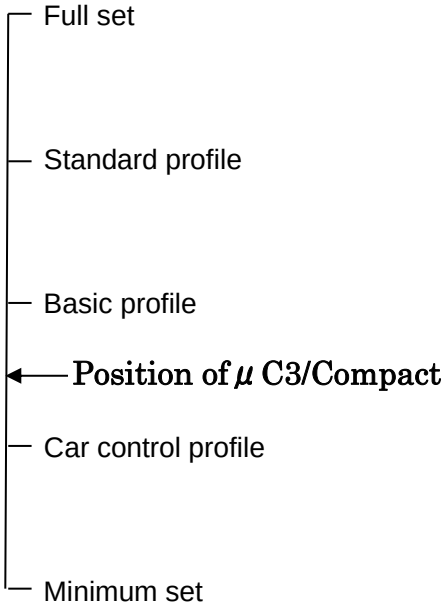
μC3/Compact is a product specialized in Compact in three concepts of μC3 to suppress memory consumption occupied in RTOS, especially, the consumption of RAM to the utmost limit. In other words, it is a product with concepts of saving memory only for build-in ROM/RAM, which is called one-chip microcontroller, as well as being able to be used for system which hesitate to adopt RTOS.

Only by generating objects statically, code size is suppressed, and management data arranged in RAM area is optimized and decreased. Besides, static API is not adopted, configurator of GUI method is used, necessary configuration data can be efficiently converted into management data, and the consumption memory is suppressed.

When designing application program using RTOS, though the allocated stack area of each task tends to make RAM area stringent, it supports to shared stack to mitigate the drawbacks mentioned above..

1. 2　Position in specification of μITRON

Though there is outline of profile in specification of μITRON4.0, μC3/Compact is being deviated from any profile. However, when defining development concept for μC3/Compact, a car control profile which is lower than a basic profile has been assumed to be basic.



As a specification that deviates from the car control profile, it is enumerated that there is not either CPU exception handler or static API because of the adoption of configurator of GUI method. In contrary, there are following supporting functions though they are not indispensable.

・System call with time-out
・Mailbox(The order of the message priority is non-supported)
・Fixed-Sized memory pool

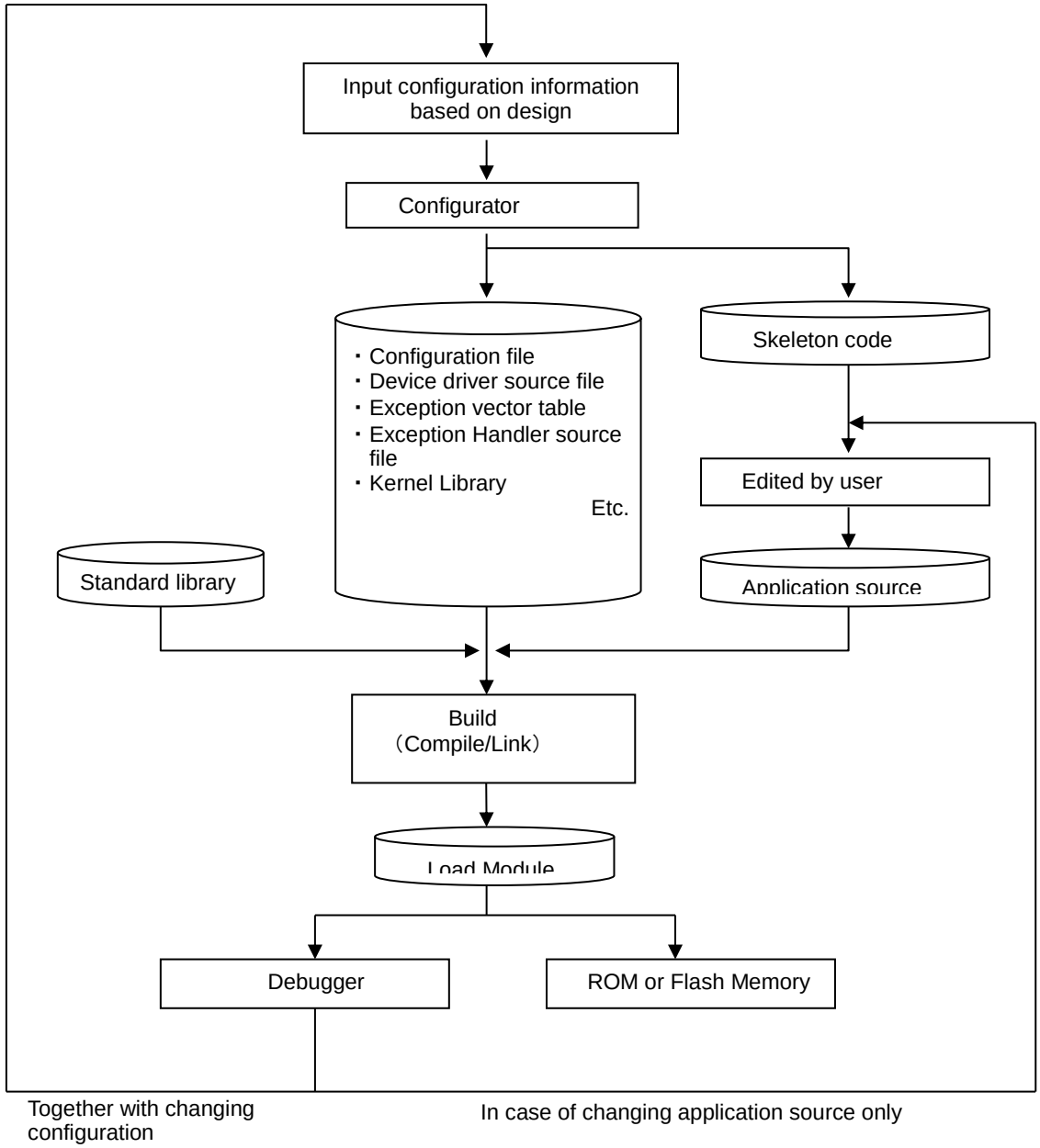
1．3 Development process

The following figure shows development process of system using μC3/Compact.

At first, input configuration information (number or attribute of necessary objects) of RTOS,which has been decided in system design, to configurator. Configurator generates code based on configuration information. In the generated code, there are skeleton code and file used without modification. This skeleton code is created to assist to describe necessary application program.

After describing application program, build (compile/link) and generate load module. Debug this load module, and write it to ROM or Flash memory to complete system when finishing debug. Moreover, in case there is some error in configuration of RTOS, retart configurator and start over again from configuration.

It is prohibited to modify and use file other than the skeleton code.



Development Process Figure

【Recommendation】

The configurator outputs skeleton code according to the code generation. Therefore, in case of directly editing skeleton code, it will be overwrited by the code generation due to change in configuration. In order to prevent from overwriting, it is recommended not to directly edit skeleton code but use template to create application program.

Chapter 2 Basic concept of μC3/Compact

2. 2 Glossary of basic terms

2. 1. 1 Task

A unit of a concurrent processing program is called "Task". In other words, multiple tasks are executed concurrently when seen from an application's point of view. In fact, concurrent program that is the number of processor, a kernel make it seem like concurrent processing by using time-sharing techniques following the scheduling rules. The task that invokes a system call is called the "invoking task" .

2. 1. 2 Dispatch and Scheduling

The act of switching the currently executing task on a processor with another, non-executing task is called "Dispatching". The mechanism in the kernel that performs dispatching is called the "Dispatcher".

The process that determines which task is to be executed next is called "Scheduling". The mechanism in the kernel that executes scheduling is called the "Scheduler".

Generally, Dispatcher and Scheduler is hardly separated in definition. In μC3, they are integrated and called "Dispatcher" and "Dispatch".

2. 1. 3 Context

The environment for program execution is called "Context", and each task, time event handler or interrupt handler is considered to have its own context. In case of switching from one context to another, it is general to use context as a register value of processor because data necessary to restart must be saved and be retrieved.

2. 1. 4 Object and ID number

The resources on which a kernel or a software component operates are generally referred to as Object, the numbers which are used to identify and distinguish objects are called ID Number. In μC3/Compact, configurator assign ID number, so that the system call is called out by using definition name of the identification number (macro name) in application. ID number of Object consist Object name + ID, such as Task ID, Semaphore ID.

Kernel objects include tasks, semaphores, Eventflags, Mailboxes Fixed-Sized memory pools, data queue, cyclic handlers, interrupt service routines and shared stacks. However, because there is no system call for reference, interrupt service routine has no ID number.

2. 1. 5 Service Call and System Call

The interface which invokes kernel or software component from application is called Service Call. In μC3, the Service Call of the kernel is called a System Call.

2. 1. 6 Priority order and priority level

The order relation decide the order of executing process, it is called “Priority order”, and parameter which is given by application for that process execution is called “Priority level” . Priority Level is displayed by numerical value (natural number), the less the value is, the bigger the Priority Level is, and vice versa.

In Priority Level of Task, there are Base Priority Level and Current Priority Level. In μC3/Compact, because Mutex is not implemented, the Base Priority Level and Current Priority Level will generally become the same Priority Level.

2. 1. 7 Restricted Tasks

By restricting some functionalitie of tasks by the task attribute, a restricted task can use a shared stack. A restricted task can not enter the WAITING state and the priority of a restricted task cannot be changed. Besides, it does not mean that if restricted task then shared stack must be used, even not using shared stack, there is still attribute of restriced task. However, if several restricted task want to share the same stack area, they need to have the same task priority.

2. 1. 8 Shared Stack

In case various tasks use the same stack space in the system with a little RAM area, that stack is called “Shared Stack”. In μC3/Compact, method of using restricted task difined by Car Control Profile and method for exclusively controlling of the kernel are prepared in order to be able to use shared stack safely.

2. 1. 9 Preemptive

If a task which has priority higher than the running task becomes ready, be able to dispatched, it is called “Preemptive”.

2. 1. 10 Time Tick

System time is controlled in kernel. The event at constant period for counting system time is called “Time Tick”. In other words, if a cycle of time tick is 1mm second, accuracy of system time is 1mm second and if it is a cycle of 2mm second, its accuracy is 2mm second.

2. 1. 11 Queuing

The maintaining function in case that there is some process requirement but it is not able to execute immediately is called “Queuing”, integrated as the counter to count the number of requires. In Queuing, there are activation request queuing and wakeup request queuing.

2. 1. 12 Queue

In case there is some required system call from a certain object, but the process is not able to be executed immediately, there will be system call which can wait till the process is executed or wait in a permitted time. In this kind of system call, it is queuing by the call of system call, and sequentially processed from the earliest one. This function is called “Queue”. Though there is task’s order in Queue, it does not support in μC3/Compact.

2. 2 Task States and Scheduling Rule

2. 2. 1 Task States

In μC3/Compact, task states are classified into 4 broad categories. The blocked state category can be further broken down into 2 sub-states. The RUNNING state and the READY state are both generically referred to as the runnable state“.

Task state transitions is shown in the following Figure:

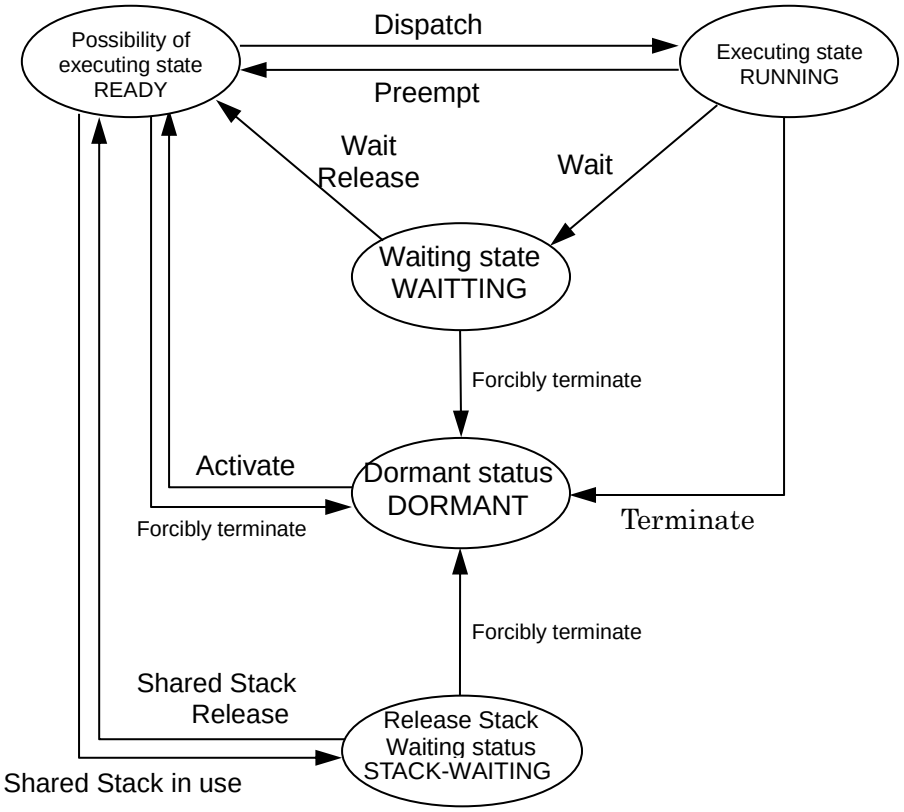


Figure of Task State Transitions

A RUNNING state

This state is when the task is currently executing. At the same time, only one task At the present, for status of execution, number of tasks changing to this executing status is up to one at the same time. Scheduler will decide from Task of possibility of executing status, and it will change to execution status based on Dispatcher. In other words, there is no change in the task of executing status even when changing into non-task context while executing the task.

B READY state

It is a state that though task is ready to excute but it is not being executed for some reason. In other words, it is the case of high Priority Order task which is in execution and the case when dispatch dose not happen. “The status of dispatching does not occur“ in μC3 means dispatching disabled state and interrupt mask which are raised more than the task level.

C Blocked state

In a status of waiting for some conditions, context of the task is stored in management area of the task so that it is possible to restart. In the status of waiting of wide sense, there are Waiting status and Shared Stack Release Waiting status.

C. 1 WAITING state

It is a status when an execution is interrupted due to no condition from system call. In details, there are Wake-up waiting, Time-passing waiting, Eventflag waiting, Semaphore waiting, Mailbox-message-receiving waiting, Data Queues-message-receiving/sending waiting, and Fixed-Sized-memory-block acquisition waiting.

C. 2 STACK-WAITING state

It is a status of waiting for releasing Shared Stack when it is in occupancy status depending on other tasks. Also, the status of releasing Shared Stack is different from waiting status, there is no release for Shared Stack release waiting in system call.

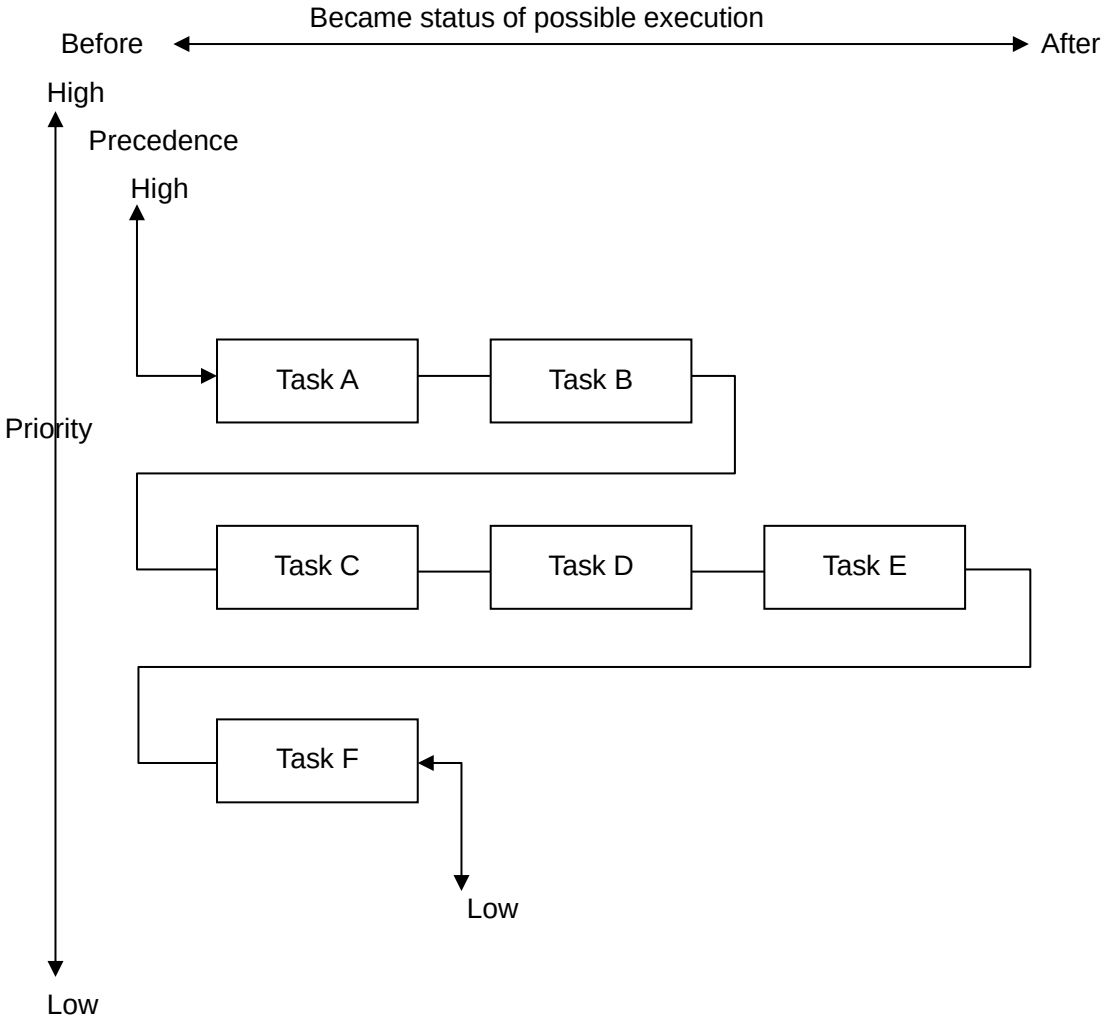
D DORMANT state

It is the status before the task is starting up or after the task ended. When it is in dormant status, information of executing status will not be saved. When it changes from dormant status to starting-up, execution will be starting from starting-up number of tasks.

2. 2. 2 Scheduling Rules

The preemptive priority-based scheduling is conducted based on the priorities assigned to tasks. If more than one runnable task exists, the highest precedence task will be in the RUNNING state. However, when the system is in a state where dispatching does not occur, the switch of the task in the RUNNING state will wait until dispatching is allowed.

The task in highest precedence is the one of highest Priority Level, and if there are a number of tasks with the same priority, it is the task in runnable state early. This relation is shown in the following chart. However, there might be change of Priority Order in tasks of the same Priority Level based on the call of system call (chg_pri,rot_rdq).



Precedence between Tasks

CHAPTER 3 Function Outline of μC3/Compact

3. 1 Context and System status

3. 1. 1 Process Unit and Context

Kernel of μC3/Compact is executed by the following process unit.

A. Interrupt handler

A.1 Interrupt service routine

B. Time Event Handler

C. Task

D. Idle

In μC3/Compact, interrupt handler is used only in the kernel, and interrupt process is described in Interrupt service routine.

Time Event Handler will only implement cyclic handler by processing activate based on time.

3. 1. 2 Task Context and Non-Task Context

The context which is a part of processing of the task is called Task Context and oppositely it is called Non-Task Context. In Non-Task Context, there are Interrupt service routine, Time event handler and Context executed by Idle.

In μC3/Compact, it is distinguishing System Call from Task Context, System Call from Interrupt Service Routine and System Call from Time Event Handler. It is impossible to call System Call from Idle. It is also impossible to use parameter or System Call specifying local task from Non-Task Context. Besides, system call with possibility of making task in waiting status in the wide sense cannot be called.

3. 1. 3 CPU Lock Status

In system status, there is status of either CPU Lock or CPU Unlock. In the status of CPU Lock, except the non-kernel interrupts, all other interrupts are prohibitted and dispatch is not happening. Moreover, in order to prohibit interrupt, starting-up of Time Event Handler is also reserved.

When it changes to status of locking CPU, it is called “Lock CPU”, and when it changes to status of unlock CPU, it is called “Unlock CPU . In detail, process of Lock CPU and Unlock CPU or status right after starting Interrupt Service Routine will be different depending on processor, please refer to “Processor dependence part Manual” for more explanation.

In case a System Call, which is possible to change to Waiting status of wide sense, is called in CPU Lock status, it is returned to E_CTX error. If CPU Lock is in realse status right after

starting execution of Time Event Handler and it is in CPU Lock status in application, then it is required that CPU Lock be in release status before returning from Handler.

CPU Lock will be in release status right after executing Task. Application is required to make CPU Lock in release status before ending local Task. Interrupt might be permitted even when CPU lock is in release status. That relation is different to processor, so please refer to “Processor dependence part Manual” for more explanation.

3. 1. 4 Dispatching Disabled State

System status will be either Dispatch Pended status or Dispatch permitted status. Dispatch is not happening in Dispatch Pended status.

When changing to Dispatch Prohibited status, it is called “Prohibit Dispatch” , and when changing to Dispatch Permitted status, it is called “Permit Dispatch” .

In Dispatch Prohibited status, when System Call, which is possible to make local task called from Task Context to Waiting status of wide sense, is called, it will return to E_CTX error. Also, it is not limited that System Call can be called from non-task context even in Dispatch Prohibited status.

Execution of Interrupt Service Routine, Time Event Handler gives no effect to Dispatch Prohibited/ Dispatch Permitted status. And Dispatch Prohibited/ Dispatch Permitted status of Task Context executing before will be still saved right after starting execution of these Handler/Routine. Besides, when System Call, which changes Dispatch Prohibited/ Dispatch Permitted status in these Handler/Routine, is called, it will return to E_CTX error.

3. 1. 5 Idle status

Idle is executed when there is no Task in runnable state, no Time Event Handler, no Interrupt Process, and that status is called “Idle Status”. Right after starting Idle execution, it will be in CPU Lock Release status, Dispatch Permitted status. In μC3/Compact, Idle status has its independent Context, but its characteristic is different from other Contexts. That different characteristic is not to preserve the Context when changing into other Contexts.

Because Context is not saved, Interrupt is happening in Idle execution and after executing Non-Task Context, it will not return to the place occuring Interrupt in Idle. In case of changing to Idle Context, it must be executed from the beginning of Idle.

Idle which prepares kernel is a simple loop without processing. In this Idle, it is possible to define Idle function by users in configuration.

3. 1. 6 Task State during Dispatch Pending State

Regarding to Dispatcher, there will be no Dispatch while executing process with high Priority Order, in CPU Lock status, while raising Interrupt level more than Task level, and in Dispatch Pending status. This status is called Dispatch Reservation status.

In Dispatch Reservation status, even there is Task of high Priority Order, this Task will not be dispatched. Dispatch of Task with high Priority Order will be reserved till Dispatch occurring status. While Dispatch is being reserved, all Tasks which have been executing till then will be in executing status, and after it becomes Dispatch occurring status, task that should be executed is in a ready condition.

Task status in Dispatch Reservation status is explained in the following chart.

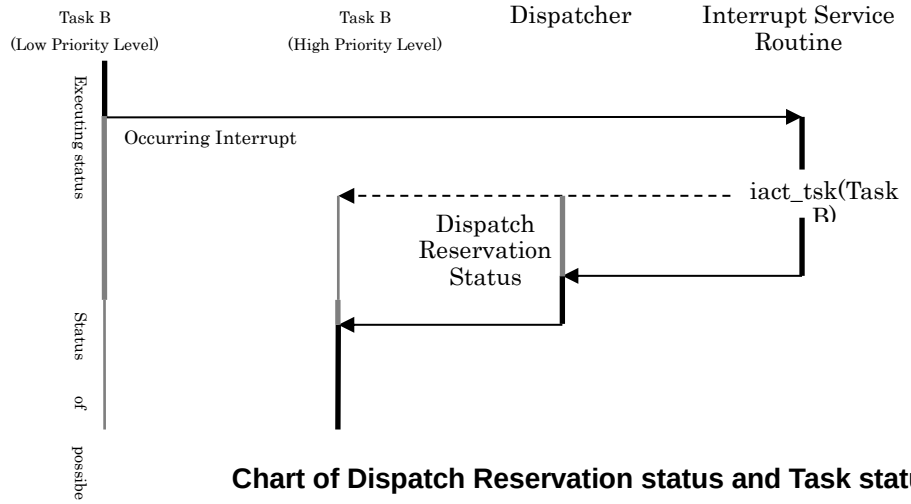


Chart of Dispatch Reservation status and Task status

There is a case in consideration when there is Task B with Priority Level that is higher than Task A, starting from Interrupt Service Routine by an interrupt occurring in execution of Task A. Because Priority Order of Interrupt Service Routine is higher than Dispatcher, even after Task B is started up, it will become Dispatch Reservation status and Dispatch is not occurring while Interrupt Service Routine is being executed. When execution of Interrupt Service Routine ends, Dispatcher is executed, then Task A changes to possible execution status and Task B changes into execution.

Even after Task B is started up in Interrupt Handler, Task A is still in execution status and Task B is possible execution status till Dispatcher is executed.

3. 2 Shared Stack

Shared Stack is Stack space which is possible to use for various Tasks. Also, Shared Stack is not able to use simultaneously for various Tasks, but it has function to control exclusively so as various Tasks are not able to use the same Stack space at the same time. There are 2 methods of exclusively controlling Shared Stack: method of specifying attribute of Restriction Task and method of using Stack Release Wating status without attribute of Restriction Task. However, it is possible for the same Shared Stack to let Task which has and has not attribute of Restriction Task to exist together.

3. 2. 1 Method of using attribute of Restricted Task

Specify attribute of Restriction Task to Task which is using Shared Stack; and Task using the same Shared Stack will specify all same Task Priority Level. By this, all Tasks using the same Shared Stack will be exclusively controlled once so that they are not dispatched till Task in execution status ends. However, there is a case in consideration when continuously starting-up Task A and Task B of Task Priority Level N that specifies a certain Shared Stack, and Task A has been dispatched.

Before ending Task A, if Task B is in executing status, these 2 Tasks are using Stack space at the same time and content of the Stack is destroyed. The reason that Task B is dispatched before ending Task A can be thought in a case when Task A is in Waiting status and there is change in Priority Order of Task Priority Level N. It is prevented by attribute of Restriction Task. It never comes off from Task Priority Order because Restriction Task is not changed to Waiting status. Also, there is chg_pri,rot_rdq as a System Call to which a change is added in the order of Task Priority Order. Therefore, when calling out chg_pri, by which specifies Restriction Task and rot_rdq which Priority Level of Restriction Task is specified by Task with the highest Priority Order, it will return to E_CTX error. In other words, chg_pri by which specifies Task A and rot_rdq by which specifies Priority Level N will return to E_CTX error.

3. 2. 2 Method of using Stack Release Waiting status

It does not specify attribute of Restriction Task but use Shared Stack. When the Task specifying Shared Stack is changing from possible execution status to executing status, if Shared Stack is not occupied with other Tasks, then occupy Shared Stack and it is dispatched; if Shared Stack is occupied with other Tasks, then change to Shared Stack Release Waiting status. If the Task which occupies Shared Stack has ended, then release the Shared Stack and change to possible execution status for Task with high Priority Order in Shared Stack Release Waiting status of that Shared Stack.

When this method is useful, it might be controlled exclusively by the application because it is

a Task that might not be executed at the same time, and there is a case that Task specifying Shared Stack is started-up. And in this case, even the Task is not confirmed to be changed to Dormant status safely, Shared Stack still can be used safely. Besides, when it is necessary to start the Task exclusively, it is possible to run it easily by specifying Shared Stack.

3. 3 Configurator

It has been decided in system design that configuration information becoming parameter of each Object would be input to configurator, and skeleton code becoming template of necessary source file and application program is generated. There are configuration information of kernel which is not depending on processor and configuration of device driver depended on processor. Besides, there are configuration information of common kernel and Object in configuration of kernel. Also, configuration of device driver is depending to processor, so please refer to “Processor dependance part Manual”, “Device dependence part Manual” for more explanation.

3. 3. 1 Configuration information of common kernel

Configuration information of common kernel is including the following items:

- Tick time specifying period of Time Tick.
- Number of Task Priority Level specifying upper value of Task Priority Level.
- Additional header file is added to include to configuration file.
- Idle function of user definition.
- Size of System Stack.
- Kernel mask level. (Only support "Ver.2.x kernel")

3. 3. 2 Configuration Information of Kernel Objects

Execute the following configuration corresponding to static API and configuration of creating Shared Stack:

- CRE_TSK Create Task
- CRE_SEM Create Semaphore
- CRE_FLG Create Even Flag
- CRE_DTQ Create Data Queue
- CRE_MBX Create Mailbox
- CRE_MPF Create Fixed-Sized Memory Pool
- CRE_CYC Create Cyclic Handler
- ATT_ISR Attach Interrupt Service Routine

3. 3. 3 Generated source code

In grand dividion, file is generated basing on thise configuration information. However, types of generated file will be different by configuration content.

- File which is not depending to generated processor.
- File which must depend to generated processor
- File depended to device driver

3. 4 Task Management Functions

Task management functions provide direct control of task states and reference to the task states. In detail, it is including the following functions:

- Activate a task(act_tsk,iact_tsk,sta_tsk)
- Terminate a task(ext_tsk,ter_tsk)
- Cancel activation requests(can_act)
- Change a task priority(chg_pri)
- Reference the task State(get_pri,ref_tsk,ref_tst)

Activation requests for a task are queued. In other words, if a task has already been activated and an activation request is made for the task, the new request is recorded. When the task terminates under this situation, the task will be automatically activated again.

However, activation requests will not be queued when the service call that activates a task with the specified start code (sta_tsk) is used. A task includes an activation request count to realize the activation request queuing. This activation request count is cleared to 0 when the task is created.

When a task is activated, its extended information (exinf) is passed as a parameter. However, when a task is activated by the service call with a start code (sta_tsk), the specified start code is passed through the parameter instead of the extended information.

When a task is activated, the task's base priority and current priority are initialized, the task's wakeup request count t is cleared.

The format to write a task in the C language is shown below:

```
void task(VP_INT exinf)
{
        /* Bod of the task */
        ext_tsk();
}
```

The behavior of a task returning from its main routine is identical to invoking ext_tsk, i.e. the task terminates.

The following kernel configuration constant is defined for use with task management functions:

TMAX_ACTCNT	Maximum activation request count (255)

3. 5 Task Dependent Synchronous Functions

Task dependent synchronization functions provide direct control of task states to synchronize tasks. In detail, it is including the following functions:

- Put a task to the sleeping state (slp_tsk,tslp_tsk)
- Wake up a task from the sleeping state (wup_tsk,iwup_tsk)
- Cancel wakeup request (can_wup)
- Forcibly release a task from waiting (rel_wai,irel_wai)
- Delay the execution of the invoking task (dly_tsk)

Wakeup requests for a task are queued. In other words, if a task is not in the sleeping state and a wakeup request is made for the task, the new request is recorded. When the task enters the sleeping state under this situation, the task will not be put in the sleeping state.

A task includes a wakeup request count to realize the wakeup request queuing. This wakeup request count is cleared to 0 when the task is activated.

The following kernel configuration constants are defined for use with task dependent synchronization functions:

TMAX_WUPCNT	Maximum wakeup request count (255)

3. 6 Synchronization and Communication Functions

Synchronization and communication functions provide synchronization and communication between tasks through objects that are independent of the tasks. The objects are semaphores, eventflags, data queues, and Mailboxes.

3. 6. 1 Semaphore

A semaphore is an object used for mutual exclusion and synchronization. A semaphore indicates the availability and number of unused resources by a resource count. In detail, it is including the following functions:

- Release resource (sig_sem,isig_sem)
- Acquire resource (wai_sem,pol_sem,twai_sem)
- Reference the state of a semaphore (ref_sem)

A semaphore has an associated resource count and a wait queue. The resource count indicates the resource availability or the number of unused resources. The wait queue manages the tasks waiting for resources from the semaphore. When a task releases a semaphore resource, the resource count is incremented by 1. When a task acquires a semaphore resource, the resource count is decremented by 1. If a semaphore has no resources available, or more precisely, the resource count is 0, a task attempting to acquire a resource will wait in the wait queue until a resource is returned to the semaphore or till the time allowed.

In order to avoid the case where too many resources are returned to a semaphore, each semaphore has a maximum resource count indicating the maximum number of unused resources available to the semaphore. If more resources are returned to the semaphore than its maximum resource count, an error will be returned.

The following kernel configuration constant is defined for use with semaphore functions:

TMAX_MAXSEM	Maximum value of the maximum definable semaphore resource count (255)

3. 6. 2 Eventflags

An eventflag is a synchronization object that consists of multiple bits in a bit pattern where each bit represents an event. In detail, it is including the following functions:

- Set an eventflag (set_flg,iset_flg)
- Clera an eventflag(clr_flg)
- Wati for an eventflag(wai_flg,pol_flg,twai_flg)
- Reference the state of an eventflag (ref_flg)

An eventflag has an associated bit pattern expressing the state of its events, and a wait queue for tasks waiting on these events. Sometimes the bit pattern of an eventflag is simply called an eventflag. A task is able to set specified bits when an event occurs and is able to clear specified bits when necessary. Tasks waiting for events to occur will wait until at least one bit or every bit in the eventflag bit pattern is set or till the time allowed. Tasks waiting for an eventflag are placed in the eventflag's wait queue.

The following data type is used for eventflag functions:

TBIT_FLGPTN The number of bits in an eventflag (depending on processor)

3. 6. 3 Data Queues

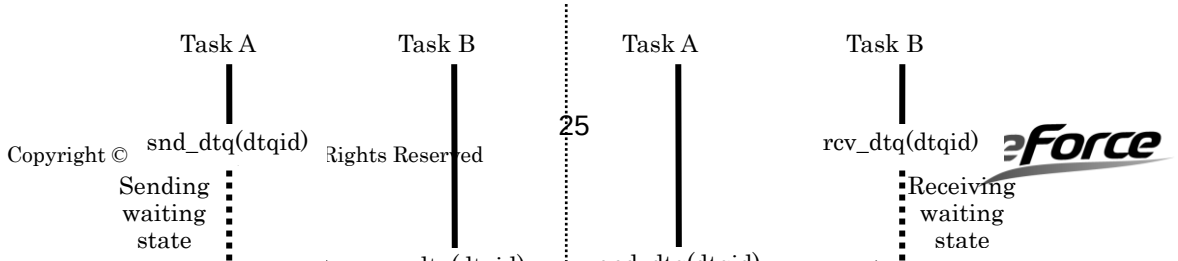
A data queue is an object used for synchronization and communication by sending or receiving a one word message, called a data element. In detail, it is including the following functions:

- Send a data element to data queue (snd_dtq,psnd_dtq,ipsnd_dtq,tsnd_dtq)
- Force-Send a data element to data queue (fsnd_dtq,ifsnd_dtq)
- Receive a data element from data queue (rcv_dtq,prcv_dtq,trcv_dtq)
- Reference the state of a data queue (ref_dtq)

A data queue has an associated wait queue for sending a data element (send-wait queue) and an associated wait queue for receiving a data element (receive-wait queue). Also, a data queue has an associated data queue area(ring-buffer) used to store sent data elements. A task sending a data element (notifying the occurrence of an event) places the data element in the data queue. If there is no room in the data queue area, the task will be in the sending waiting state for a data queue until there is room for the data element in the data queue area or till the time allowed.

Synchronous message passing can be performed by setting the number of data elements that can be stored in the data queue area to 0. The sending task and the receiving task wait unti the other calls the complimentary service call, at which time the data element is transferred. The one word data element to be sent and received can be an integer or the address of a message located in a memory area shared by the sender and the receiver. A data element that is sent and received is copied from the sender to the receiver.

If task A invokes snd_dtq first, task A is moved to the WAITING state until task B invokes rcv_dtq. During this time, task A is in the sending waiting state for a data queue. If, on the other hand, task B invokes rcv_dtq first, task B is moved to the WAITING state until task A invokes snd_dtq.



Synchronous Communication through a Data Queue

3. 6. 4 Mailboxes

A Mailbox is an object used for synchronization and communication by sending or receiving a message placed in a shared memory. In detail, it is including the following functions:

- Send a message to a Mailbox (snd_mbx)
- Receive a message from a Mailbox (rcv_mbx,pol_mbx,trcv_mbx)
- Reference the state of a Mailbox (ref_mbx)

Mailbox is including queue of waiting for receiving message and Message cue.
When a message is sent to Mailbox, message will be passed to a Task which is in receiving status, and the Task is changing from waiting status to possible execution status. In case there is no Task in waiting status, a message will be put to Message cue.

When a message is received from Mailbox, if there is message in Message cue, it will be taken out. If there is no message, it is connected to queue of waiting for receiving and changed to receiving-waiting status till a message is sent or till a permitted time. In fact, messages which is sent or received by Mailbox are only the ones of beginning number in memory, and content of sent or received message is not copied.

Transmission of messages in Mailbox is done by the first number of message packet prepared by application as a parameter. Besides, a message is received by the first number of message packet as return parameter. In detail, message packet is composed by a message header that kernel can specify order of message queue to the first field with the message itself used in the application that continues to it.

For example, T_MSGPKT type of message packet is defined as following:

```
typedef struct t_msgpkt{
    T_MSG*      pk_msg;      /* message header */
                             /* the message used by application */
```

} T_MSGPKT;

In application, content of message which is message queue is never rewritten. Also, a message which is already in Message cue must not be sent again to Mailbox. In case either of cases above is disobeyed, message will be destructed and it will lead to a fatal error.

【Recommendation】

It is possible to use memory block dynamically secured from the Fixed-Sized memory pool, and area secured statically as the message packet. As usage, it is recommend that Task of sending side will save memory block from memory pool to send as message packet; Task of receiving side will return that memory block directly to memory pool after taking out message content.

3. 7 Memory Pool Management Functions

Memory pool management functions provide dynamic memory management by software and it is including fixed-sized memory pools.

3. 7. 1 Fixed-Sized Memory Pools

A fixed-sized memory pool is an object for dynamically managing fixed-sized memory blocks. In detail, it is including the following functions: Fixed

- Acquire a memory block from a fixed-sized memory pool (get_mpf,pget_mpf,tget_mpf)
- Release a memory block to a fixed-sized memory pool(rel_mpf)
- Reference the state of a fixed-sized memory pool (ref_mpf)

A fixed-sized memory pool has an associated memory area where fixed-sized memory blocks are allocated (this is called fixed-sized memory pool area or simply memory pool area) and an associated wait queue for acquiring a memory block. If there are no memory blocks available, a task trying to acquire a memory block from the fixed-sized memory pool will be in the waiting state for a fixed-sized memory block until a memory block is released or till the time allowed. The task waiting to acquire a fixed-sized memory block is placed in the fixed-sized memory pool's wait queue.

In case memory block is returned, application will return it to Fixed-Sized memory pool of same ID number which had been gained memory block, and the first number of gained memory block must be used in the time of return. Also, memory block which has already been returned must not be overlapped to return. In case either of cases above is disobeyed, management information of the Fixed-Sized memory pool will be destructed and it will lead to a fatal error.

3. 8 Time Management Functions

Time management functions provide time-dependent processing. It is including each function of system time management and cycle handlers.

3. 8. 1 System Time Management

System time management functions provide control over system time. In detail, it is including the following functions:

- Set and get the system time (set_tim,get_tim)
- Supply a time tick for updating the system time (isig_tim)

The system time is initialized to 0 when the system is started and will be updated every time isig_tim is invoked by the application. However, System Time is managed by mili-second unit, and all time specified by System Call or configurator is in mili-second unit.

The following features depend on the system time: processing of timeouts, releasing tasks from waiting after a call to dly_tsk, and activation of cyclic handlers. However, even changing System Time by setting System time (set_tim), time-out time of System Call, which has already been called, will not be changed.

3. 8. 2 Cyclic Handlers

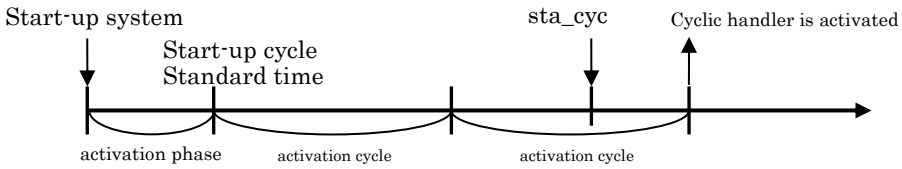
A cyclic handler is a time event handler activated periodically. In detail, it is including the following functions:

- Start a cyclic handler's operation (sta_cyc)
- Stop a cyclic handler's operation (stp_cyc)
- Reference the state of a cyclic handler (ref_cyc)

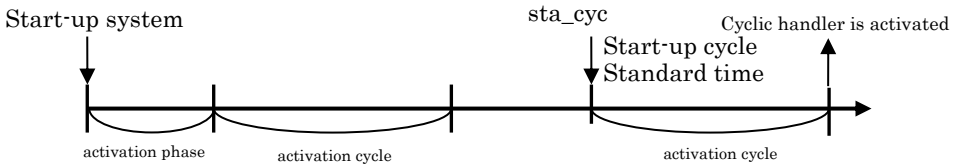
Status of cycle handler will be either operation status or non-operation status.

When starting up System, if TA_STA attribute has been specified, it will be in operation status. Also, if either TA_STA attribute or TA_PHS attribute is specified, Time when starting-up phase has been added at the system starting time will become the time that should start nextly. In case neither of the attribute is specified, Time which has been added starting-up phase to the time called by System Call (sta_cyc) starting in operation of cycle handler at the system starting time will become the time that should start nextly. Then that extended information of cycle handler is made as parameter (exinf) to start up cycle handler. At this time, Time which has been added starting-up phase to the time starting-up cycle handler will become the time that should start nextly.

In the status of non-operation of cycle handler when there is TA_PHS attribute, even it becomes the time of start up cycle handler, cycle handler will not be start up, and only time which should start up nextly is decided. When System Call (sta_cyc) which starts operation of cycle handler is called, it will change to operation status of cycle handler, Time that cycle handler should be started nextly is decided again if necessary. When System Call (stp_cyc) which stops operation of Cycle Handler is called, it will be changed to non-operation status of Cycle Handler.



(a)When the activation phase is preserved(with TA_PHS specifying)



(b) When the activation phase is not preserved (without TA_PHS specifying)

Preserving Activation Phase

Starting-up cycle of Cycle Handler is based on time which should have for starting up Cycle Handler (not the time of being already started), it is accepted as phase time specifying nextly starting-up time for Cycle Handler. Therefore, The interval of time when the Cycle handler is started might become individually shorter than starting cycle, but it is consistent with starting cycle when averaging the long period.

The n time of starting-up Cycke Handler is secured to be executed after System Call generating Cycle Handler is called and more than time of (starting phase＋starting cycle × (n-1)) has passed. The start of times n of A guarantees to do after the time of B or more passes after the system call that generates A is called. For example, in a System which cycle of Time tick is 10 mili-second, if generating a Cycle Handler with 15 mili-second of starting phase and 25 mili-second of starting cycle, then System time started up by Cycle starting Handler will be 20 mili-second, 40 mili-second, 70 mili-second, 90 mili-second, 120 mili-second.

Cycle Handler is described in the following form:

```
void cychdr(VP_INT exinf)
{
        The Cycle Handler
}
```

3. 9 System State Management Functions

System state management functions provide control of and reference to the various system states. In detail, it is including the following functions:

- Rotate task precedence (rot_rdq,irot_rdq)
- Reference the ID of the task in the RUNNING state (get_tid,iget_tid)
- Lock and unlock the CPU (loc_cpu,iloc_cpu,unl_cpu,iunl_cpu)
- Enable and disable dispatching (dis_dsp,ena_dsp)
- Reference the context and the system state (sns_ctx,sns_loc,sns_dsp,sns_dpn,ref_sys)

3. 10 Interrupt Management Functions

Interrupt management functions provide management for interrupt service routines started by external interrupts. In detail, it is including the following functions:

- Disable and enable an interrupt (dis_int,ena_int)
- Change and reference the interrupt mask (chg_ims,get_ims)

The following data types are used for interrupt management functions:

INTNO	Interrupt number
IMASK	Interrupt Mask

A part of IMASK in data type of Interrupt Mask is different in content by processor's architecture. Also, implementation function of disabling/enabling an interrupt(dis_int,ena_int) is different by processor. Please refer to “Processor dependence part Manual” for more explanation.

When calling an interrupt service routine, the extended information (exinf) of the interrupt service routine is passed as a parameter.

The format to write an interrupt service routine in the C language is shown below:

```
void isr(VP_INT exinf)
{
        /* Body of the interrupt service routine */
}
```

3. 11 System Configuration Management Functions

System configuration management functions provide management for the system configuration and version information. In detail, it is including the following functions:

- Reference the system configuration (ref_cfg)
- Reference version information (ref_ver)

CHAPTER 4 Usage of Configurator

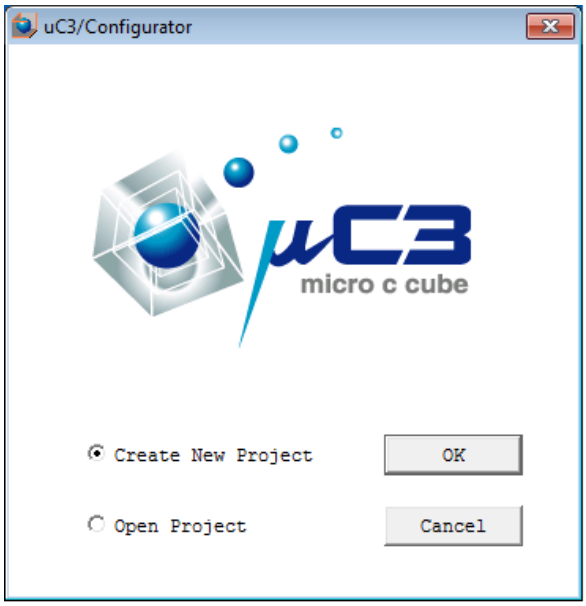
4. 1 Operation of the configurator : "Ver.2.x configurator"

4. 1. 1 Starting up Configurator

Please double click “µC3conf.exe” to start up.

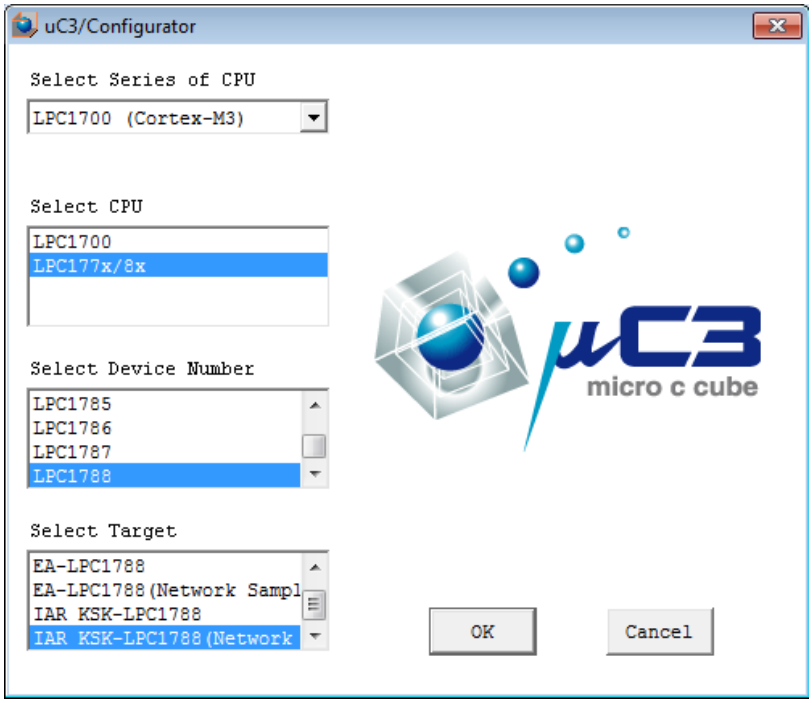
A. In case of creating a new project

After selecting “Create a new project” , click “OK” and go to “Select CPU” .



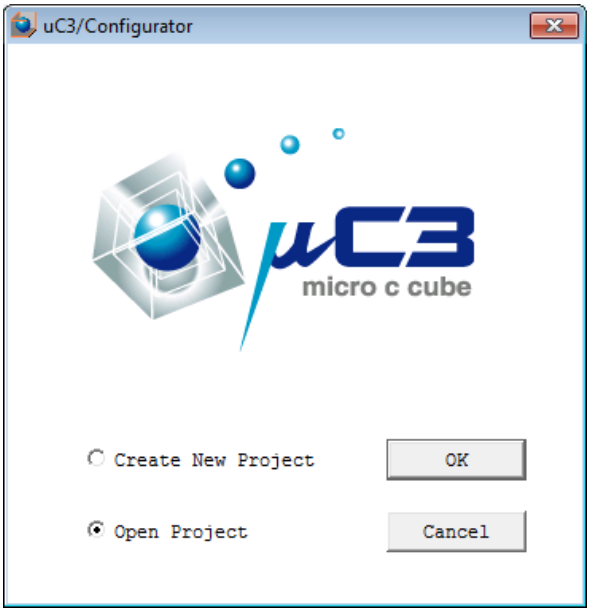
Select CPU

After selecting CPU series, CPU, serial number, target in List, click “OK” and go to “Main screen” .



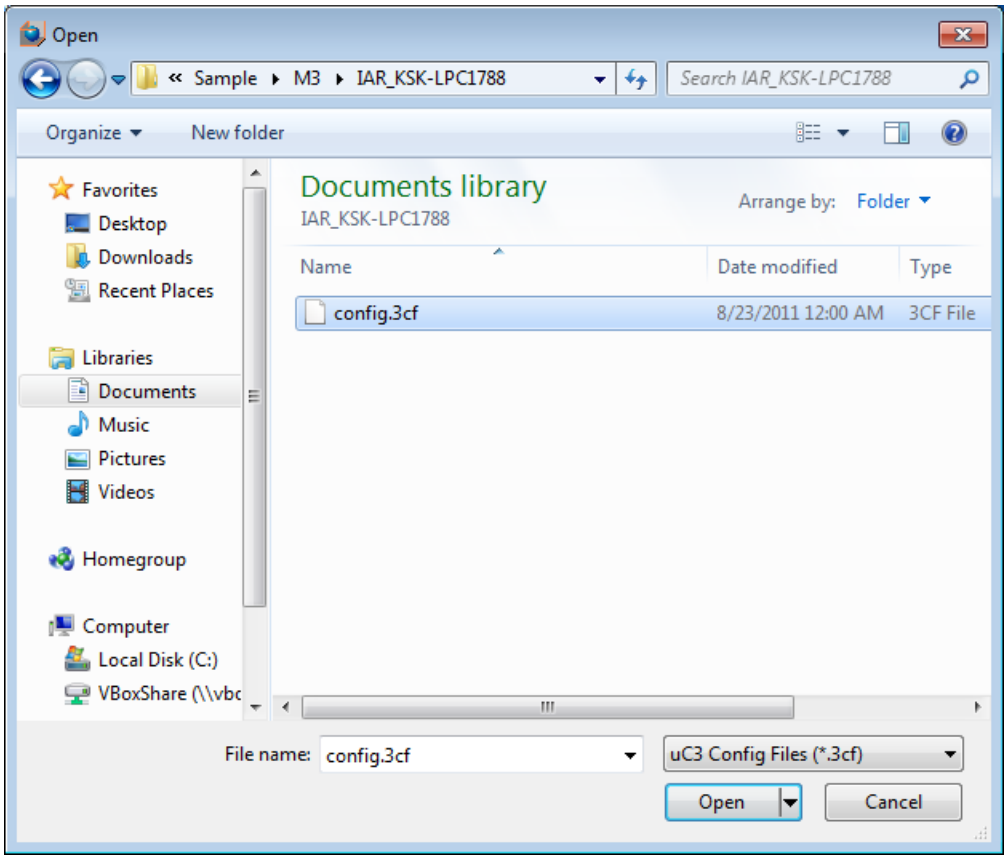
B. In case of opening an existing project

After selecting “Open existing project” , click “OK” and go to “Open file” .



Open file

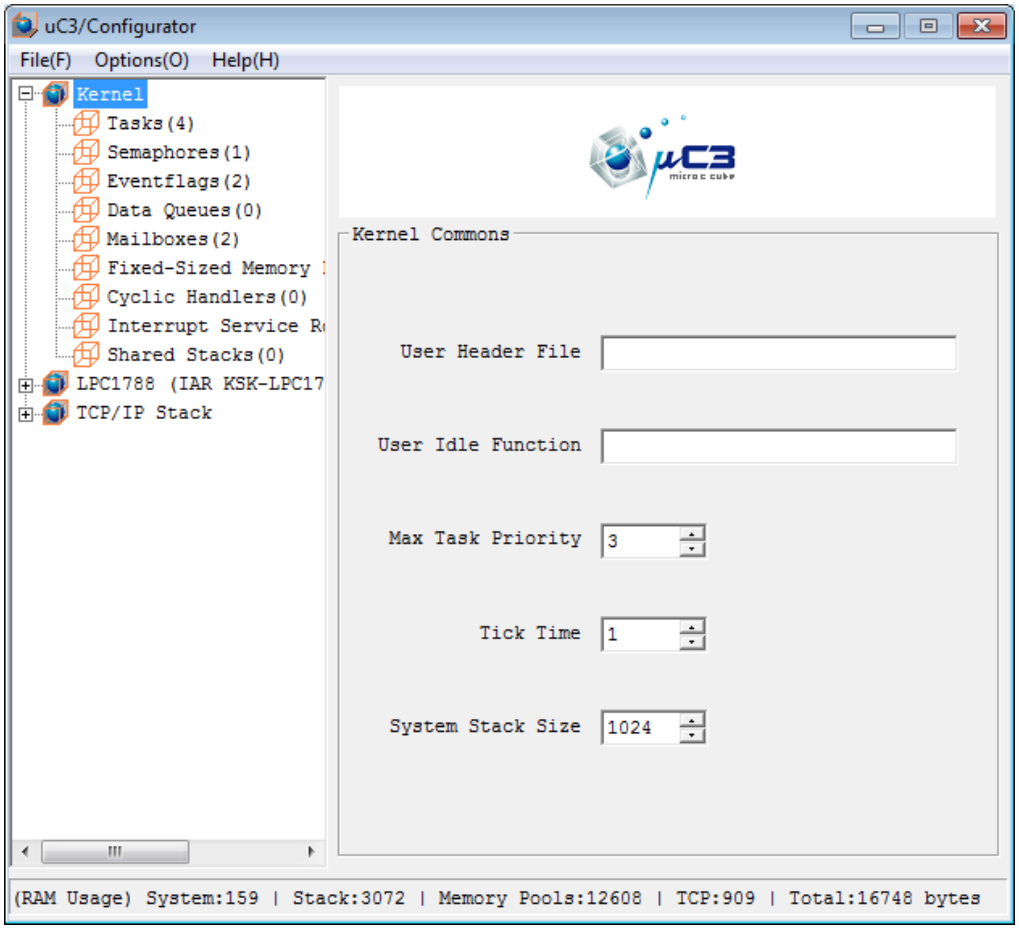
After selecting a saved project file(extension.3cf), click “Open” and go to “Main screen” .



C. Main screen

After starting up, it will go to the main screen where it is possible to refer or edit project. By clicking to each Object of Tree Display, it will switch to each Object Configutation screen.

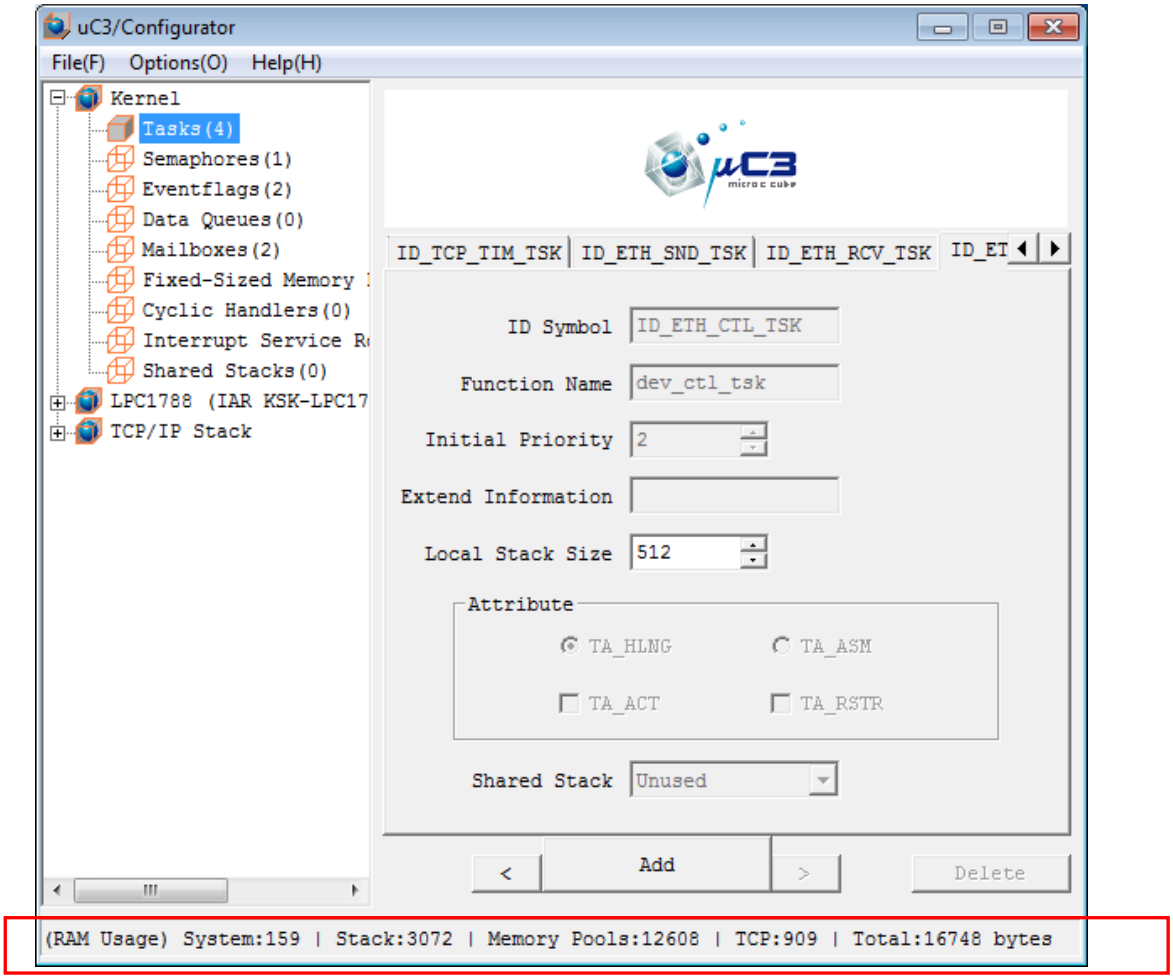
Here, there is configuration of kernel or processor dependence part. Please refer to “Processor dependence part Manual” for more explanation.



4. 1. 2 Set-up kernel

In kernel configuration, there are configurations of common kernel and Objects such as Task, Semaphore etc…. In configuration screen of each Object type, 1 Object is corresponding to 1 tab.

At the bottom of status bar, it often displays using capacity of memory which is managed by kernel. The following figure is an example of Configuration screen of an Object.



In each item, there is tab which is displayed in grey and impossible to change or to select “Delete” button. This kind of tab is Object which is added and synchronized to configuration of device driver.

“Add” button

To add a new Object or its corresponding tab.

“Delete” button

To delete Object of tab which is selected at the moment.

“←” button

Move to the left of a tab which is selected at the moment.

“→” button

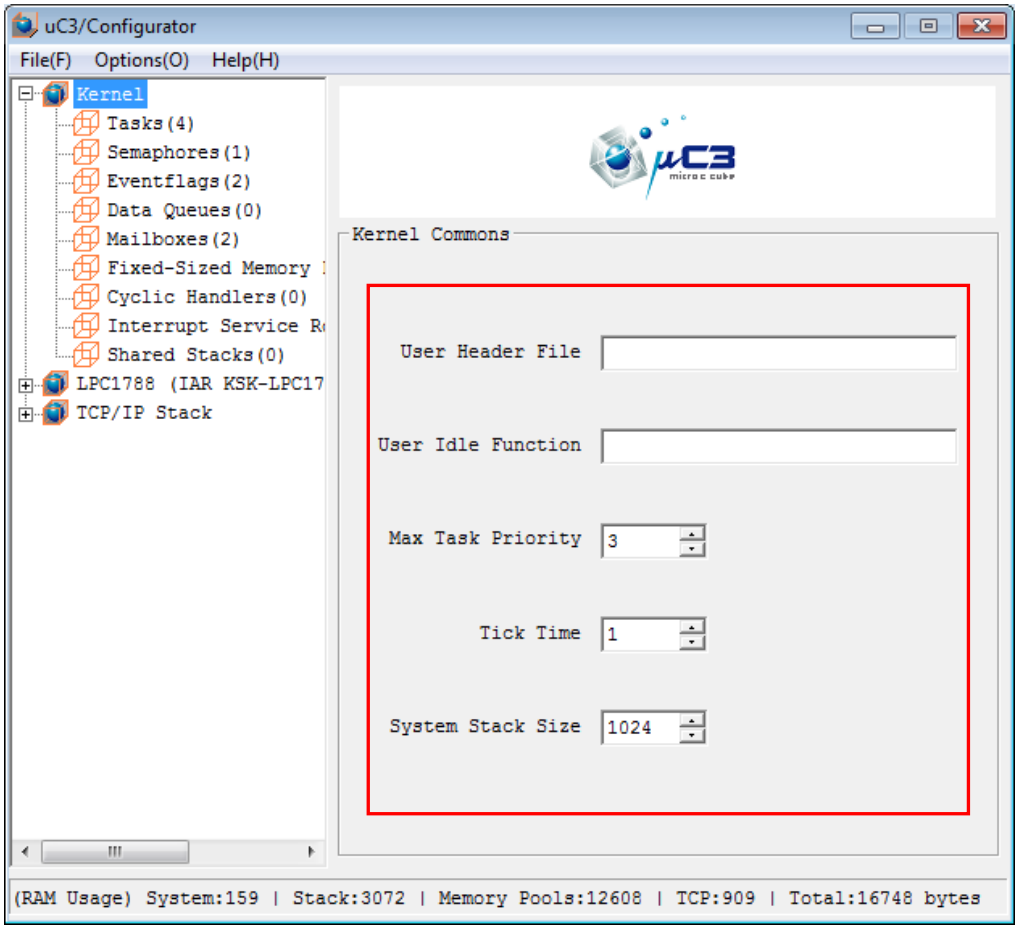
Move to the right of a tab which is selected at the moment.

Memory using capacity

Item	Content
System	Memory used by kernel itself, management area of Object, buffer of Data Queues.
Stack	System stack, Individual Task stack, Shared stack
Memory pool	Memory pool area of Fixed-Sized memory pool

4. 1. 2. 1 Configuration of common kernel

When clicking to kernel of Tree Display, configuration screen of common kernel will be displayed to execute configuration for common kernel.



Add header file

When defining pointer to value of macro or variable as extended information of Task and Cycle Handler, file name of header file which describes external declaration of macro definition or variable will be specified. In detail, when a file name is specified here, that file will be included in kernel_cfg.c.

Idle function

Specify that function name when not using standard Idle function of internal kernel but replacing it with Idol function of user definition.

Number of Task Priority Level

It is possible to specify from 1 to 16, and Task Priority Level which is upper to this value.

Tick Time

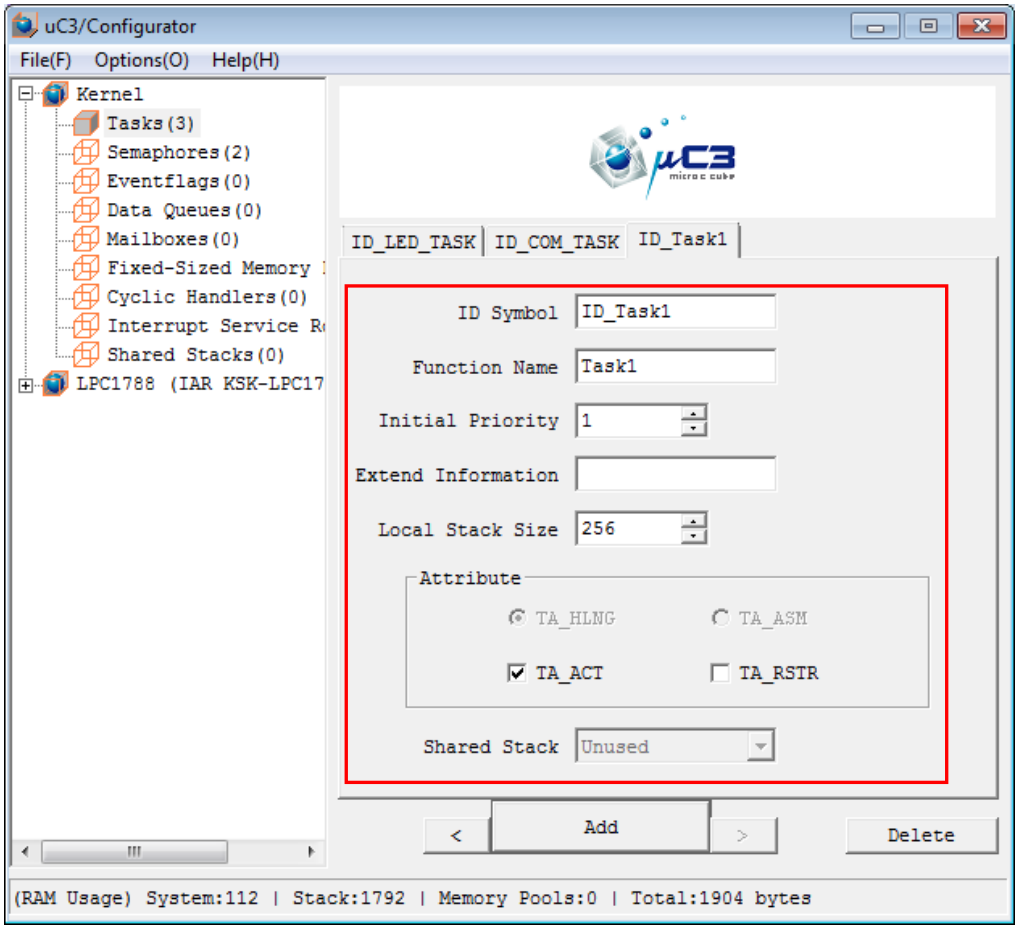
Cycle of Time Tick is specified in mili-second unit. The less the value is, the more accuracy the time is, but the overhead will be bigger.

System Stack Size

Size of Stack area used by Time Event Handler and interrupt Service Routine is specified in byte unit.

4. 1. 2. 2 Configuration of Task

When clicking to Task of Tree Display, configuration screen of Task will be displayed, configuration corresponding to CRE_TSK of Task creating API will be executed.



ID Symbol

Please specify optional definition name displaying ID number of Task. This definition name is macro-defined in kernel_id.h.

Function name

Specify function name of option Task.

Initial value of Priority Level

When starting up Task, please specify value of initializing Task Priority Level which is not exceeding Task Priority Level number of common kernel. When specifying shared stack and attribute of Restriction Task(TA_RSTR=ON), the same task priority as other tasks which specified the shared stack will be specified.

Extension information

In case there is extension information which is passing to Task, then specify it, and leave it it

blank if it is unnecessary. In extension information, it is possible to specify numerical value, value which is macro-defined and pointer to variable. In case of passing pointer to variable, attach "&" to the beginning of variable name.

Stack size

Please specify size of peculiar stack to Task. When a field of peculiar stack, which is out of "Not use" , is specified, it will become invalid and impossible to change.

TA_HLNG/TA_ASM

It will become fixed TA_HLNG, and impossible to change in μC3/Compact.

TA_ACT

When checking, TA_ACT attribute will become ON, and Task is created in possible execution status.

TA_RSTR

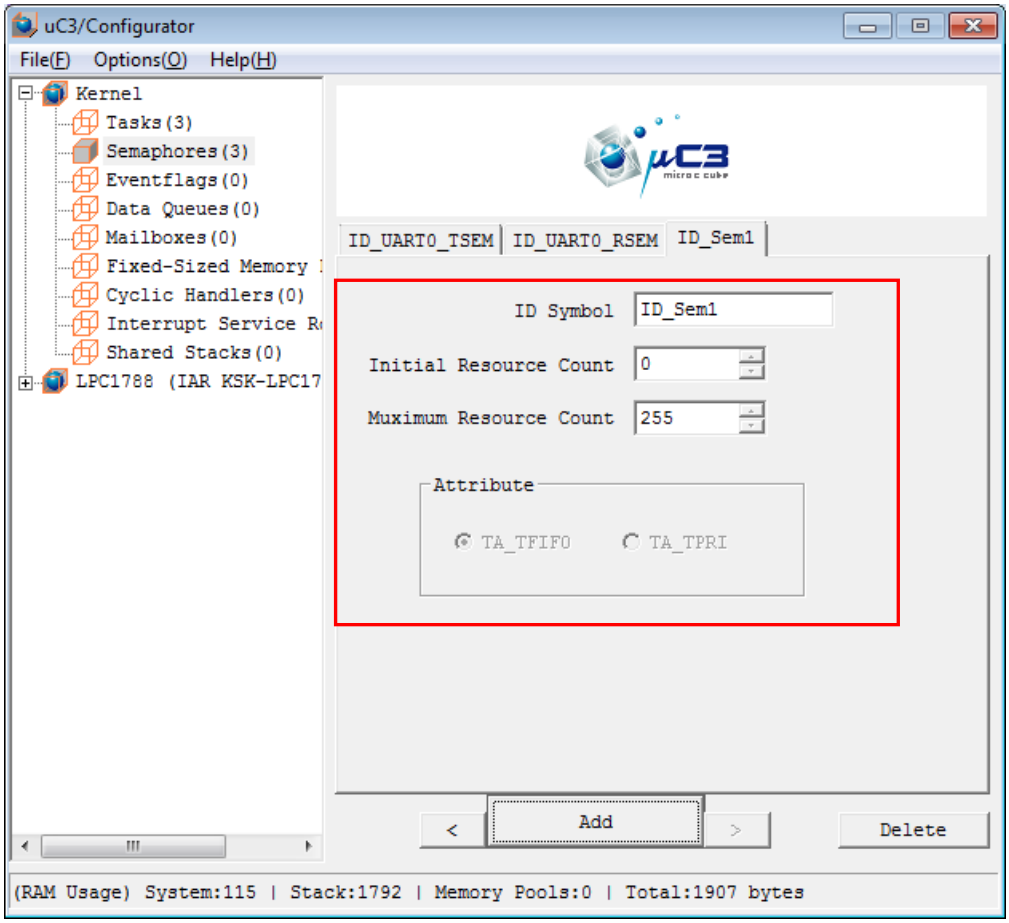
When checking, TA_RSTR attribute will become ON, and attribute of Restriction Task is given. When selecting the shared stack which was described later, it will be automatically checked, but it is possible to remove the check later. In case shared stack is used by various Tasks, all those Tasks must be either TA_RSTR=ON, or TA_RSTR=OFF.

Shared stack

Regarding to field of shared stack, if there are more than 1 shared stack defined, it will be possible to select that definition name.

4. 1. 2. 3 Configuration of Semaphore

When clicking to Tree Display of Semaphore, configuration screen of semaphore will be displayed for configuration which is corresponding to CRE_SEM of semaphore creating API.



ID Symbol

Please specify optional definition name which displays ID number of semaphore. This definition name is macro-defined in kernel_id.h.

Initial value of resource number

Specify initial value of semaphore count which is not exceeding maximum resource number.

Maximum resource number

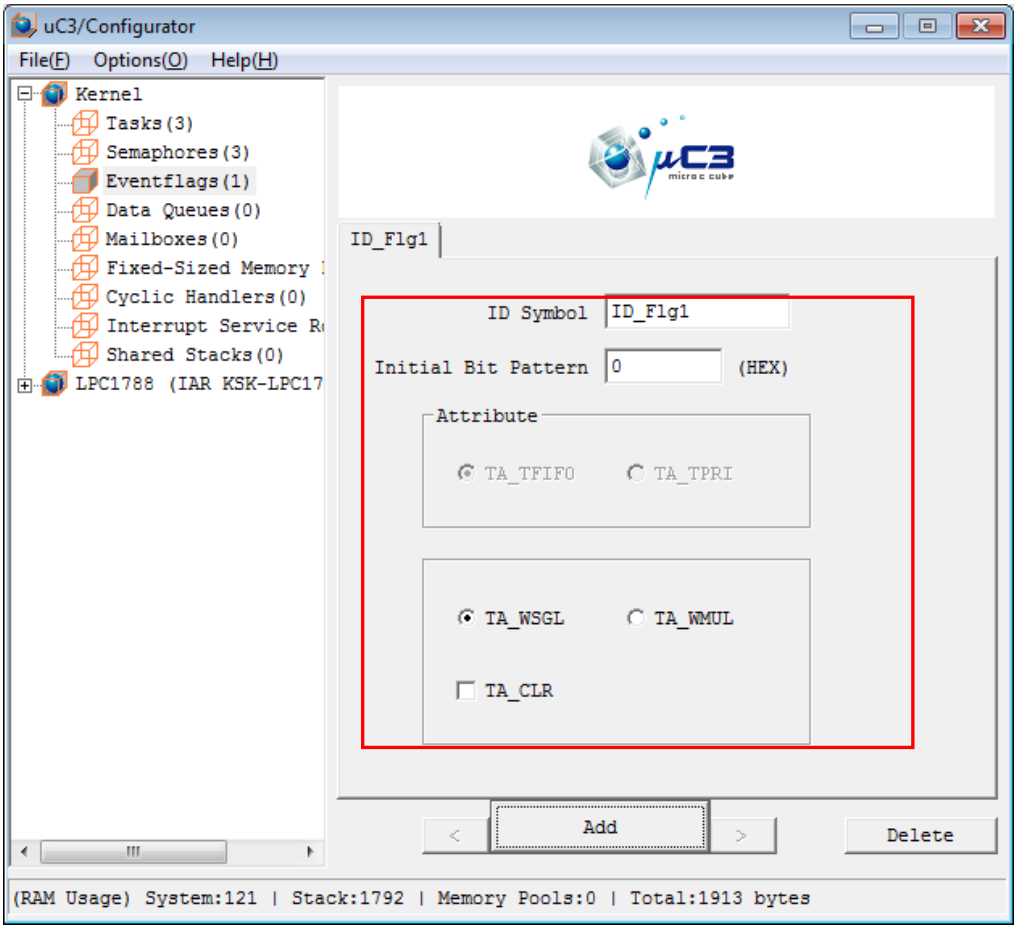
Please specify maximum value of semaphore count. The maximum value which can be specified is 255.

TA_TFIFO/TA_TPRI

It will become fixed TA_TFIFO and impossible to change in µC3/Compact.

4. 1. 2. 4 Configuration of Eventflag

When clicking to Tree Display of Eventflag, configuration screen of Eventflag will be displayed for configuration which is corresponding to CRE_FLG of Eventflag creating API.



ID Symbol

Please specify optional definition name which displays ID number of Eventflag. This definition name is macro-defined in kernel_id.h.

Initial value of bit pattern

Please specify initial value of Eventflag by hexadecimal number.

TA_TFIFO/TA_TPRI

It will become fixed TA_TFIFO and impossible to change in μC3/Compact.

TA_WSGL/TA_WMUL

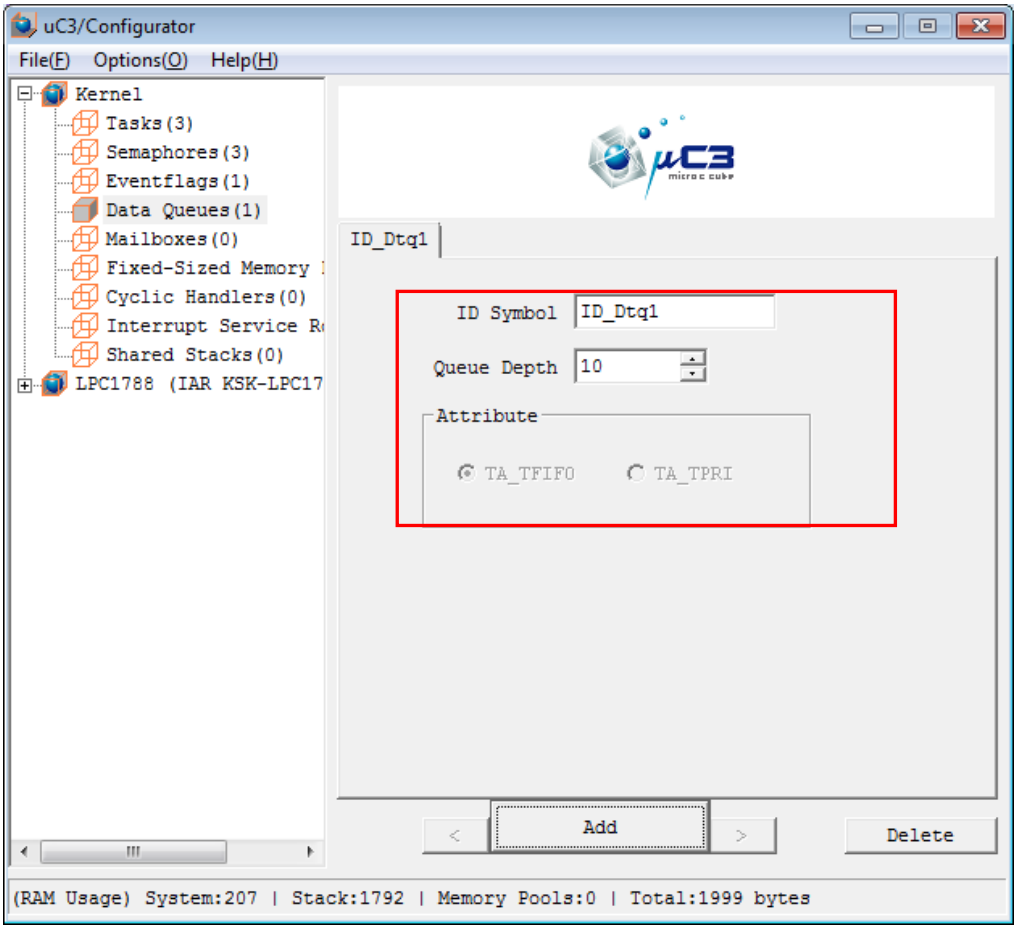
By specifying TA_WSGL, waiting of various Tasks will be prohibited. By specifying TA_WMUL, waiting of various Tasks will be permitted.

TA_CLR

If checking, attribute of TA_CLR will be ON, and when Task is released from Eventflag waiting by a condition approval, all bits of bit pattern are cleared.

4. 1. 2. 5 Configuration of Data Queues

When clicking to Tree Display of Data Queues, configuration screen of Data Queues will be displayed for configuration which is corresponding to CRE_DTQ of Data Queues creating API.



ID Symbol

Please specify optional definition name which displays ID number of semaphore. This definition name is macro-defined in kernel_id.h.

Number of data

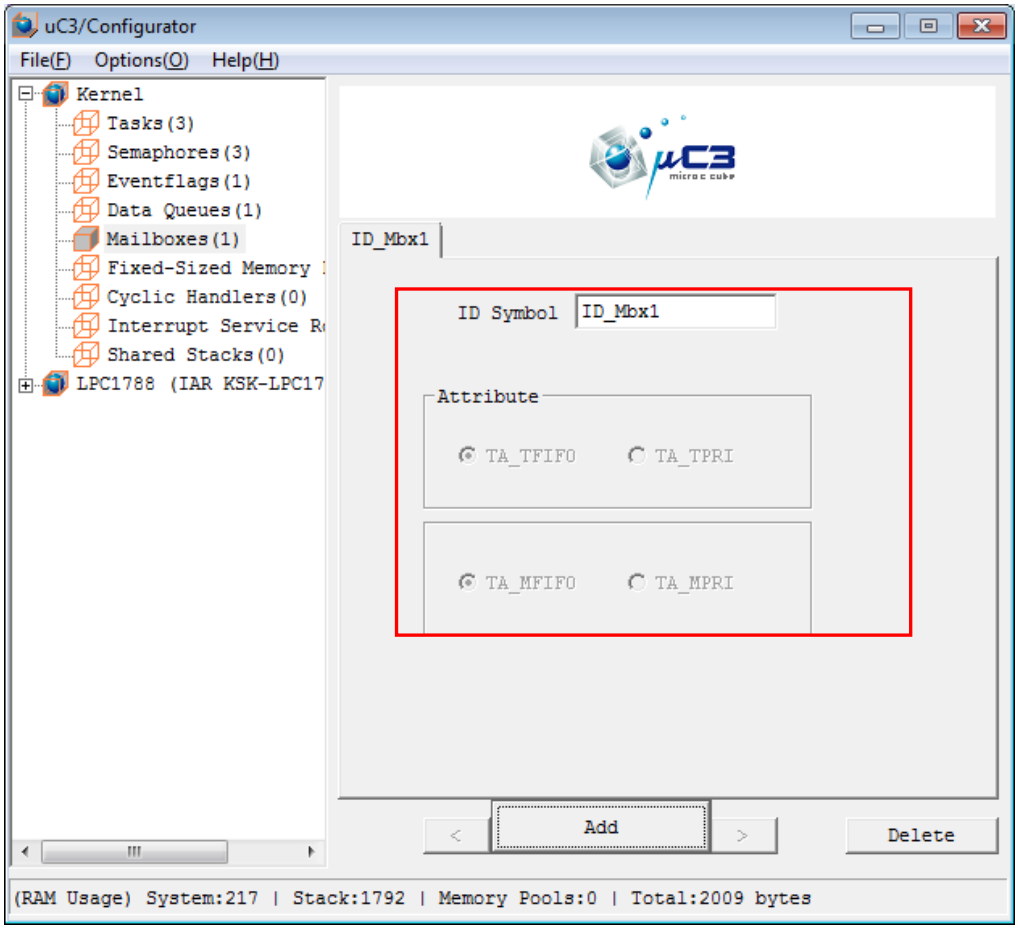
Spacify number of Data Queues (number of data).

TA_TFIFO/TA_TPRI

It will become fixed TA_TFIFO, and impossible to change in μC3/Compact.

4. 1. 2. 6 Configuration of Mailbox

When clicking to Tree Display of Mailbox, configuration screen of Mailbox will be displayed for configuration which is corresponding to CRE_MBX of Mailbox creating API.



ID Symbol

Please specify optional definition name which displays ID number of Mailbox. This definition name is macro-defined in kernel_id.h.

TA_TFIFO/TA_TPRI

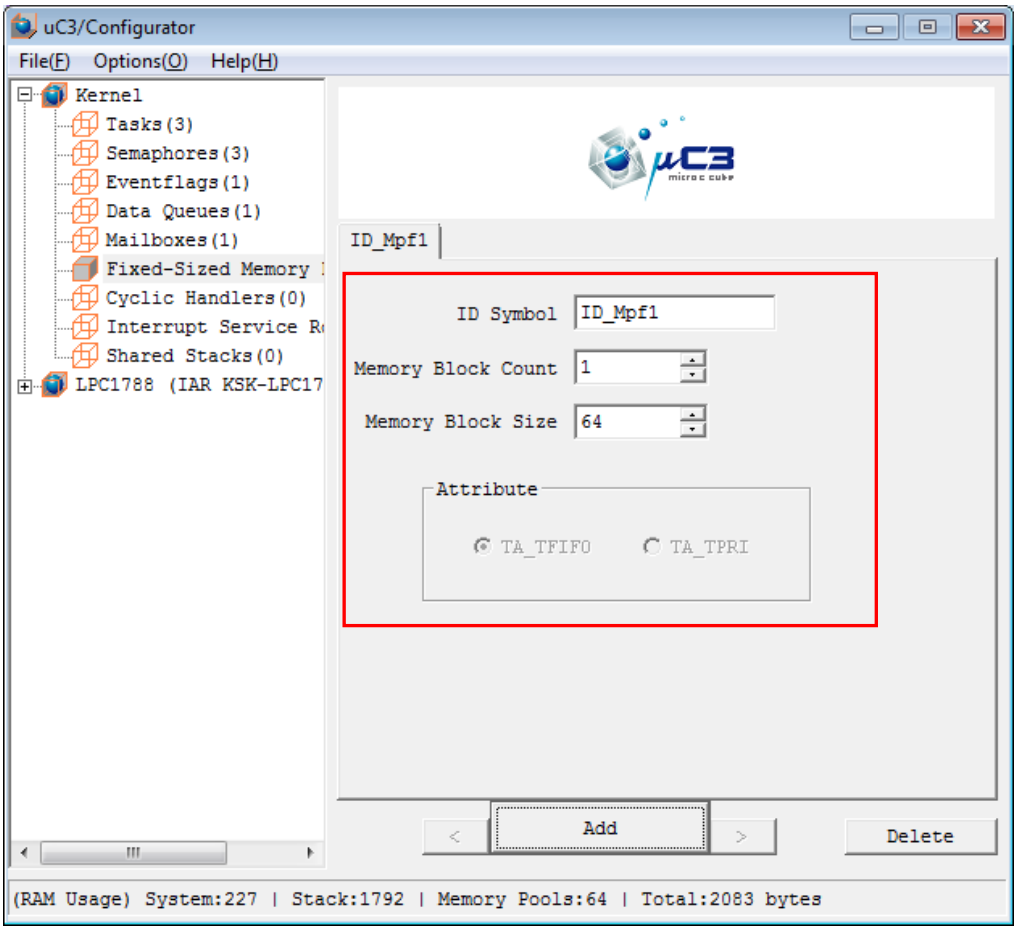
It will become fixed TA_TFIFO and impossible to change in µC3/Compact.

TA_MFIFO/TA_MPRI

It will become fixed TA_MFIFO and impossible to change in µC3/Compact.

4. 1. 2. 7 Configuration of Fixed-Sized memory pool

When clicking to Tree Display of Fixed-Sized memory pool, configuration screen of Fixed-Sized memory pool will be displayed for configuration which is corresponding to CRE_MPF of Fixed-Sized memory pool creating API.



ID Symbol

Please specify optional definition name which displays ID number of Fixed-Sized memory pool. This definition name is macro-defined in kernel_id.h.

Number of memory block

Specify number of memory block.

Size of memory block

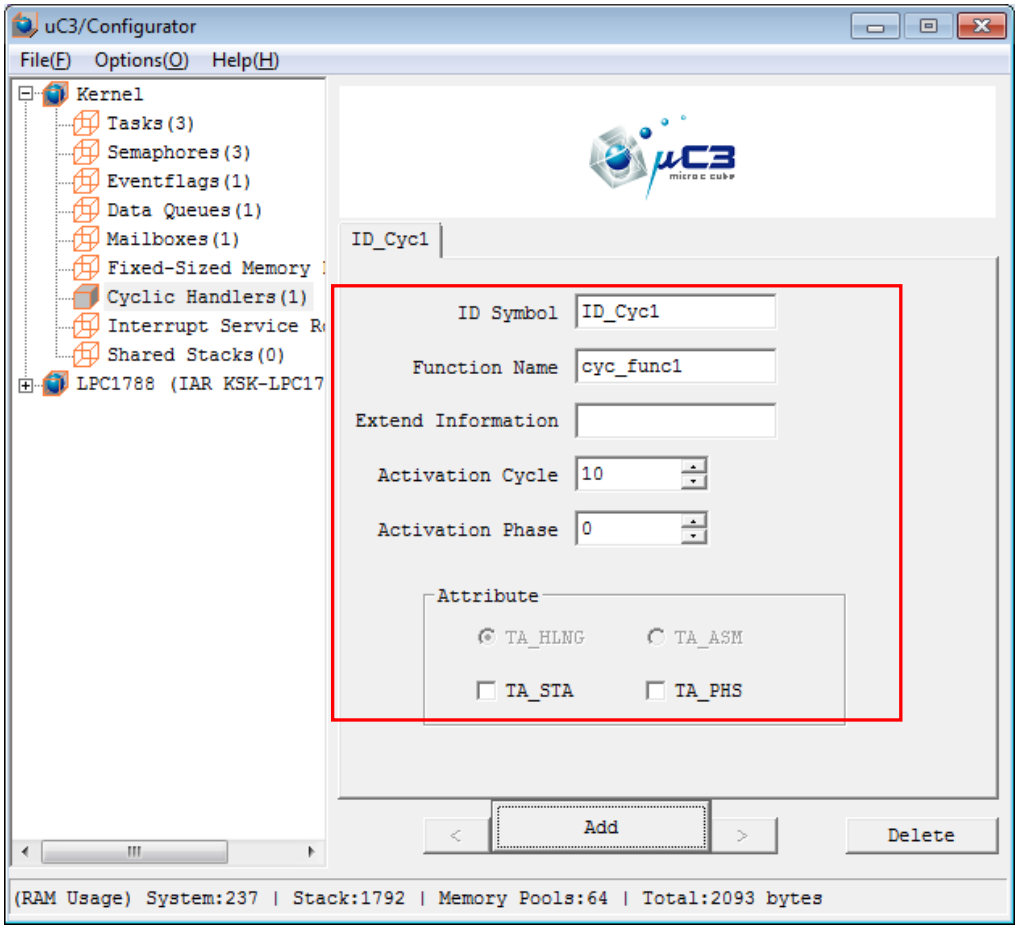
Specify size of memory block (number of byte).

TA_TFIFO/TA_TPRI

It will become fixed TA_TFIFO and impossible to change in μC3/Compact.

4. 1. 2. 8 Configuration of Cycle Handler

When clicking to Tree Display of Cycle Handler, configuration screen of Cycle Handler will be displayed for configuration which is corresponding to CRE_CYC of Cycle Handler creating API.



ID Symbol

Please specify optional definition name which displays ID number of Cycle Handler. This definition name is macro-defined in kernel_id.h.

Function name

Specify function name of optional Cycle Handler.

Extension information

If there is extension information which is passing to Cycle Handler, specify it, or in case of unnecessary, just leave it in blank. In extension information, it is possible to specify numerical value, macro-defined value, pointer to variable. If passing pointer to variable, attach “&” to the beginning of variable name.

Starting-up cycle

Starting-up cycle of Cycle Handler is specified by mili-second unit. However, small value

cannot be specified by Tick time.

Starting-up phase

Starting-up phase of Cycle Handler is specified by mili-second unit.

TA_HLNG/TA_ASM

It is impossible to change in μC3/Compact.

TA_STA

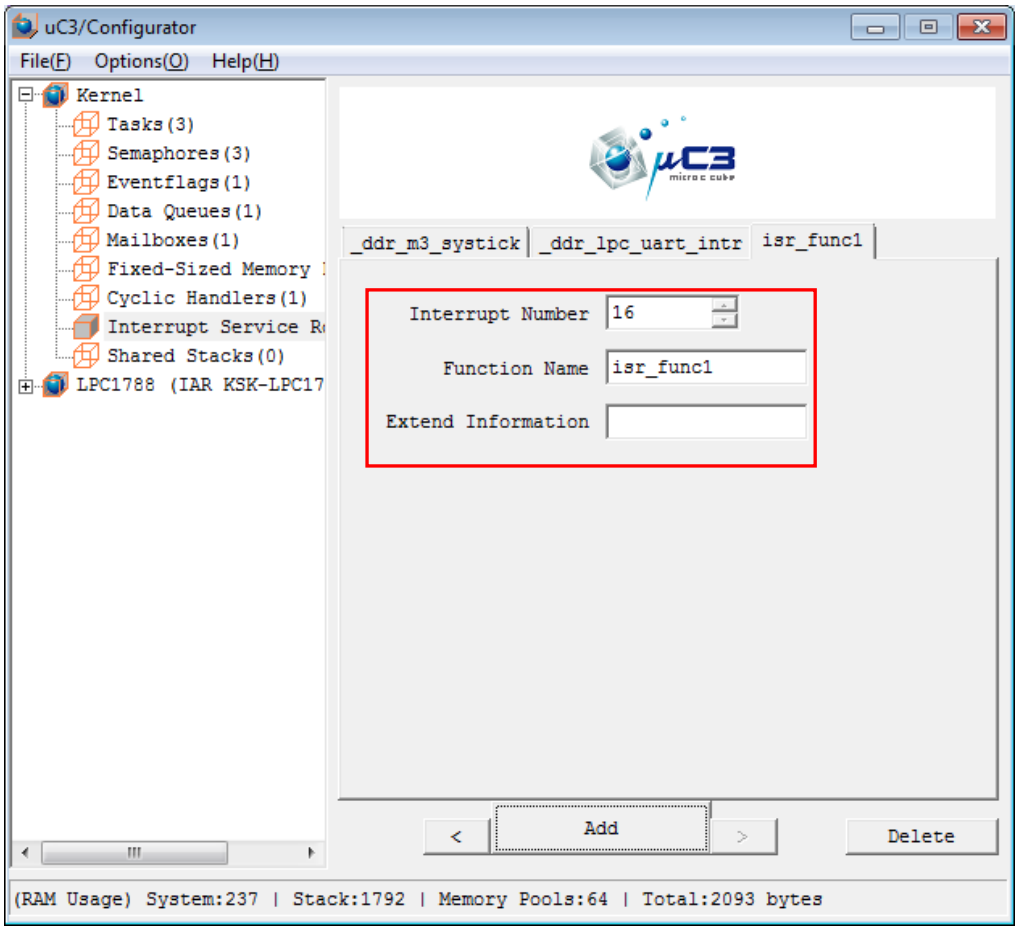
If checking and attribute of TA_STA is ON, Cycle Handler is generated by operation status.

TA_PHS

If checking and attribute of TA_PHS is ON, phase when generating Cycle Handler is saved.

4. 1. 2. 9 Configuration of Interrupt Service Routine

When clicking to Tree Display of Interrupt Service Routine, configuration screen of Interrupt Service Routine will be displayed for configuration which is corresponding to ATT_ISR of added API of Interrupt Service Routine.



Interrupt number

Please specify interrupt number. When configurating various Interrupt Service Routine to the same interrupt number, calling order will follow order of tab and the more it is on the left, the faster it will be called.

Function name

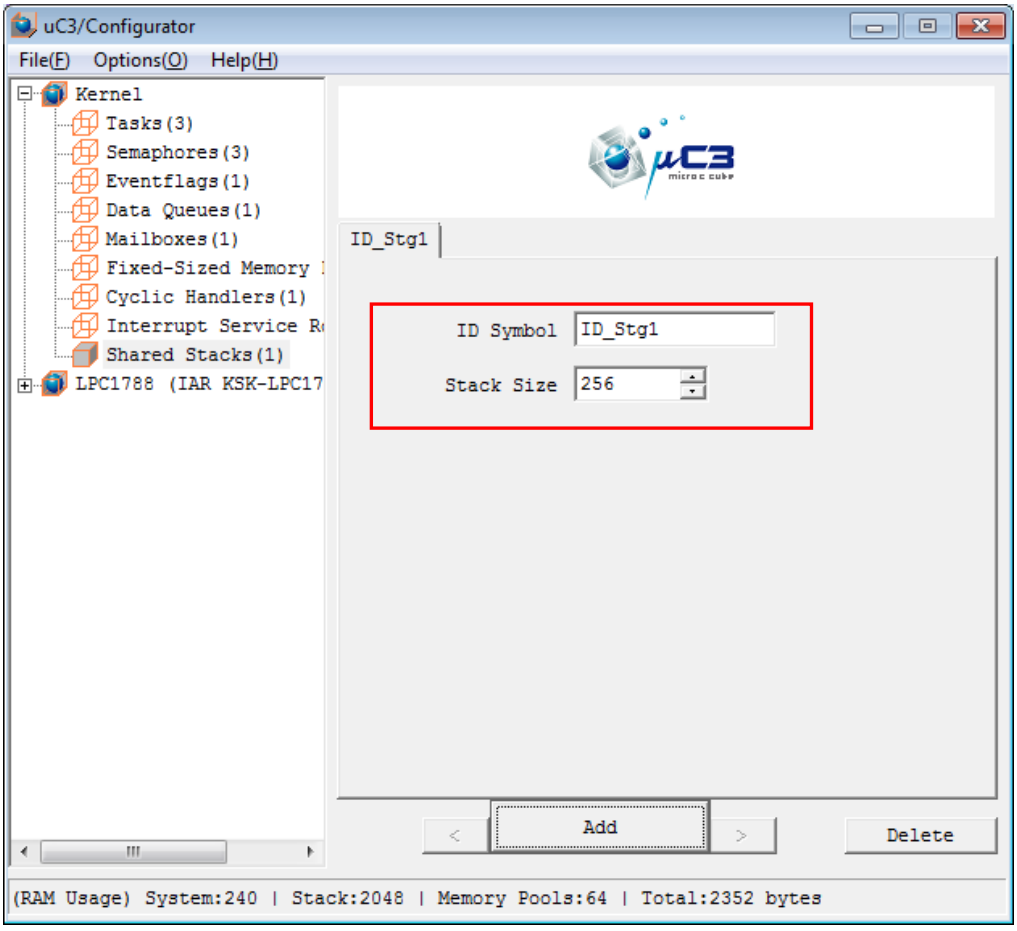
Specify function name of optional Interrupt Service Routine.

Extension number

If there is extension information which is passing to Interrupt Service Routine, specify it, or in case of unnecessary, just leave it in blank. In extension information, it is possible to specify numerical value, pointer to variable.

4. 1. 2. 10 Configuration of Shared stack

When clicking to Tree Display of Shared Stack, configuration screen of Shared Stack will be displayed for configuration of Shared Stack.



ID Symbol

Please specify optional definition name which displays ID number of Shared Stack. This definition name is used to select Shared Stack in configuration screen of Task.
In case there is even 1 Task using this Shared Stack, it will be impossible to change definition name.

Size of Stack

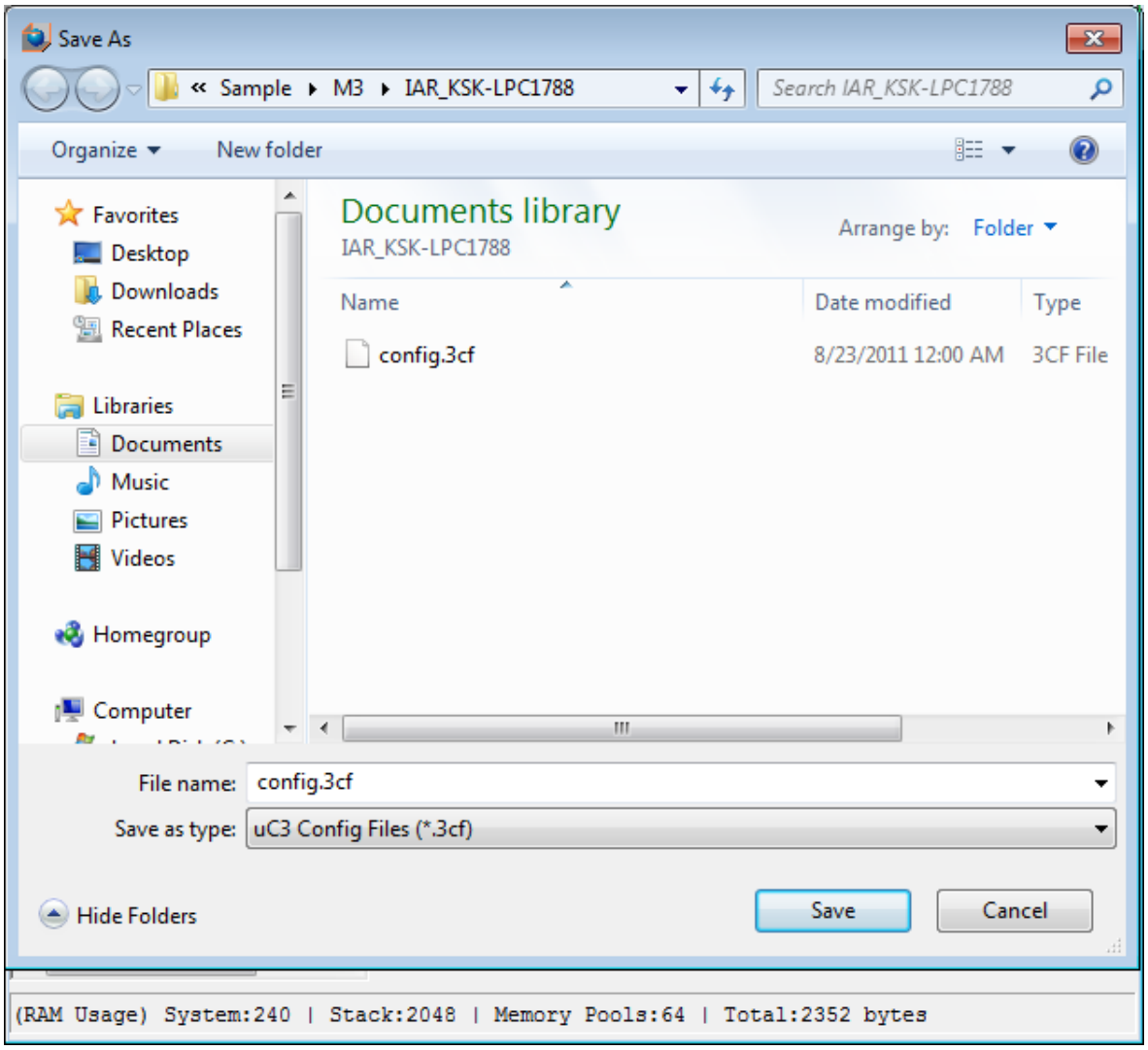
Specify size of Shared Stack (byte number). The Stack size of Task selecting the use of Shared Stack is fixed to size of Shared Stack. Therefore, Stack size of Task which uses the most Stack is specified by the Task specifying this Shared Stack.

Deletion

In case there is even 1 Task using that Shared Stack, display warning message and it will be deleted. In that case, Shared Stack of the Task is changed to “Not use” .

4. 1. 3 Saving project file

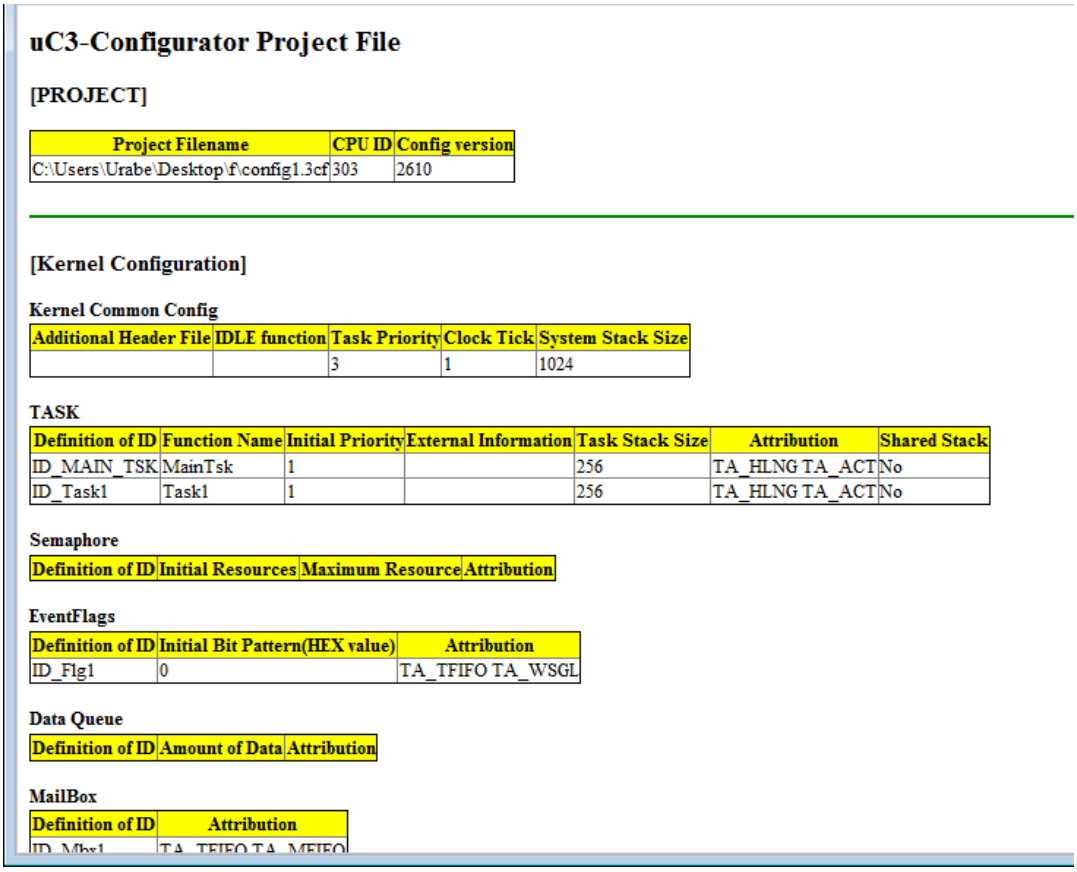
From “File” → “Save…(S)” , open “name and save screen” , specify saving folder for project file and click “OK” .



Regarding to the saved file, the file that changed project file (default config..3cf)and extension to “xml” would be saved.

By opening this file by browser, it is possible to confirm configuration information.

Note: This XML information is only Japanese in current version.



uC3-Configurator Project File

[PROJECT]

Project Filename	CPU ID	Config version
C:\Users\Urabe\Desktop\f\config1.3cf	303	2610

[Kernel Configuration]

Kernel Common Config

Additional Header File	IDLE function	Task Priority	Clock Tick	System Stack Size
		3	1	1024

TASK

Definition of ID	Function Name	Initial Priority	External Information	Task Stack Size	Attribution	Shared Stack
ID_MAIN_TSK	MainTsk	1		256	TA_HLNG TA_ACT	No
ID_Task1	Task1	1		256	TA_HLNG TA_ACT	No

Semaphore

Definition of ID	Initial Resources	Maximum Resource	Attribution

EventFlags

Definition of ID	Initial Bit Pattern(HEX value)	Attribution
ID_Flg1	0	TA_TFIFO TA_WSGL

Data Queue

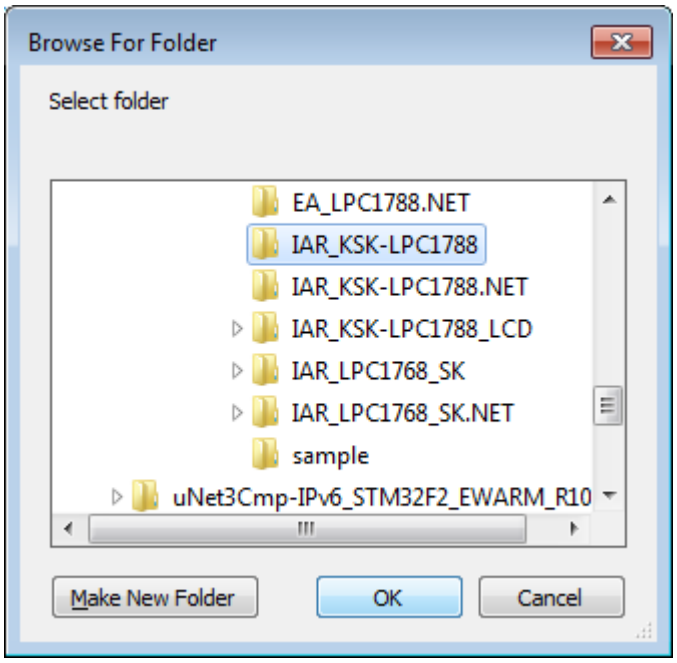
Definition of ID	Amount of Data	Attribution

MailBox

Definition of ID	Attribution
ID_Mbx1	TA_TFIFO TA_MFIFO

4. 1. 4 Generate source

From “File” → “Generate source…(G)” , open “screen of referring folder” , specify optional folder which deploy to create file and click “OK” .



In case there is already skeleton code main.c existing, confirming message will be displayed in other to prevent application file which has been finished editing from being overwritten and deleted.

【Recommendation】

In order to prevent skeleton code from being overwritten and deleted, it is recommended not to directly edit to skeleton code but using template to create application program.

A. Files which are not depended to a surely created processor

File	Content
kernel_id.h	Defined header file of Object ID or Device ID
kernel_cfg.c	Configuration information file of kernel
kernel.h	Header file of kernel
main.c	Skeleton code such as main(), initially set-up function, Task or Handler

B. Files which are depended to a surely created processor

File	Content
itron.h	Header file of kernel
Start-up	Initialization process by Power-on reset(Assembler language)
Vector table	Interrupt vector table(Assembler language)
Exceptional Handler	Exceptional Handler including interrupt Handler(Assembler language)
Kernel Library	Library summarizing basis part of kernel and system call group

C. Files depended on device driver

File	Content
I/O defined file	Header file defining I/O of processor
DDR_xxxxx.c	Source file of device driver
DDR_xxxxx.h	Header file of of device driver
DDR_xxxxx_cfg.h	Configuration file of of device driver

These created files are different according to configuration or processor or device.

4. 1. 5 Error check when creating source

When creating source, the following items will be checked. In case there is some problem, error message will be displayed and file will not be created.

- Check items which must not empty ID or function name.
- Check scope of total ID.
- Check scope of Task Priority Level.
- Check relation of Task Priority Level and Restriction Task attribute among Tasks which use Stack in common.
- Check scope of initial value of Semaphore.
- Check scope of start-up cycle of Cycle Handler.

4. 1. 5. 1 Total ID

All Object ID, including ID used in RTOS which user cannot see, will be managed by unique 8-bit value. Therefore, maximum of total ID will be 255, and number which can create Object will become less than 255.

Total ID is calculated like following formula:

	Upper limit of Task Priority Level
	Number of Shared Stack
	Number of Task
	Number of Semaphore
	Number of Eventflag
	Number of Mailbox
	Double number of Data Queues
	Number of Fixed-Sized memory pool
＋)	Number of Cycle Handler
	Total ID

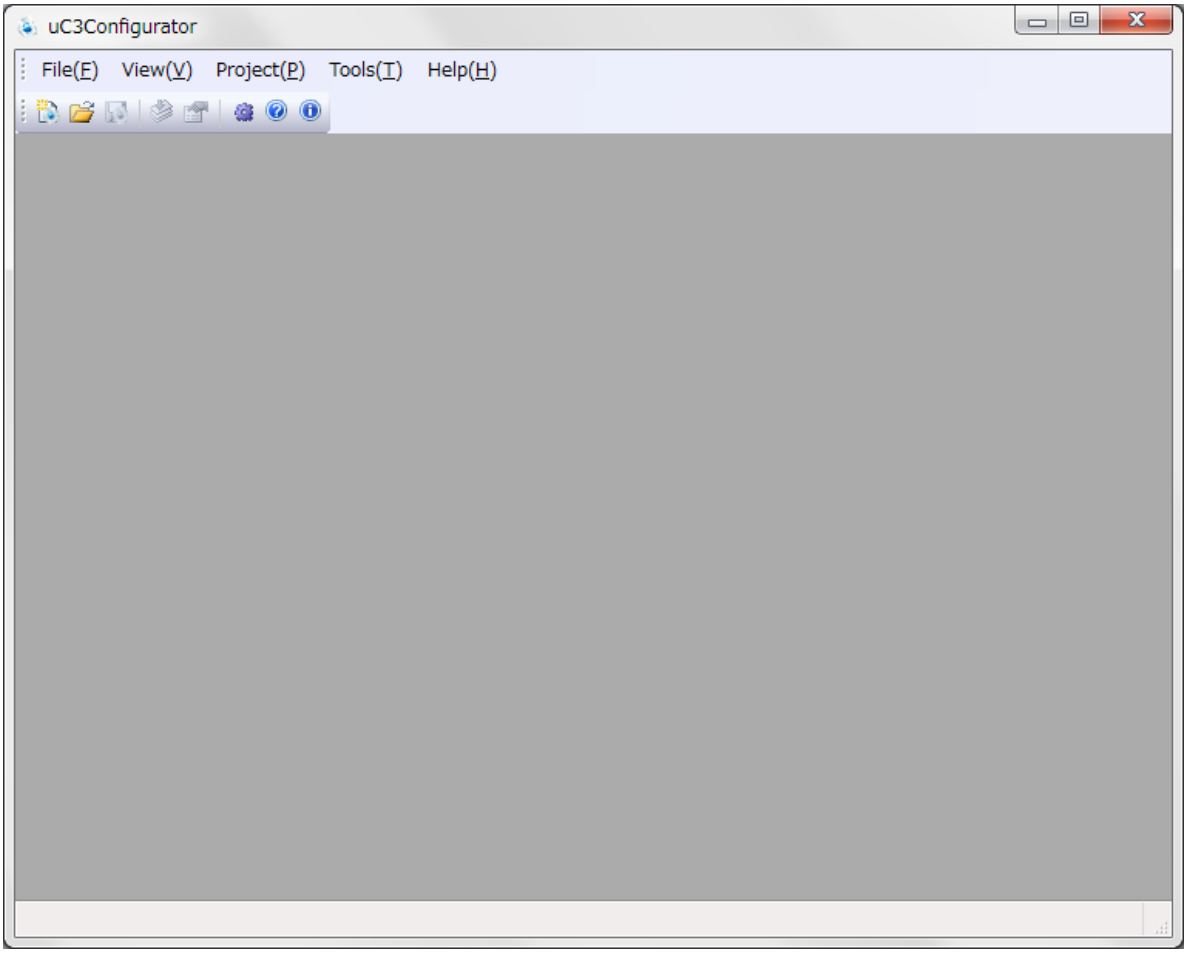
【Complement】

In the evaluation edition of µC3/Compact, total ID is limited to 16.

4. 2 Operation of the configurator : Current Version

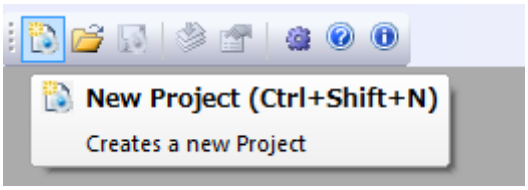
4. 2. 1 Starting up Configurator

Please double click “μC3conf.exe” to start up.



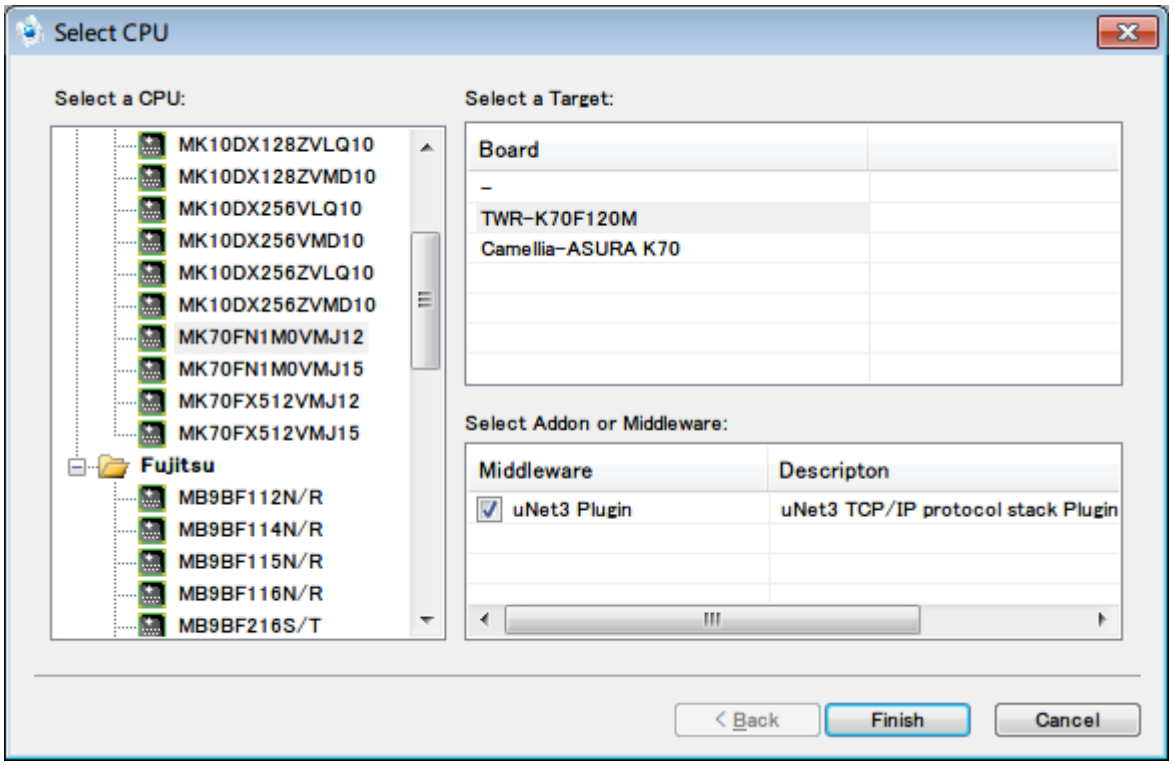
A. In case of creating a new project

From the Configurator toolbar, click “New Project” and go to “Select CPU” .



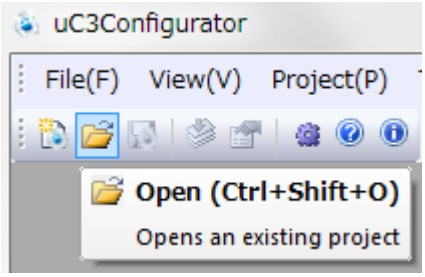
Select CPU

After selecting CPU vendors, CPU, serial number, target in List, click “OK” and go to “Main screen” .



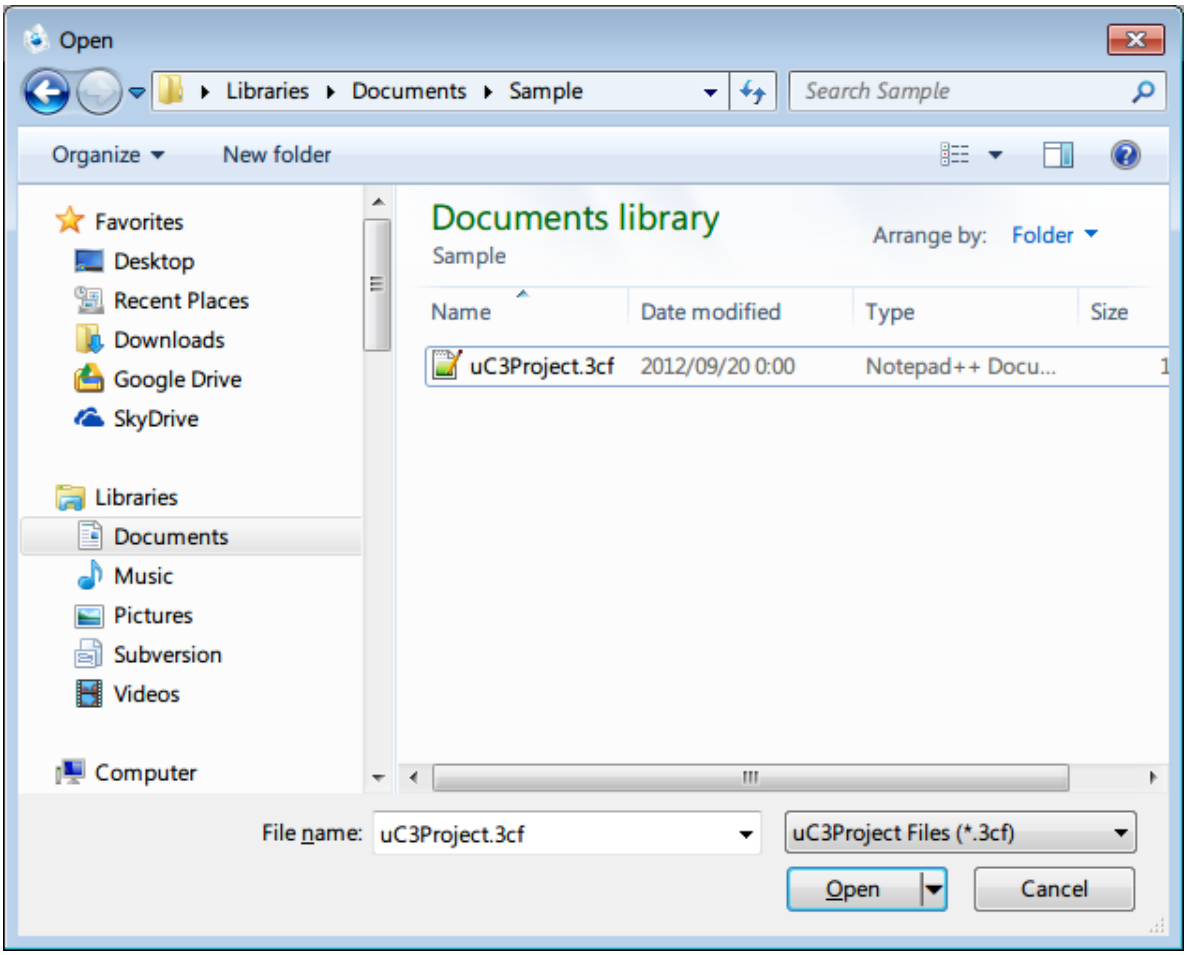
B. In case of opening an existing project

From the Configurator toolbar, click “Open” and go to “Open file” .



Open file

After selecting a saved project file(extension.3cf), click “Open” and go to “Main screen” .

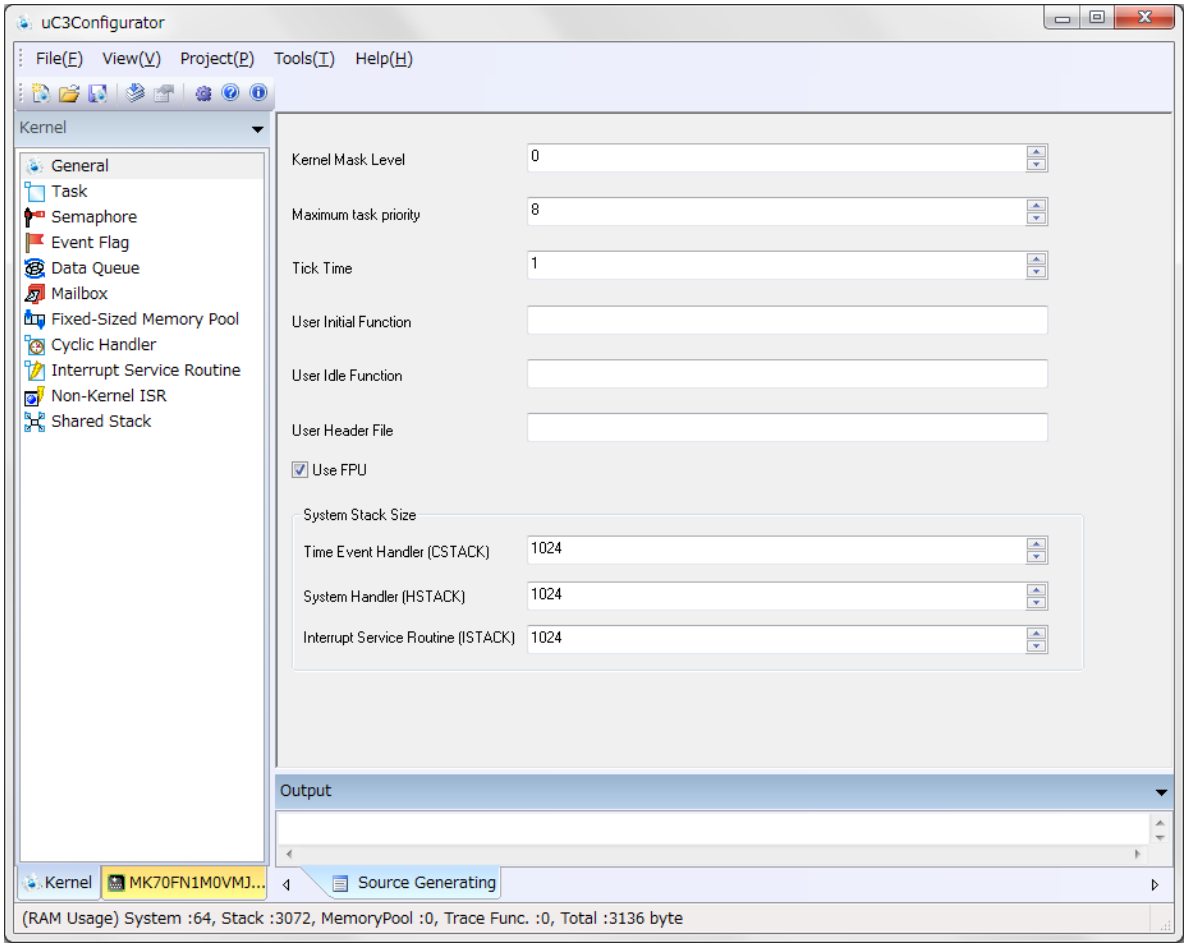


*:Project file created by "Ver.2.x configurator" is read only kernel configuration.
(CPU configuration is not read)

C. Main screen

After starting up, it will go to the main screen where it is possible to refer or edit project. There is a menu screen to the left of the main screen. By clicking to each Object of Menu Screen, it will switch to each Object Configutation screen.

Here, there is configuration of kernel or processor dependence part. Please refer to “Processor dependence part Manual”, “Device dependence part Manual” for more explanation.

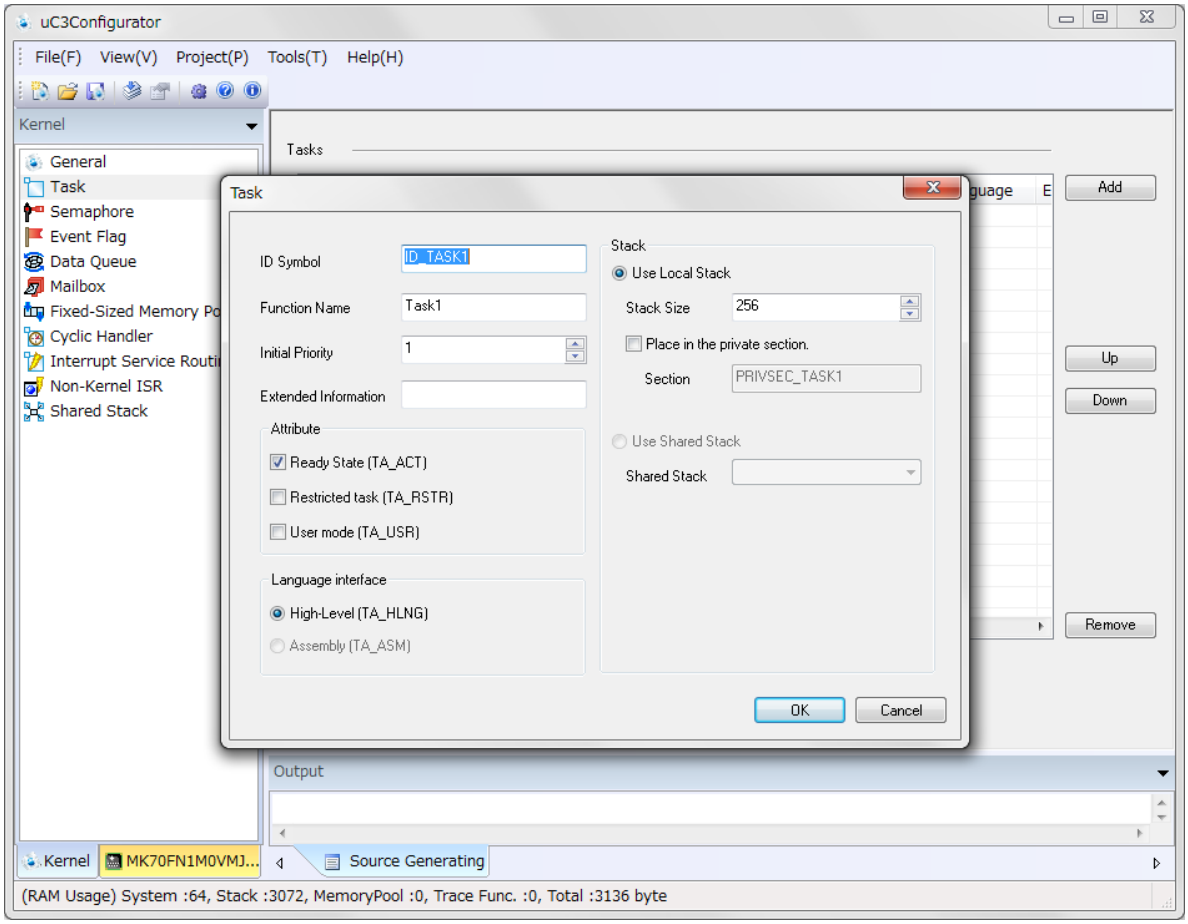


*:Due to limitations of space, the description has been described with separate configuration screen and menu screen.

4. 2. 2 Set-up kernel

In kernel configuration, there are configurations of common kernel and Objects such as Task, Semaphore etc.… In configuration screen of each Object type, 1 Object is corresponding to 1 line of the list.

The following figure is an example of Configuration screen of an Object.



“Add” button

Dialog for adding a new object appears. The object is added by setting the required items. In addition, you can double-click the list in Configuration screen, it is possible to update the contents of the object can be achieved.

“Delete” button

To delete Object of the list in Configuration screen which is selected at the moment. There is an object can not be deleted. This kind of Object is Object which is added and synchronized to configuration of device driver.

“Up” button

Move to the up of a list item which is selected at the moment.

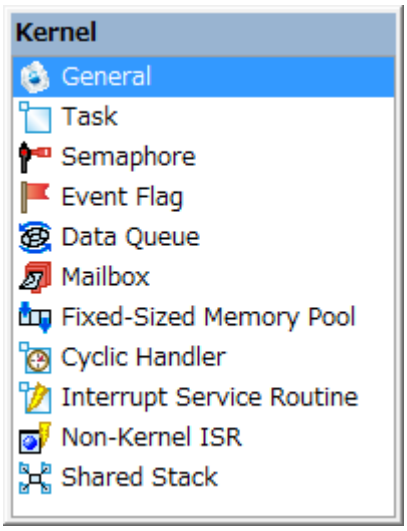
“Down” button

Move to the down of a list item which is selected at the moment.

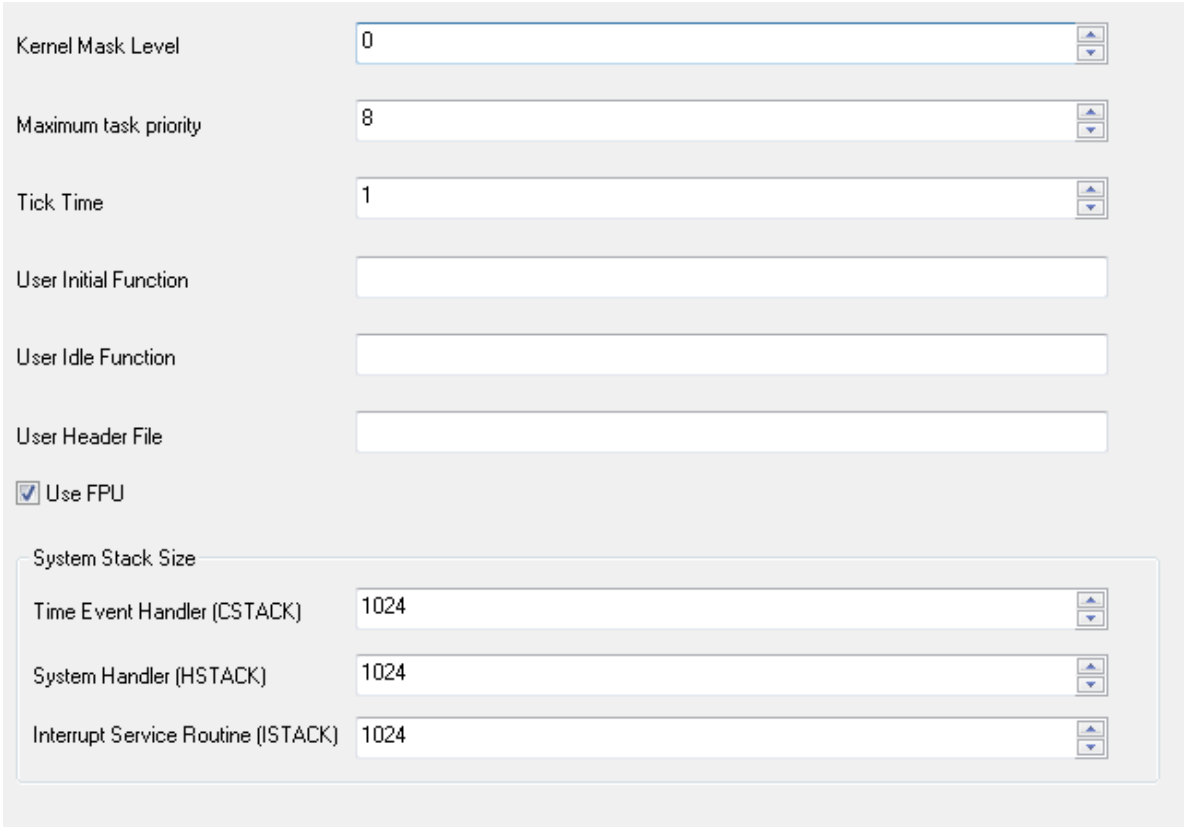
4. 2. 2. 1 Configuration of general kernel

When clicking to “General” of Menu Screen, configuration screen of general kernel will be displayed to execute configuration for general kernel.

Menu Screen



Configuration Screen



Kernel Mask Level

Set the Kernel Mask Level. Please refer to "Processor dependence part Manual" for more explanation.

Maximum task priority

It is possible to specify from 1 to 16, and Task Priority Level which is upper to this value.

Tick Time

Cycle of Time Tick is specified in mili-second unit. The less the value is, the more accuracy the time is, but the overhead will be bigger.

User Initial function

Specify that function name when need the application initialization process.

User Idle function

Specify that function name when not using standard Idle function of internal kernel but replacing it with Idol function of user definition.

User header file

When defining pointer to value of macro or variable as extended information of Task and Cycle Handler, file name of header file which describes external declaration of macro definition or variable will be specified. In detail, when a file name is specified here, that file will be included in kernel_cfg.c.

Use FPU *

Choose to use the Floating Point Unit.

Time Event Handler (CSTACK)

Size of Stack area used by Idle and Cyclic Handler is specified in byte unit.

System Handler (HSTACK)

Size of Stack area used by Interrupt and Interrupt Service Routine is specified in byte unit.

Interrupt Service Routine (ISTACK) *

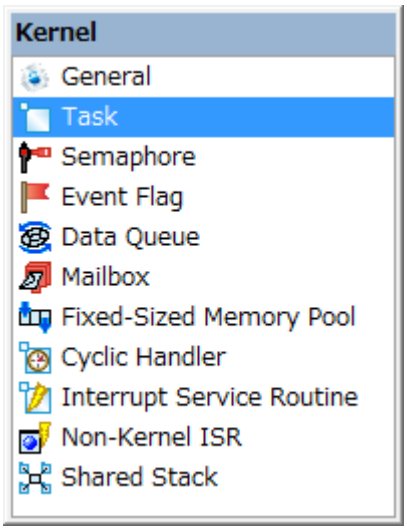
Size of Stack area used by Interrupt Service Routine is specified in byte unit.

***:Is not displayed when the device is not supported.Please refer to "Device dependence part Manual" for more explanation.**

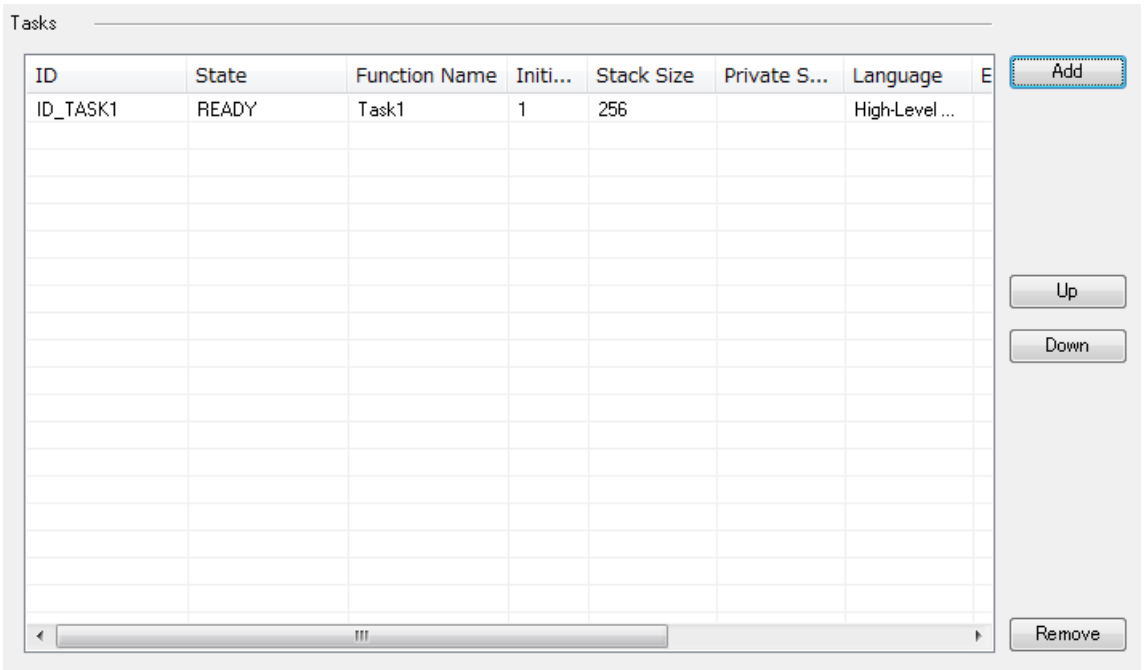
4. 2. 2. 2 Configuration of Task

When clicking to “Task” of Menu Screen, configuration screen of Task will be displayed, configuration corresponding to CRE_TSK of Task creating API will be executed.

Menu Screen



Configuration Screen



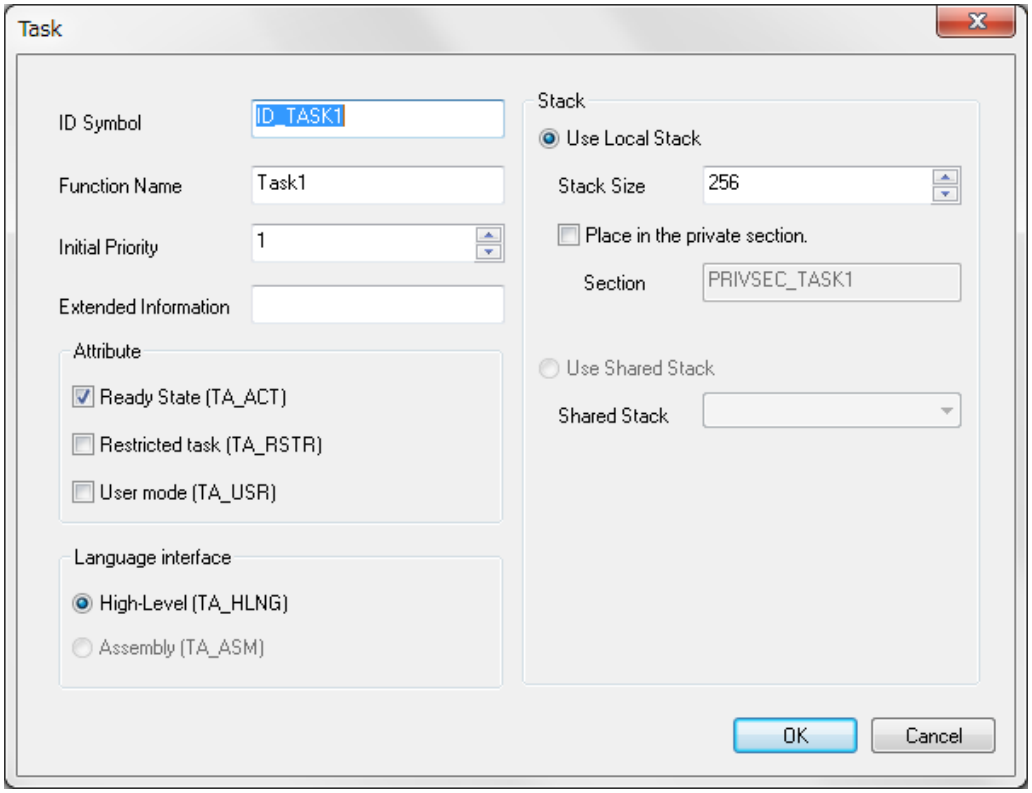
Tasks

A list of tasks that are currently set will be displayed. Editing screen is displayed by double-clicking an item in the list.

Add

Screen to add a new task is displayed.

Task Set Screen



ID Symbol

Please specify optional definition name displaying ID number of Task. This definition name is macro-defined in kernel_id.h.

Function name

Specify function name of option Task.

Initial Priority

When starting up Task, please specify value of initializing Task Priority Level which is not exceeding Task Priority Level number of common kernel. When specifying shared stack and attribute of Restriction Task(TA_RSTR=ON), the same task priority as other tasks which specified the shared stack will be specified.

Extended information

In case there is extension information which is passing to Task, then specify it, and leave it it blank if it is unnecessary. In extension information, it is possible to specify numerical value, value which is macro-defined and pointer to variable. In case of passing pointer to variable, attach “&” to the beginning of variable name.

Ready State (TA_ACT)

When checking, TA_ACT attribute will become ON, and Task is created in possible execution status.

Restricted task (TA_RSTR)

When checking, TA_RSTR attribute will become ON, and attribute of Restriction Task is given. When selecting the shared stack which was described later, it will be automatically checked, but it is possible to remove the check later. In case shared stack is used by various Tasks, all those Tasks must be either TA_RSTR=ON, or TA_RSTR=OFF.

User mode (TA_USR) *

This function is device dependent.

High-Level (TA_HLNG) / Assembly (TA_ASM)

It will become fixed TA_HLNG, and impossible to change in μC3/Compact.

Stack size

Please specify size of peculiar stack to Task. Select "Use the local stack", will be enabled for this control.

Use Local Stack / Use Shared stack

Specify whether to use local stack or shared stack. Regarding to field of shared stack, if there are more than 1 shared stack defined, it will be possible to select “Use Shared stack”.

Shared Stack

Specify ID of shared stack. Select "Use the local stack", will be enabled for this control.

Place in the private section *

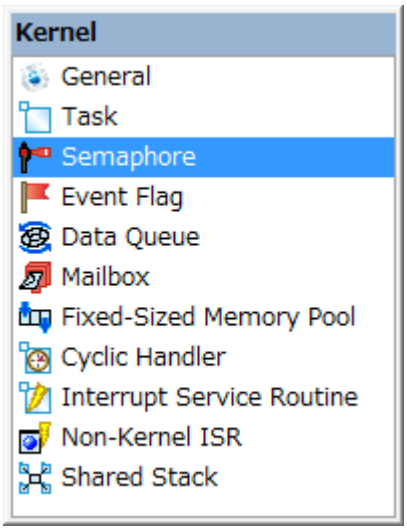
This function is device dependent.

***:Is not displayed when the device is not supported.Please refer to “Device dependence part Manual” for more explanation.**

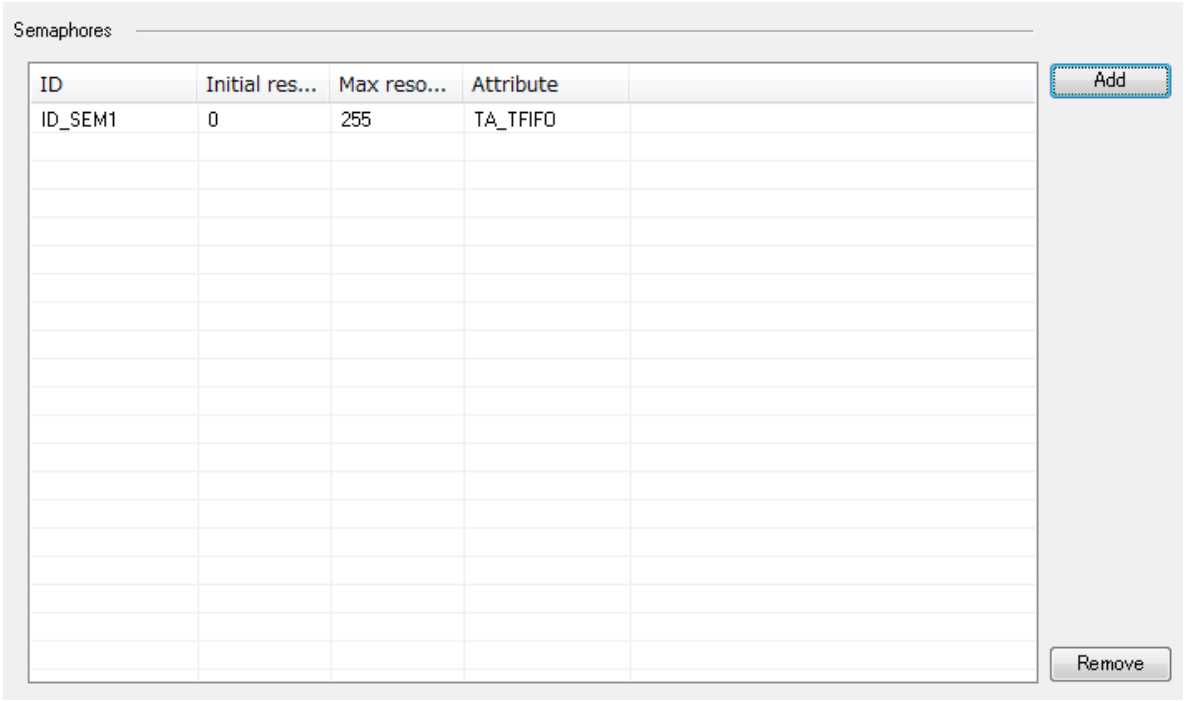
4. 2. 2. 3 Configuration of Semaphore

When clicking to “Semaphore” of Menu Screen, configuration screen of Semaphore will be displayed for configuration which is corresponding to CRE_SEM of Semaphore creating API.

Menu Screen



Configuration Screen



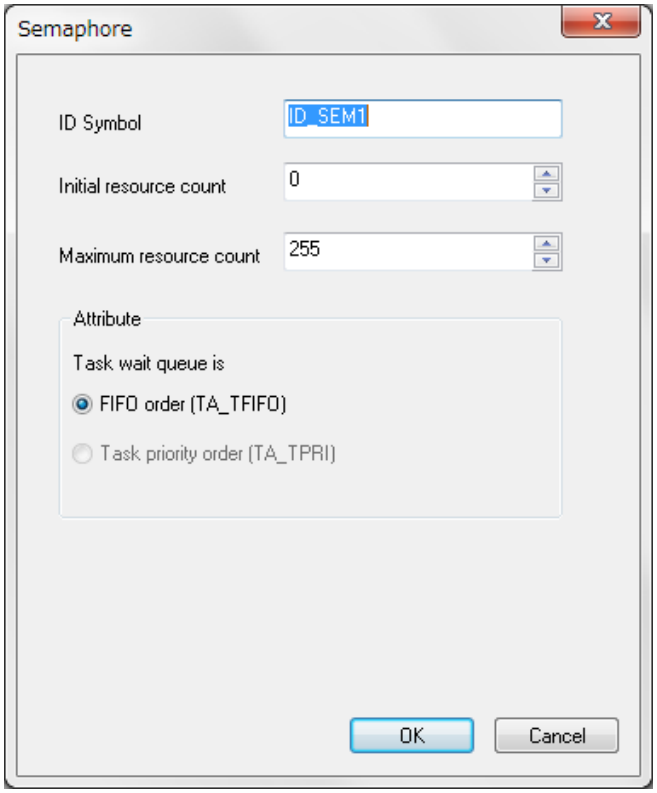
Semaphores

A list of Semaphores that are currently set will be displayed. Editing screen is displayed by double-clicking an item in the list.

Add

Screen to add a new Semaphore is displayed.

Semaphore Set Screen



ID Symbol

Please specify optional definition name which displays ID number of Semaphore. This definition name is macro-defined in kernel_id.h.

Initial resource count

Specify initial value of Semaphore count which is not exceeding maximum resource number.

Maximum resource number

Please specify maximum value of Semaphore count. The maximum value which can be specified is 255.

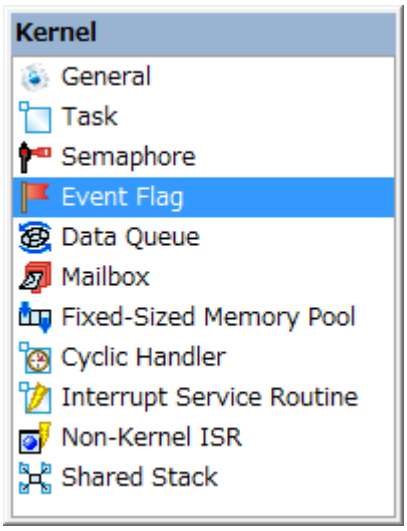
FIFO order (TA_TFIFO) / Task priority order (TA_TPRI)

It will become fixed TA_TFIFO and impossible to change in μC3/Compact.

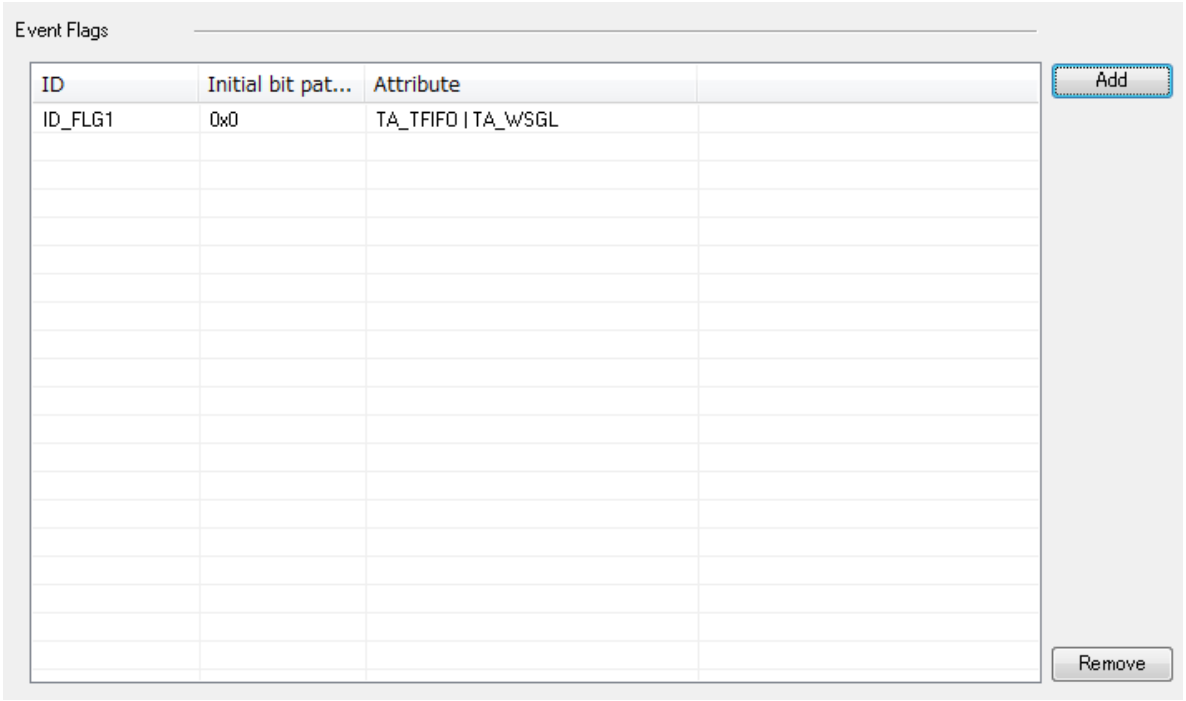
4. 2. 2. 4 Configuration of Event Flag

When clicking to “Event Flag” of Menu Screen, configuration screen of Event Flag will be displayed for configuration which is corresponding to CRE_FLG of Event Flag creating API.

Menu Screen



Configuration Screen



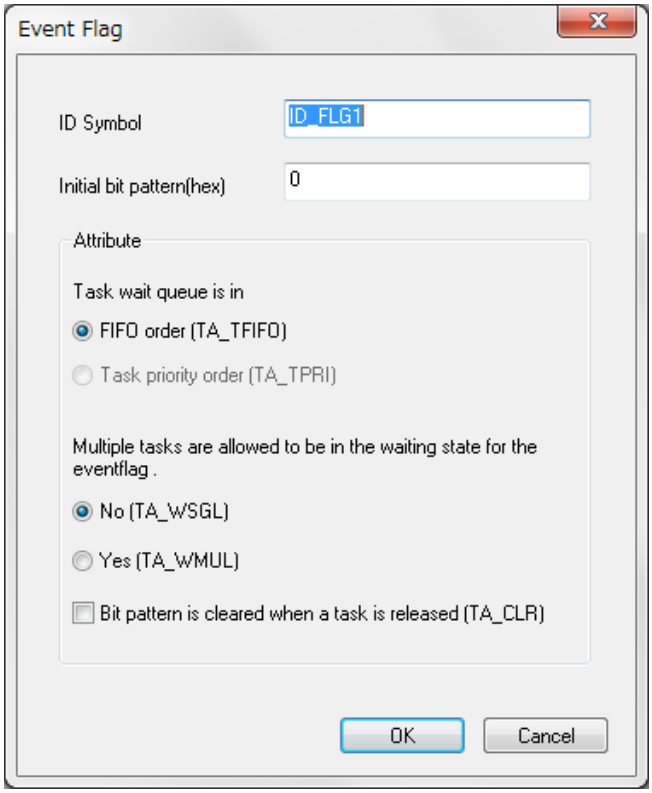
Event Flags

A list of Event Flags that are currently set will be displayed. Editing screen is displayed by double-clicking an item in the list.

Add

Screen to add a new Event Flag is displayed.

Event Flag Set Screen



ID Symbol

Please specify optional definition name which displays ID number of Event Flag. This definition name is macro-defined in kernel_id.h.

Initial bit pattern (hex)

Please specify initial value of Event Flag by hexadecimal number.

FIFO Order (TA_TFIFO) / Task priority order (TA_TPRI)

It will become fixed TA_TFIFO and impossible to change in μC3/Compact.

No (TA_WSGL) / Yes (TA_WMUL)

By specifying TA_WSGL, waiting of various Tasks will be prohibited. By specifying TA_WMUL, waiting of various Tasks will be permitted.

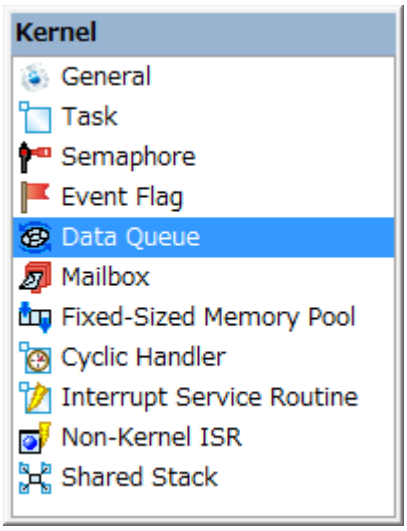
Bit pattern is cleared when a task is released (TA_CLR)

If checking, attribute of TA_CLR will be ON, and when Task is released from Event Flag waiting by a condition approval, all bits of bit pattern are cleared.

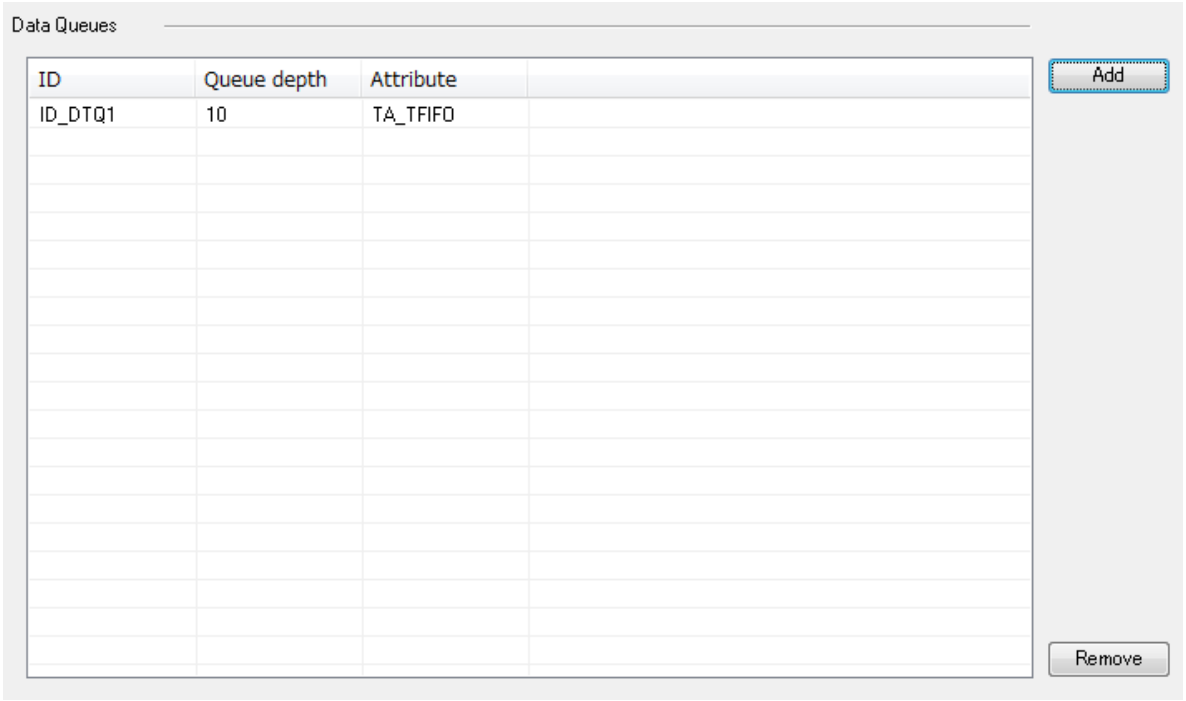
4. 2. 2. 5 Configuration of Data Queues

When clicking to “Data Queue” of Menu Screen, configuration screen of Data Queues will be displayed for configuration which is corresponding to CRE_DTQ of Data Queues creating API.

Menu Screen



Configuration Screen



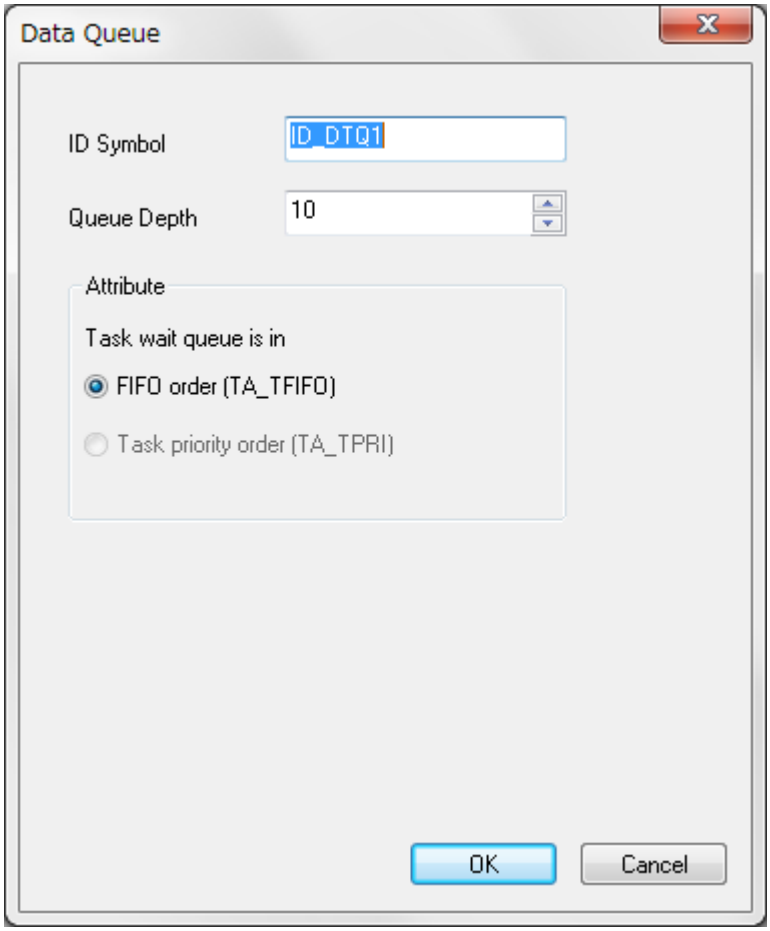
Event Flags

A list of Data Queues that are currently set will be displayed. Editing screen is displayed by double-clicking an item in the list.

Add

Screen to add a new Data Queue is displayed.

Data Queue Set Screen



ID Symbol

Please specify optional definition name which displays ID number of Data Queue. This definition name is macro-defined in kernel_id.h.

Queue Depth

Spacify number of Data Queues (number of data).

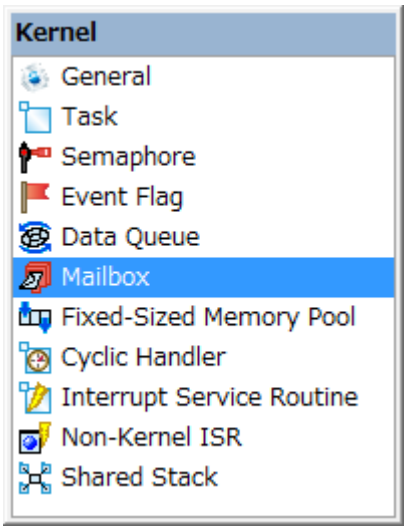
FIFO Order (TA_TFIFO) / Task priority order (TA_TPRI)

It will become fixed TA_TFIFO, and impossible to change in μC3/Compact.

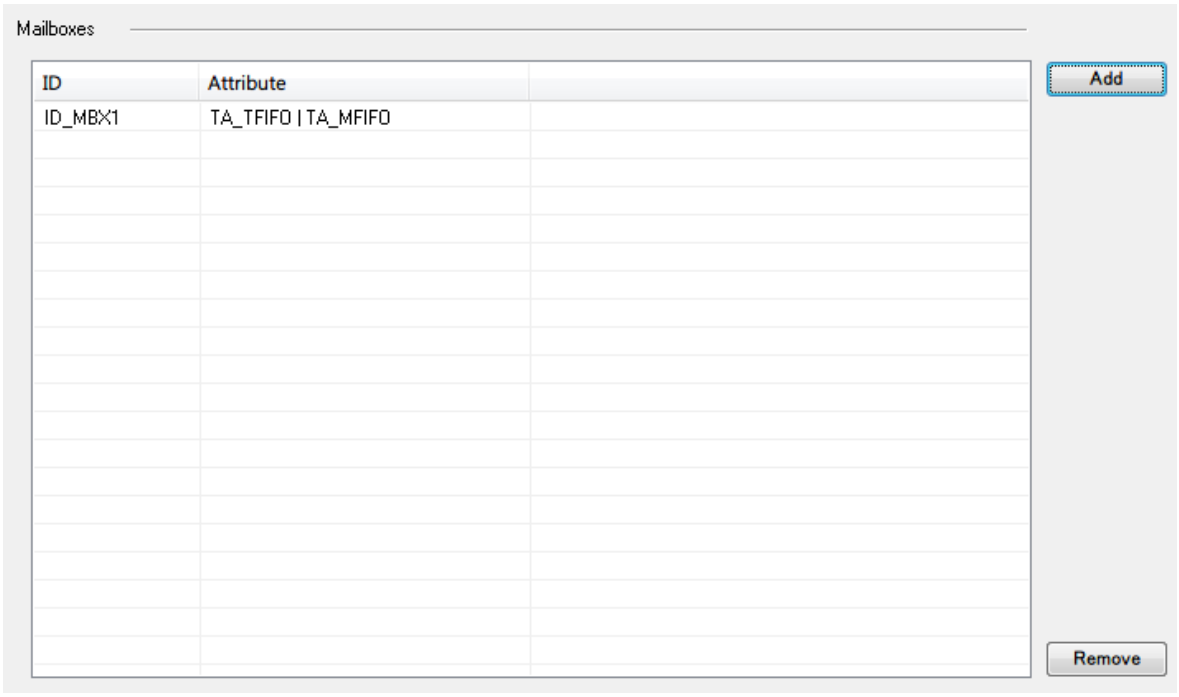
4. 1. 2. 6 Configuration of Mailbox

When clicking to “Mailbox” of Menu Screen, configuration screen of Mailbox will be displayed for configuration which is corresponding to CRE_MBX of Mailbox creating API.

Menu Screen



Configuration Screen



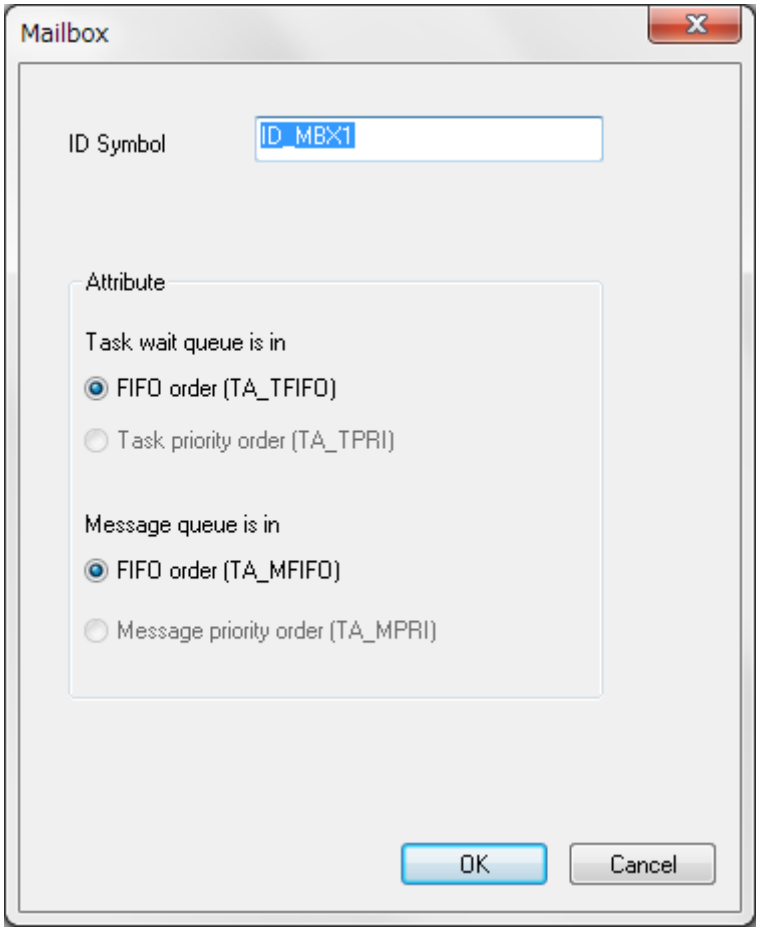
Mailboxes

A list of Mailboxes that are currently set will be displayed. Editing screen is displayed by double-clicking an item in the list.

Add

Screen to add a new Mailbox is displayed.

Mailbox Set Screen



ID Symbol

Please specify optional definition name which displays ID number of Mailbox. This definition name is macro-defined in kernel_id.h.

FIFO order (TA_TFIFO) / Task priority order (TA_TPRI)

It will become fixed TA_TFIFO and impossible to change in μC3/Compact.

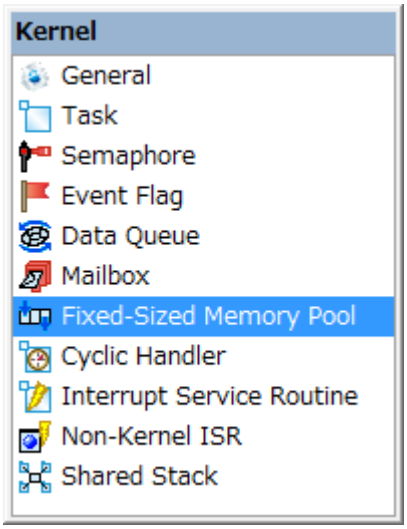
FIFO order (TA_MFIFO) / Message priority order (TA_MPRI)

It will become fixed TA_MFIFO and impossible to change in μC3/Compact.

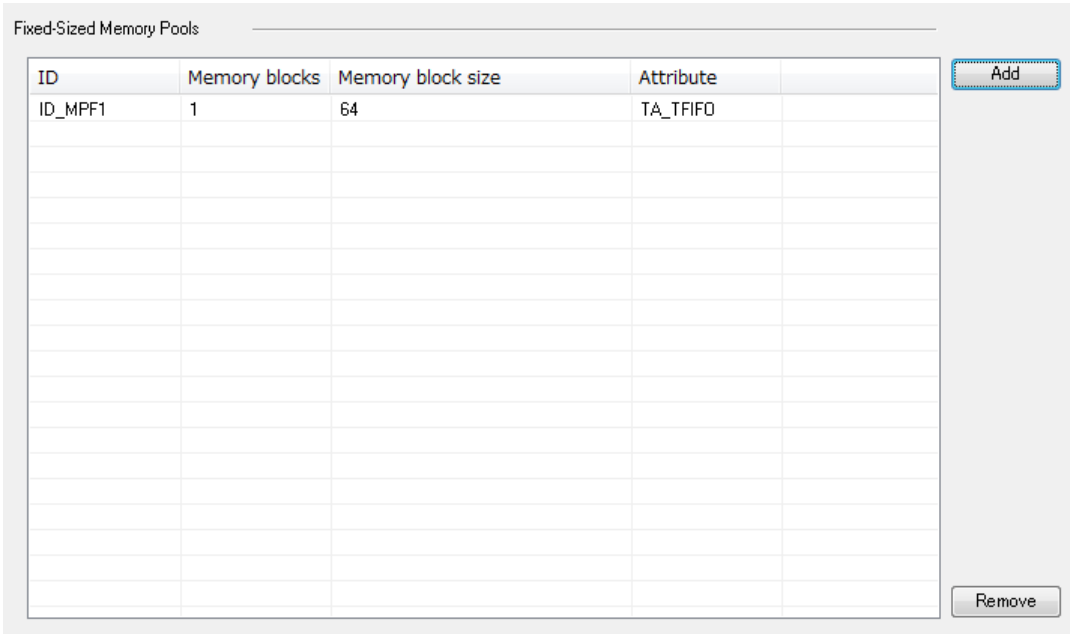
4. 2. 2. 7 Configuration of Fixed-Sized Memory Pool

When clicking to “Fixed-Sized Memory Pool” of Menu Screen, configuration screen of Fixed-Sized Memory Pool will be displayed for configuration which is corresponding to CRE_MPF of Fixed-Sized Memory Pool creating API.

Menu Screen



Configuration Screen



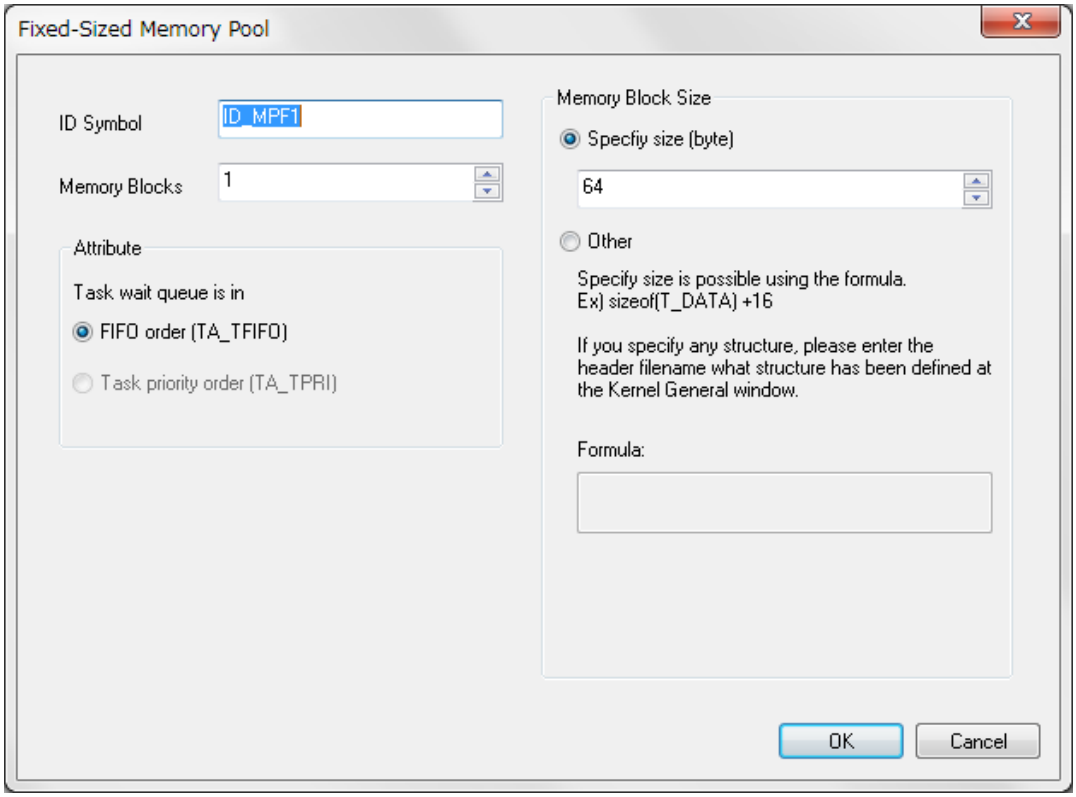
Fixed-Sized Memory Pools

A list of Fixed-Sized Memory Pools that are currently set will be displayed. Editing screen is displayed by double-clicking an item in the list.

Add

Screen to add a new Fixed-Sized Memory Pool is displayed.

Fixed-Sized Memory Pool Set Screen



ID Symbol

Please specify optional definition name which displays ID number of Fixed-Sized Memory Pool. This definition name is macro-defined in kernel_id.h.

Memory Blocks

Specify number of memory block.

FIFO order (TA_TFIFO) / Task priority order (TA_TPRI)

It will become fixed TA_TFIFO and impossible to change in μC3/Compact.

Specify size (byte)

Specify size of memory block (number of byte).

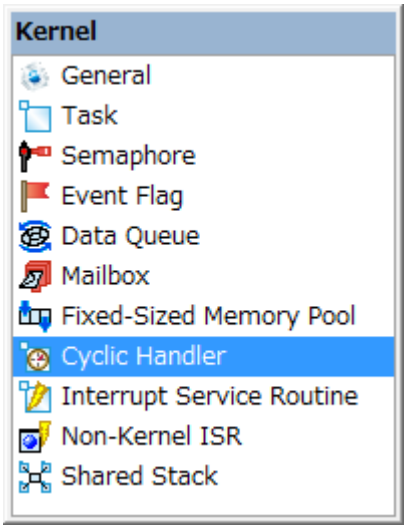
Other

Specify size of memory block (number of byte) using operator sizeof, and arithmetic operators. In this case, memory usage is approximate. In addition, it is necessary to specify the header file structure is defined in the configuration "General" to specify a user-defined structure.

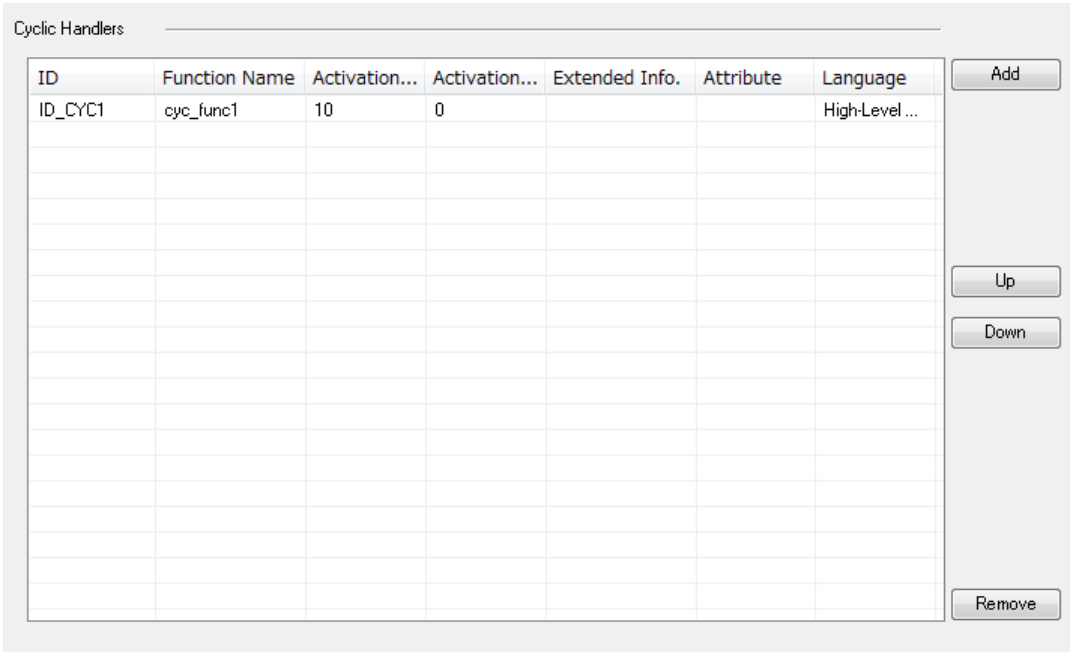
4. 2. 2. 8 Configuration of Cycle Handler

When clicking to “Cyclic Handler” of Menu Screen, configuration screen of Cycle Handler will be displayed for configuration which is corresponding to CRE_CYC of Cycle Handler creating API.

Menu Screen



Configuration Screen



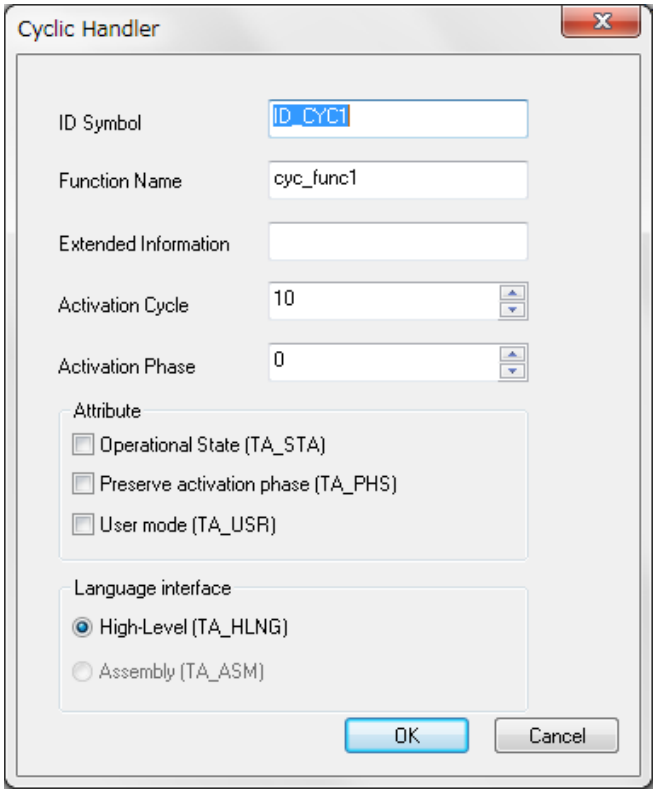
Cyclic Handlers

A list of Cyclic Handlers that are currently set will be displayed. Editing screen is displayed by double-clicking an item in the list.

Add

Screen to add a new Cyclic Handler is displayed.

Cyclic Handler Set Screen



ID Symbol

Please specify optional definition name which displays ID number of Cycle Handler. This definition name is macro-defined in kernel_id.h.

Function name

Specify function name of optional Cycle Handler.

Extended information

If there is extension information which is passing to Cycle Handler, specify it, or in case of unnecessary, just leave it in blank. In extension information, it is possible to specify numerical value, macro-defined value, pointer to variable. If passing pointer to variable, attach “&” to the beginning of variable name.

Activation Cycle

Starting-up cycle of Cycle Handler is specified by mili-second unit. However, small value cannot be specified by Tick time.

Activation Phase

Starting-up phase of Cycle Handler is specified by mili-second unit.

High-Level (TA_HLNG) / Assembly (TA_ASM)

It is impossible to change in µC3/Compact.

Operational State (TA_STA)

If checking and attribute of TA_STA is ON, Cycle Handler is generated by operation status.

Preserve activation phare (TA_PHS)

If checking and attribute of TA_PHS is ON, phase when generating Cycle Handler is saved.

User mode (TA_USR) *

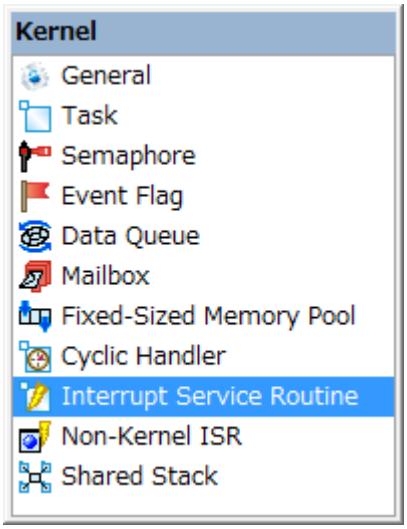
This function is device dependent.

***:Is not displayed when the device is not supported.Please refer to "Device dependence part Manual" for more explanation.**

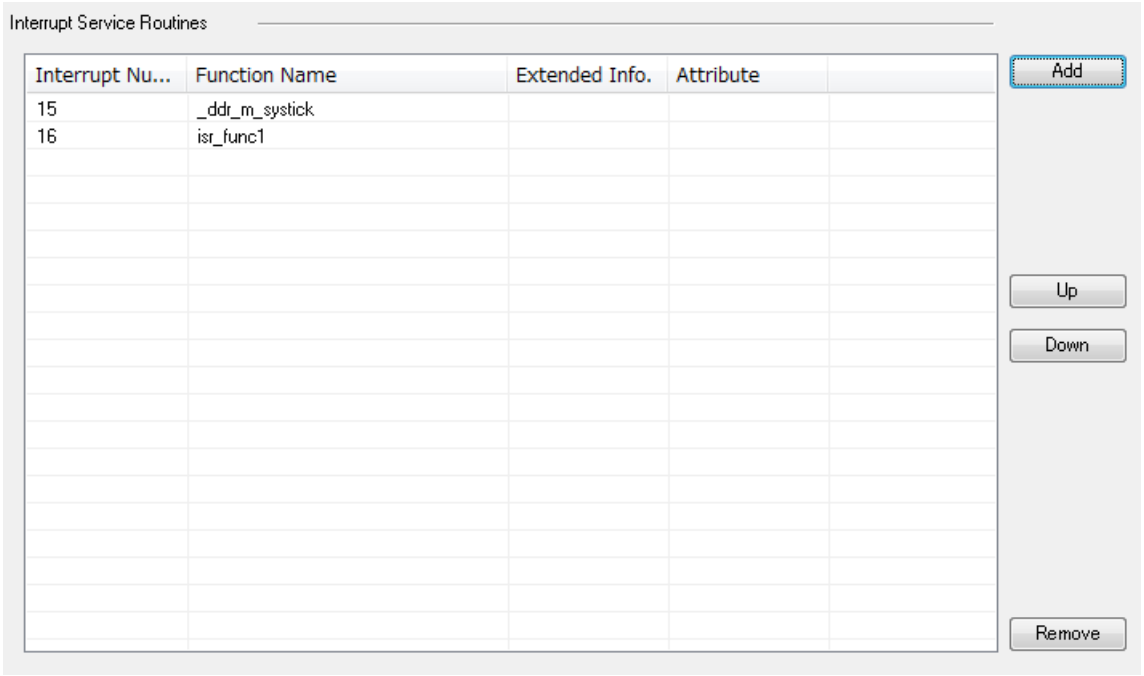
4. 2. 2. 9 Configuration of Interrupt Service Routine

When clicking to “Interrupt Service Routine” of Menu Screen, configuration screen of Interrupt Service Routine will be displayed for configuration which is corresponding to ATT_ISR of added API of Interrupt Service Routine.

Menu Screen



Configuration Screen



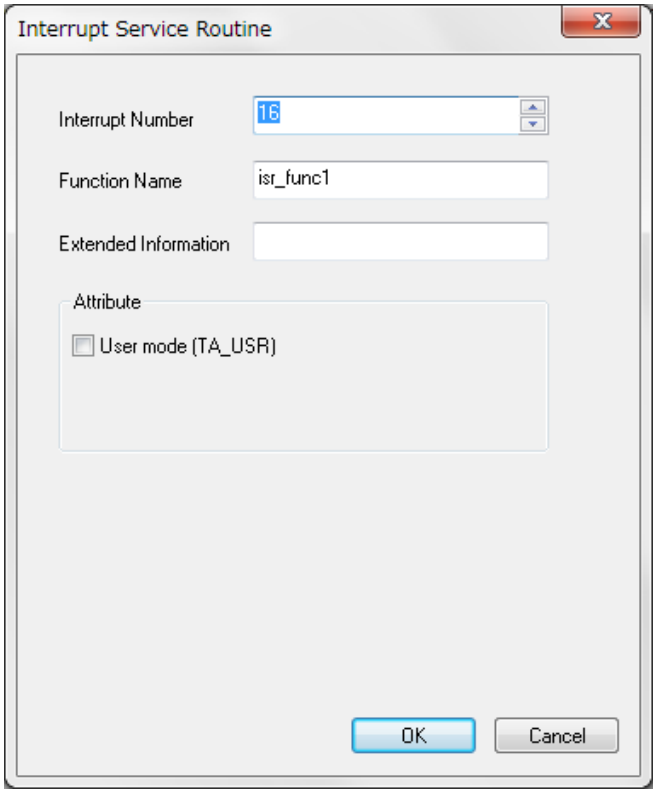
Interrupt Service Routines

A list of Interrupt Service Routines that are currently set will be displayed. Editing screen is displayed by double-clicking an item in the list.

Add

Screen to add a new Interrupt Service Routine is displayed.

Interrupt Service Routine Set Screen



Interrupt number

Please specify interrupt number. When configurating various Interrupt Service Routine to the same interrupt number, calling order will follow order of tab and the more it is on the left, the faster it will be called.

Function name

Specify function name of optional Interrupt Service Routine.

Extended Information

If there is extension information which is passing to Interrupt Service Routine, specify it, or in case of unnecessary, just leave it in blank. In extension information, it is possible to specify numerical value, pointer to variable.

User mode (TA_USR) *

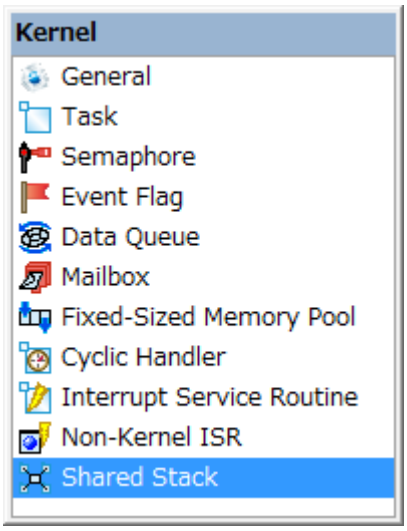
This function is device dependent.

***:Is not displayed when the device is not supported.Please refer to "Device dependence part Manual" for more explanation.**

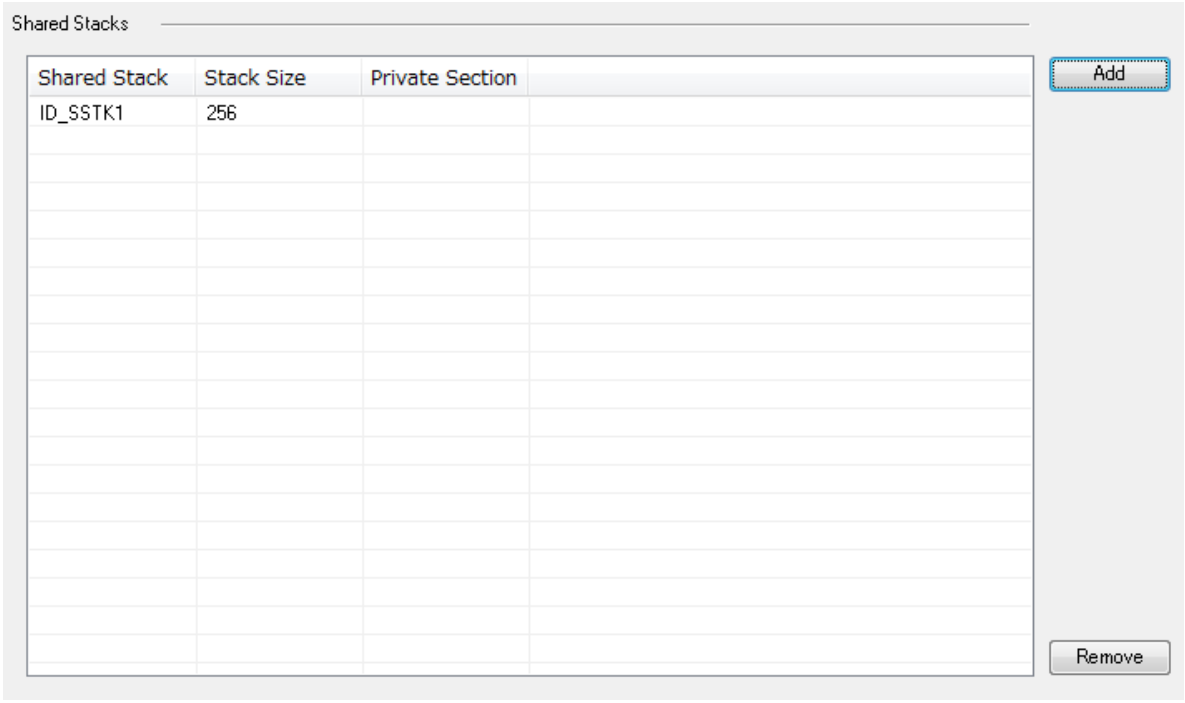
4. 2. 2. 10 Configuration of Shared stack

When clicking to “Shared Stack” of Menu Screen, configuration screen of Shared Stack will be displayed for configuration of Shared Stack.

Menu Screen



Configuration Screen



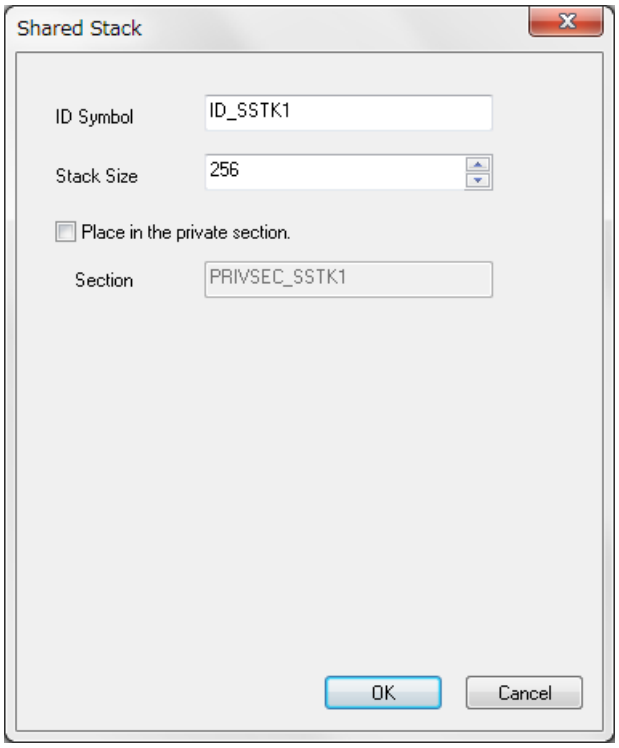
Shared Stacks

A list of Shared Stacks that are currently set will be displayed. Editing screen is displayed by double-clicking an item in the list.

Add

Screen to add a new Shared Stack is displayed.

Shared Stack Set Screen



ID Symbol

Please specify optional definition name which displays ID number of Shared Stack. This definition name is used to select Shared Stack in configuration screen of Task.
In case there is even 1 Task using this Shared Stack, it will be impossible to change definition name.

Stack Size

Specify size of Shared Stack (byte number). The Stack size of Task selecting the use of Shared Stack is fixed to size of Shared Stack. Therefore, Stack size of Task which uses the most Stack is specified by the Task specifying this Shared Stack.

Deletion

In case there is even 1 Task using that Shared Stack, display warning message and it will be deleted. In that case, Shared Stack of the Task is changed to “Not use” .

Place in the private section *

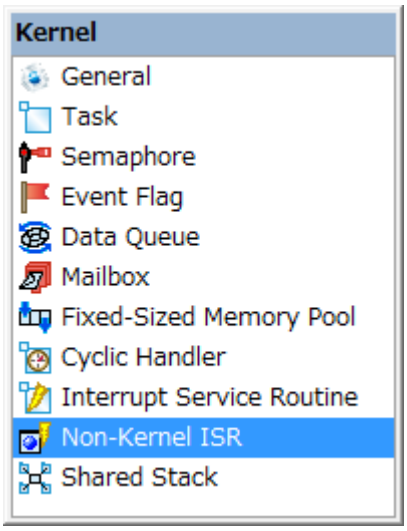
This function is device dependent.

***:Is not displayed when the device is not supported.Please refer to “Device dependence part Manual” for more explanation.**

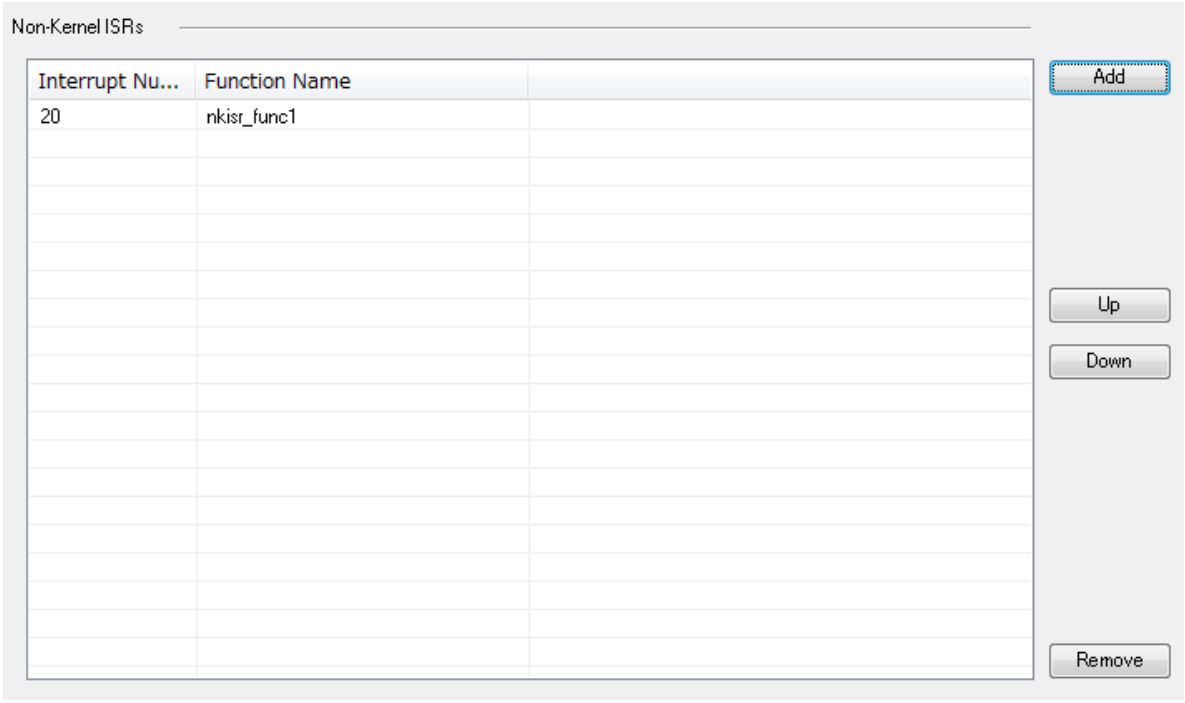
4. 2. 2. 11 Configuration of Non-Kernel ISR (Interrupt Service Routine)

When clicking to “Non-Kernel ISR” of Menu Screen, configuration screen of Non-Kernel ISR will be displayed for configuration of Non-Kernel ISR.

Menu Screen



Configuration Screen



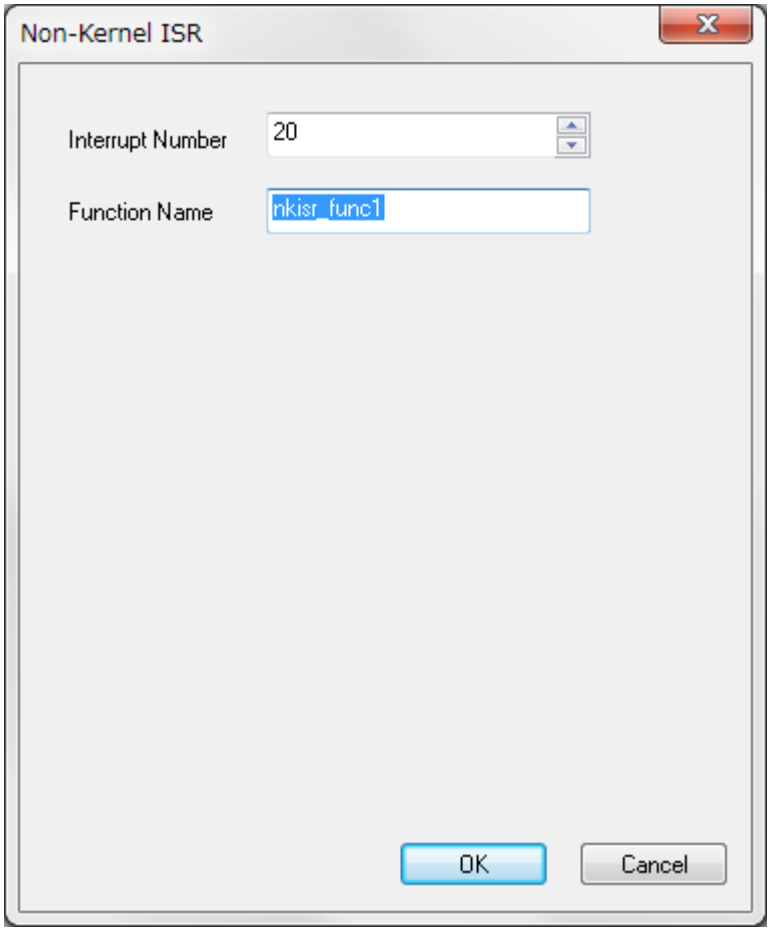
Non-Kernel ISRs

A list of Non-Kernel ISRs that are currently set will be displayed. Editing screen is displayed by double-clicking an item in the list.

Add

Screen to add a new Non-Kernel ISR is displayed.

Non-Kernel ISR Set Screen



Interrupt number

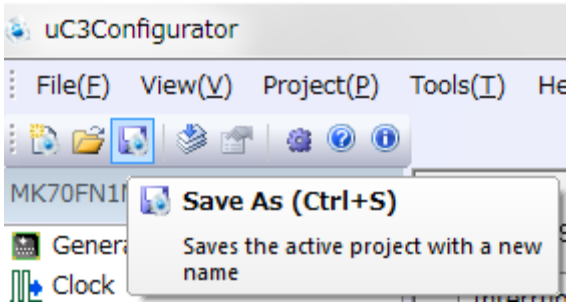
Please specify interrupt number. Must be interrupt number of Non-Kernel ISR is unique. Can not register it must be unique interrupt number, including the Non-Kernel ISRs and Interrupt Service Routines.

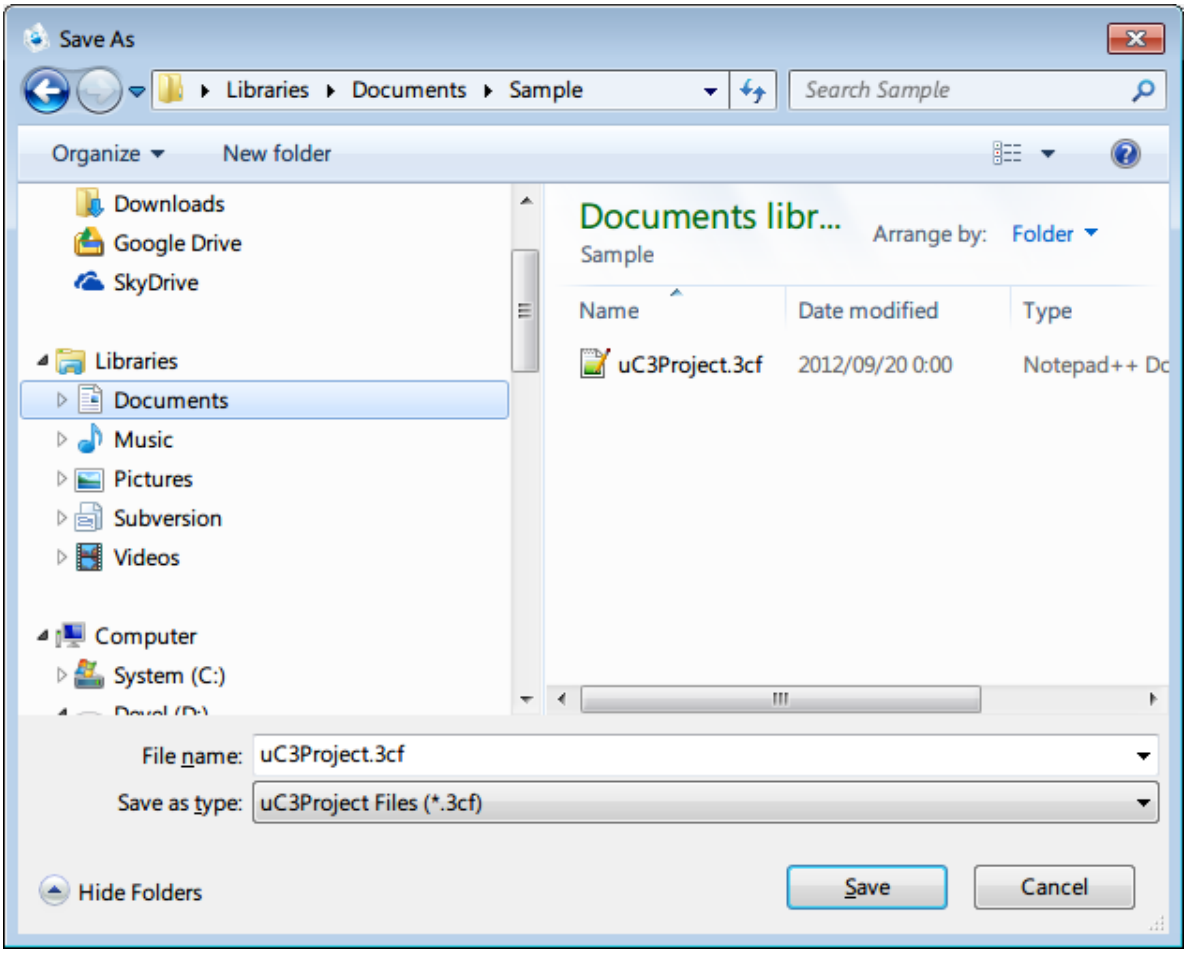
Function name

Specify function name of optional Non-Kernel ISR.

4. 2. 3 Saving project file

From the Configurator toolbar, click “Save As”, open “name and save screen” specify saving folder for project file and click “OK” .





Regarding to the saved file, the file that changed project file (default uC3Project.3cf)and extension to “xml” would be saved.

By opening this file by browser, it is possible to confirm configuration information.

uC3/Configurator Configuration List

[Using Plugins]

File Name
C:\Users\makino\Downloads\uNet3\uNet3_Kinetis_MDK-ARM_R200\uC3\Configurator\Compact\Kernel\ARM\Kinetis\uC3CmpKnlKinetis.plugin
C:\Users\makino\Downloads\uNet3\uNet3_Kinetis_MDK-ARM_R200\uC3\Configurator\Compact\CPU\Freescale\Kinetis\K\uC3CmpCpuKinetisK.plugin

---

[Kernel Configuration]

Kernel General

Kernel Mask Level	Maximum task priority	Tick Time	User Initial Function	User Idle Function	User Header File	Time Event Handler (CSTACK)	System Handler (HSTACK)	Interrupt Service Routine (ISTACK)	FPU
0	8	1				1024	1024	1024	Use

Task

ID Symbol	Function Name	Initial Priority	Extended Information	Stack Size	Attributes	Shared Stack	Use Private Section	Private Section Name
ID_TASK1	Task1	1		256	TA_HLNG | TA_ACT		No	PRIVSEC_TASK1
ID_TASK2	Task2	1		256	TA_HLNG | TA_ACT | TA_RSTR		No	PRIVSEC_TASK2

Semephore

ID Symbol	Initial resource count	Maximum resource count	Attributes
ID_SEM1	0	255	TA_TFIFO

EventFlag

ID Symbol	Initial bit pattern(hex)	Attributes
ID_FLG1	0x0	TA_TFIFO | TA_WSGL

DataQueue

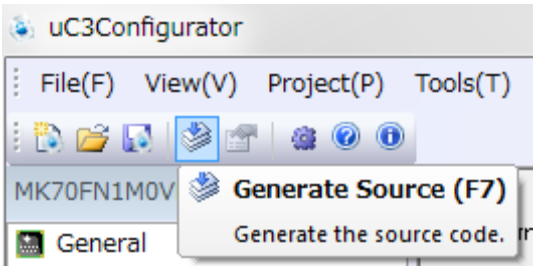
ID Symbol	Queue Depth	Attributes
ID_DTQ1	10	TA_TFIFO

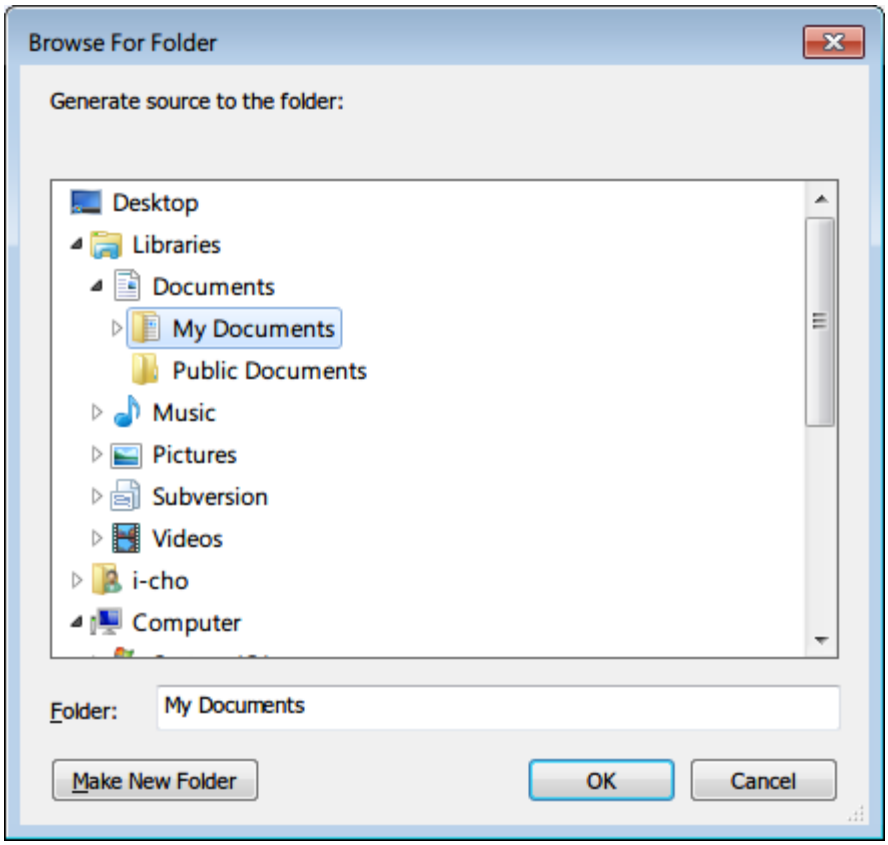
Mailbox

ID Symbol	Attributes
ID_MBX1	TA_TFIFO | TA_MFIFO

4. 2. 4 Generate source

From the Configurator toolbar, click “Generate Source”, open “screen of referring folder” , specify optional folder which deploy to create file and click “OK” .





In case there is already skeleton code main.c existing, previous “main.c” is backed up as “main.bak”.

【Recommendation】

In order to prevent skeleton code from being overwritten and deleted, it is recommended not to directly edit to skeleton code but using template to create application program.

A. Files which are not depended to a surely created processor

File	Content
kernel_id.h	Defined header file of Object ID or Device ID
kernel_cfg.c	Configuration information file of kernel
kernel.h	Header file of kernel
main.c	Skeleton code such as main(), initially set-up function, Task or Handler

B. Files which are not depended to a surely created processor

File	Content
itron.h	Kernel header file
hw_init.c	Initialization file that depends on the hardware and devices
hw_dep.h	Header file that depends on the hardware and devices
Start up	Initialization process by power-on reset (assembly language)
Vector table	Interrupt vector table (assembly language)
Exception Handler	Exception handler, including the interrupt handler (assembly language)
Kernel lib	

C. Files depended on device driver

File	Content
I/O defined file	Header file defining I/O of processor
DDR_xxxxx.c	Source file of device driver
DDR_xxxxx.h	Header file of of device driver
DDR_xxxxx_cfg.h	Configuration file of of device driver

These created files are different according to configuration or processor or device.

4. 2. 5 Error check when creating source

When creating source, the following items will be checked. In case there is some problem, error message will be displayed and file will not be created.

- Check items which must not empty ID or function name.
- Check scope of total ID.
- Check scope of Task Priority Level.
- Check relation of Task Priority Level and Restriction Task attribute among Tasks which use Stack in common.
- Check scope of initial value of Semaphore.
- Check scope of start-up cycle of Cycle Handler.

4. 1. 5. 1 Total ID

All Object ID, including ID used in RTOS which user cannot see, will be managed by unique 8-bit value. Therefore, maximum of total ID will be 255, and number which can create Object will become less than 255.
Total ID is calculated like following formula:

Upper limit of Task Priority Level
Number of Shared Stack
Number of Task
Number of Semaphore
Number of Eventflag
Number of Mailbox
Double number of Data Queues
Number of Fixed-Sized memory pool
＋) Number of Cycle Handler
———————————————————————
Total ID

【Complement】
In the evaluation edition of μC3/Compact, total ID is limited to 16.

CHAPTER 5 Explanation of System Call

5. 1 Task Management Functions

act_tsk	Activate Task
iact_tsk	

【Format】

ER ercd = act_tsk (ID tskid) ;

ER ercd = iact_tsk (ID tskid) ;

【Parameter】

ID	tskid	ID number of Task starting-up

【Return value】

ER	ercd	Successful completion (E_OK)or Error code

【Error code】

E_ID	Incorrect ID number(tskid is incorrect or cannot be used)
E_QOVR	Queuing overflow(Overflow of queuing number required for start-up)

【Call Context】	act_tsk	iact_tsk
Task	Possible	Possible
Time Event Handler	Possible	Possible
Interrupt Service Routine	Possible	Possible

【Explanation】

Start-up Task which is specified by tskid. Concretely, target task is changed from dormant to ready status. Extension information of Task is passed as parameter when starting-up Task.

When the target Task is not in dormant status, Task start-up request will be in queuing. Specifically, 1 will be added to queuing number of Task start-up request. However, when adding 1 to queuing number of Task start-up request and if it exceeds maximum value of queuing number of Task start-up request, it will return to E_QOVR error.

In tskid, definition name of Task ID, which has been created by configurator, is used and specified.

Also, when specifying TSK_SELF(＝0), local Task will be made as target Task. However, when it is specified by a call from Non-Task Context, it will return to E_ID error.

【Recommendation】

Because act_tsk and iact_tsk of µC3/Compact are mounted as the same System Call, it does not relate to Call Context but it is possible to use in the same way. However, it is recommended to use act_tsk in case of calling from Task Context, and act_tsk in other cases.

can_act Cancel Task Activation requests

【Format】

ER_UINT actcnt = can_act (ID tskid) ;

【Parameter】

ID	tskid	ID number of Cancel Task start-up request

【Return value】

ER_UINT	actcnt	Frequency of start-up request in queuing(Positive value or 0)or Error Code

【Error Code】

E_ID	Incorrect ID number(tskidis incorrect or cannot be used)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

For Task specified by tskid, clear queuing number of start-up request, and return to Start-up request queuing number which is before clearing.

In tskid, definition name of Task ID, which has been created by configurator, is used and specified. Also, when specifying TSK_SELF(＝0), local Task will be made as target Task. However, when it is specified by a call from Non-Task Context, it will return to E_ID Error.

sta_tsk Activate Task(with a Start Code)

【Format】

ER ercd = sta_tsk (ID tskid, VP_INT stacd) ;

【Parameter】

ID	tskid	ID number of Task start-up
VP_INT	stacd	Task start-up code

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(tskid is incorrect or cannot be used)
E_OBJ	Object status Error(The target Task is not in dormant status or it is using Shared Stack)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Possible

【Explanation】

Start-up Task which is specified by tskid. Concretely, target task is changed from dormant to ready status. Start-up code(stacd)is passed as parameter when starting-up Task.

In case the target Task is not in dormant status, Task will not be in start-up request queuing but returned to E_OBJ Error. Even in the case when the target Task specifies Shared Stack, it will be returned to E_OBJ Error.

In tskid, definition name of Task ID, which has been created by configurator, is used and specified.

【Recommendation 】

sta_tsk exists for compatibility with the μITRON3.0 specification. In μC3／Compact, there is limit when using Shared Stack, and it is recommended not to use sta_tsk but use act_tsk,iact_tsk.

ext_tsk Terminate Invoking Task

【Format】

void ext_tsk ( ) ;

【Parameter】

No

【Return value】

No

【Call Context】

Task	Possible
Time Event Handler	Impossible
Interrupt Service Routine	Impossible

【Explanation】

Terminate the invoking Task. Concretely, the invokingTask will be changed from execution status to dormant status. In case Start-up request queuing number of the invoking Task is more than 1, subtract 1 from Start-up request queuing number and change the invoking Task to possible execution status. In this time, initialization of Task Priority Level as well as wake-up counter number are also cleared, and stack pointer is initialized as a process for starting up Task. Extension information of Task is passed as parameter when starting-up Task.

In case it is called from Task Context, there will be no return from this System Call. However, in case it is called from Non-Task Context, it will return without returning Error Code.

ter_tsk Terminate Task

【Format】

ER_ercd = ter_tsk (ID tskid) ;

【Parameter】

ID	tskid	ID number of Terminate Task

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(tskidis incorrect or cannot be used)
E_ILUSE	System Call is used incorrectly(the target Task is local Task)
E_OBJ	Object status error(the target Task is in dormant status)

【Call Context】

Task	Possible
Time Event Handler	Impossible
Interrupt Service Routine	Impossible

【Explanation】

Task which is specified by tskid is forcedly changed to dormant status.

In case there is more than 1 Start-up request queuing number of the target Task, subtract 1 from Start-up request queuing number and change it to possible execution status. In this time, initialization of Task Priority Level as well as wake-up counter number are also cleared, and stack pointer is initialized as a process for starting up Task. Extension information of Task is passed as parameter when starting-up Task.

When it is in dormant status, the target Task will return E_OBJError. Also, this System Call is impossible to end local Task. In case of local Task, the target Task will return E_ILUSE Error.

In tskid, definition name of Task ID, which has been created by configurator, is used and specified.

chg_pri Change Task Priority

【Format】

ER_ercd = chg_pri (ID tskid, PRI tskpri) ;

【Parameter】

ID	tskid	ID number of Task to be changed
PRI	tskpri	Base Priority Level after change

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(tskidis incorrect or cannot be used)
E_NOSPT	The target Task is Restriction Task attribute.
E_PAR	ParameterError(tskpri is incorrect)
E_OBJ	Object status error(The target Task isin dormant status)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

The Current Priority of Task specified by tskid to a value specified by tskpri. In tskid, definition name of Task ID, which has been created by configurator, is used and specified. If TSK_SELF(＝0)is specified to tskid, the invlokingTask will be the target Task. Also, if TPRI_INI(＝0)is specified to tskpri, the Current Priority Level of the target Task will be changed to Priority Level when starting up Task.

When the target Task is in runnable state, Priority Order of Task will be changed according to Priority Level after change. In Tasks which have the same priority Level as the Priority Level after change, Priority Order of the target Task will be the lowest.

get_pri Reference Task Priority

【Format】

ER_ercd = get_pri (ID tskid, PRI *p_tskpri) ;

【Parameter】

ID	tskid	ID number of the task to reference

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
PRI	tskpri	The Current Priority Level of Task

【Error Code】

E_ID	Incorrect ID number(tskidis incorrect or cannot be used)
E_OBJ	Object status error(the target Task is in dormant status)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Refer to the Current Priority Level of Task specified by tskid and return to tskpri.

In tskid, definition name of Task ID, which has been created by configurator, is used and specified. If TSK_SELF(＝0)is specified to tskid, local Task will be the target Task.

【Usage】

```
PRI tskpri;                    /*Secure area storing the Current Priority Level of Task */
ER ercd;
                    /* Make pointer to storing area to become Parameter and call it */
ercd = get_pri(ID_Task1, &tskpri);
```

ref_tsk Reference Task status

【Format】

ER ercd = ref_tsk(ID tskid, T_RTSK *pk_rtsk) ;

【Parameter】

ID	Tskid	ID number of the task to be referenced
T_RTSK *	pk_rtsk	Pointer to packet returning status of Task

【Return value】

ER	Ercd	Successful completion(E_OK)or Error Code
Content of pk_rtsk(T_RTSK type)		
STAT	Tskstat	Task status
PRI	Tskpri	The Current Priority Level of Task
PRI	Tskbpri	Base Priority Level of Task
STAT	Tskwait	Waiting factor
ID	Wobjid	ID number of waiting Object
TMO	Lefttmo	Time till time-out
UINT	Actcnt	Start-up request queuing number
UINT	Wupcnt	Wake-up request queuing number
UINT	Suscnt	Forced waiting request nest number(μC3/Compact has not been corresponding yet)

【Error Code】

E_ID	Incorrect ID number(tskidis incorrect or cannot be used)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Refer to status relating to Task specified by tskid, return to packet specified by papk_rtsk.

In tskstat, either of the following value will be returned basing on the target Task status.

TTS_RUN	0x01	Execution status
TTS_RDY	0x02	possible execution status
TTS_WAI	0x04	Waiting status
TTS_DMT	0x10	Dormant status

In case the target Task is not in dormant status, return the Current Priority Level of Task to

tskpri, and Base Priority Level to tskbpri. In µC3/Compact, normally, the Current Priority Level and Base Priority Level are the same. When the target Task is in dormant status, irregular value will be returned.

In tskwait of the case when the target Task is in waiting status, it will return either of the following value based on factor which is becoming waiting status of the target Task. In case the target Task is not in waiting status, an irregular value will be returned.

TTW_SLP	0x0001	Wake-up waiting status
TTW_DLY	0c0002	Time-passing waiting status
TTW_SEM	0x0004	Waiting status of acquiring Semaphore resource
TTW_FLG	0x0008	Eventflag waiting status
TTW_SDTQ	0x0010	Waiting status of transmission to Data Queues
TTW_RDTQ	0x0020	Waiting status of receiving from Data Queues
TTW_MBX	0x0040	Waiting status of receiving from Mailbox
TTW_MPF	0x2000	Waiting status of acquiring Fixed-Sized memory block

In wobjid when the target Task is in waiting status but it is not in either wake-up waiting status or time-passing waiting status, then ID number of waiting Object will be returned. An irregular value will be returned to wobjid in other cases.

In lefttmo when the target Task is in waiting status but not in time-passing waiting status, time which is till the target Task becoming time-out will be returned. Concretely, a value reducing the present time from time which become time-out is returned. However, the value returned to lefttmo will become time secured till time-out, and infact, it will be smaller than time till time-out. Therefore, it will return 0 to lefttmo in case time-out is made by the next Time Tick. When the target Task is in waiting status by permanent waiting (without time-out), it will return TMO_FEVR to lefttmo. An irregular value will be returned to lefttmo of the case which the target Task is not in waiting status or in time-passing waiting status.

Start-up request queuing number of the target Task is returned to actcnt.

When the target Task is not in dormant status, wake-up request queuing number is returned to wupcnt, and forced waiting request nest number is returned to suscnt. An irregular value is returned in case the target Task is in dormant status.

In tskid, definition name of Task ID, which has been created by configurator, is used and specified. If TSK_SELF(＝0)is specified to tskid, local Task will be the target Task.

【Usage】

```
T_RTSK rtsk;            /* Secure area which is storing Task status */
ER ercd;
                        /* Make pointer to storing area to become Parameter and call it */
ercd = ref_tsk (ID_Task1, & rtsk);
```

ref_tst ReferenceTask status(simple edition)

【Format】

ER ercd = ref_tst(ID tskid, T_RTST *pk_rtst) ;

【Parameter】

ID	Tskid	ID number of the task to be referenced
T_RTST *	pk_rtst	Pointer to packet returning the Task status

【Return value】

ER	Ercd	Successful completion(E_OK)or Error Code
Content of pk_rtst(T_RTST type)		
STAT	tskstat	Task status
STAT	tskwait	Waiting factor

【Error Code】

E_ID	Incorrect ID number(tskid is incorrect or cannot be used)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Refer to the lowest status relating to Task which is specified by tskid, and return to packet specified by pk_rtst.

This System Call is a simple edition of ref_tsk. In tskstat and tskwait, a same value with the one returned by ref_tsk will be returned.

In tskid, definition name of Task ID, which has been created by configurator, is used and specified. If TSK_SELF(＝0)is specified to tskid, local Task will be the target Task.

5. 2 Task Dependent Synchronization Functions

slp_tsk	**Put Task to Sleep**
tslp_tsk	**Put Task to Sleep(with timeout)**

【Format】

ER ercd = slp_tsk ( ) ;

ER ercd = tslp_tsk (TMO tmout) ;

【Parameter】

TMO	Tmout	Specify time-out(only tslp_tsk)

【Return value】

ER	Ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_RLWAI	Compulsive release of waiting status(Receive rel_wai while in waiting status)
E_TMOUT	Polling failure or time-out(tslp_tsk)

【Call Context】	**slp_tsk**	**tslp_tsk**
Task	Possible	Possible
Time Event Handler	Impossible	Impossible
Interrupt Service Routine	Impossible	Impossible

【Explanation】

Change local Task to get-up waiting status. However, in case there is more than 1 get-up request queuing number of local Task, subtract 1 from get-up request queuing number, and keep executing without changing local Task to waiting status.

tslp_tsk is System Call which added time-out function to slp_tsk. Also, it is possible to specify TMO_POL(＝0)or TMO_FEVR(＝－1)to tmout. In μC3/Compact, tslp_tsk which specifies TMO_FEVR to tmout will be used as slp_tsk.

wup_tsk	Wake up Task
iwup_tsk	

【Format】

ER ercd = wup_tsk (ID tskid) ;

ER ercd = iwup_tsk (ID tskid) ;

【Parameter】

ID	tskid	ID number of get-up Task

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(tskidis incorrect or cannot be used)
E_OBJ	Object status error(the target Task is in dormant status)
E_QOVR	Queuing overflow(Overflow of wake-up request queuing number)

【Call Context】	wup_tsk	iwup_tsk
Task	Possible	Possible
Time Event Handler	Possible	Possible
Interrupt Service Routine	Possible	Possible

【Explanation】

Task specified by tskid will be released from wake-up request waiting status. E_OK will be returned to the Task which has been released from waiting as a return value of System Call in waiting status. In case the target Task is not either in wake-up request waiting status or dormant status, add 1 to wake-up request queuing number of Task. However, when adding 1 to wake-up request queuing number of Task and if it is exceeding maximum value of wake-up request queuing number, then E_QOVRError will be returned. Also, when it is in dormant status, it will become E_OBJError.

In tskid, definition name of Task ID, which has been created by configurator, is used and specified. If TSK_SELF(＝0)is specified to tskid, local Task will be the target Task. However, when it is specified by a call from Non-Task Context, E_IDError will be returned.

【Recommendation 】

Because wup_tsk and iwup_tsk of μC3/Compact are mounted as the same System Call, so it can be the same Usage, regardless Call Context. However, in case of calling from Task Context, it is recommended to use wup_tsk, or use iwup_tsk in other cases.

can_wup Cancel Task wakeup requests

【Format】

ER_UINT wupcnt = can_wup (ID tskid) ;

【Parameter】

ID	Tskid	ID number of Task for wake-up request cancel

【Return value】

ER_UINT	Wupcnt	Frequency of wake-up request in queuing(positive value or 0)or Error Code

【Error Code】

E_ID	Incorrect ID number(tskidis incorrect or cannot be used)
E_OBJ	Object status error(the target Task is in dormant status)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

For Task specified by tskid, clear queuing number of wake-up request, and return to Wake-up request queuing number which is before clearing.

In tskid, definition name of Task ID, which has been created by configurator, is used and specified. If TSK_SELF(＝0)is specified to tskid, local Task will be the target Task.

rel_wai Release Task from Waiting
irel_wai

【Format】

ER ercd = rel_wai (ID tskid) ;

ER ercd = irel_wai (ID tskid) ;

【Parameter】

ID	tskid	ID number of Task which is for compulsive release of waiting status

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(tskidis incorrect or cannot be used)
E_OBJ	Object status error(when the target Task is not in waiting status)

【Call Context】	**rel_wai**	**irel_wai**
Task	Possible	Possible
Time Event Handler	Possible	Possible
Interrupt Service Routine	Possible	Possible

【Explanation】

When Task specified by tskid is in waiting status, it will be forcedly changed to possible execution status. In the Task which has been released from waiting by this System Call, E_RLWAIError will be returned as a return value of System Call in waiting status. In case the target Task is not in waiting status, E_OBJError will be returned.

In tskid, definition name of Task ID, which has been created by configurator, is used and specified.

【Recommendation 】

Because rel_wai and irel_wai of μC3/Compact are mounted as the same System Call, so it can be the same Usage, regardless Call Context. However, in case of calling from Task Context, it is recommended to use rel_wai, or use irel_wai in other cases.

dly_tsk Delay Task

【Format】

ER ercd = dly_tsk (RELTIM dlytim) ;

【Parameter】

RELTIM	Dlytim	Amount of time to delay the invoking task(relative time)

【Return value】

ER	Ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_RLWAI	Compulsive release of waiting status(receive rel_wai in waiting status)

【Call Context】

Task	Possible
Time Event Handler	Impossible
Interrupt Service Routine	Impossible

【Explanation】

This service call delays the execution of the invoking task for the amount of time specified in dlytim. When the task is released from waiting after the relative time expires, the service call completes and returns E_OK.

5. 3 Synchronization and Communication Functions

5. 3. 1 Semaphores

sig_sem	Release Semaphore resource
isig_sem	

【Format】

ER ercd = sig_sem(ID semid) ;

ER ercd = isig_sem(ID semid) ;

【Parameter】

ID	semid	ID number of Semaphore resource return

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(semid is incorrect or cannot be used)
E_QOVR	Queuing overflow(return when maximum resource number is exceeded)

【Call Context】	sig_sem	isig_sem
Task	Possible	Possible
Time Event Handler	Possible	Possible
Interrupt Service Routine	Possible	Possible

【Explanation】

When there is Task which is waiting to acquire rsource for Semaphore specified by semid, then release the Task waiting at the beginning in the waiting queue and change it to possible execution status. At this time, resource number of the target Semaphore is not changed. Also, return E_OK to Task which has been released from waiting as a return value of System Call in waiting status. In case there is not Task waiting for acquiring resource, add 1 to resource number of the target Semaphore. When adding 1 to resource number of Semaphore and if it exceeds maximum resource number of Semaphore, then return E_QOVRError.

In semid, definition name of Semaphore ID created by configurator is used and specified.

【Recommendation 】

Because sig_sem and isig_sem of μC3/Compact are mounted as the same System Call, so it can be the same Usage, regardless Call Context. However, in case of calling from Task Context, it is recommended to use sig_sem, or use isig_sem in other cases.

wai_sem	**Acquire Semaphore resource**
pol_sem	**Acquire Semaphore resource(Polling)**
twai_sem	**Acquire Semaphore resource(With time-out)**

【Format】

ER ercd = wai_sem(ID semid) ;

ER ercd = pol_sem(ID semid) ;

ER ercd = twai_sem(ID semid, TMO tmout) ;

【Parameter】

ID	Semid	ID number of Semaphore resource acquisition
TMO	Tmout	Specify time-out(only twai_sem)

【Return value】

ER	Ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(semidis incorrect or cannot be used)
E_PAR	ParameterError(tmout is incorrect；only twai_sem)
E_RLWAI	Compulsive release of waiting status(receive rel_wai in waiting status；other than pol_sem)
E_TMOUT	Polling failure or time-out(other than wai_sem)

【Call Context】	**wai_sem**	**pol_sem**	**twai_sem**
Task	Possible	Possible	Possible
Time Event Handler	Impossible	Possible	Impossible
Interrupt Service Routine	Impossible	Impossible	Impossible

【Explanation】

Acquire 1 resource from Semaphore specified by semid. In case there is more than 1 Semaphore resource number, then subtract 1 from Semaphore resource number so that it will not be in waiting status and end System Call. When Semaphore resource number is 0, leave the resource number in 0, connect to waiting queue of local Task, and change it to Semaphore resource acquisition waiting status. When other Tasks are already in waiting queue, it will be connected to the last of waiting queue of local Task.

pol_sem is System Call running process of wai_sem by pooling, and twai_sem is System Call which is added time-out function to wai_sem. Also, it is possible to specify TMO_POL(＝0)or TMO_FEVR(＝-1)to tmout. In μC3/Compact, twai_sem specifying TMO_FEVR to tmout is used as wai_sem, and twai_sem specifying TMO_POL to tmout is used as pol_sem.

In semid, definition name of Semaphore ID created by configurator is used and specified.

ref_sem Reference Semaphore state

【Format】

ER ercd = ref_sem(ID semid, T_RSEM *pk_rsem) ;

【Parameter】

ID	semid	ID number of Semaphore for refering status
T_RSEM*	pk_rsem	Pointer to packet which returns Semaphore status

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
Content of pk_rsem(T_RSEM type)		
ID	wtskid	ID number of Task in the beginning of waiting queue of Semaphore
UINT	semcnt	The present resource number of Semaphore

【Error Code】

E_ID	Incorrect ID number(semidis incorrect or cannot be used)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Refer status of Semaphore specified by semid and return to packet specified by pk_rsem.

Return ID number of Task in the beginning of Semaphore waiting queue to wtskid. In case there is no Task waiting for resource acquisition, return TSK_NONE(=0).

Return the present resource number of Semaphore to semcnt.

In semid, definition name of Semaphore ID created by configurator is used and specified.

5. 3. 2 Eventflags

set_flg	Set Eventflag
iset_flg	

【Format】

ER ercd = set_flg(ID flgid, FLGPTN setptn) ;

ER ercd = iset_flg(ID flgid, FLGPTN setptn) ;

【Parameter】

ID	flgid	ID number of Eventflag for setting
FLGPTN	setptn	Set bit pattern

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(flgid is incorrect or cannot be used)
E_PAR	ParameterError(setptn is incorrect)

【Call Context】	set_flg	iset_flg
Task	Possible	Possible
Time Event Handler	Possible	Possible
Interrupt Service Routine	Possible	Possible

【Explanation】

Bit pattern of Eventflag specified by flgid will be updated by bit pattern before calling System Call, and logical disjunction (OR)of each bit of setptn value. Update bit pattern of Eventflag and search to see if waiting release condition, which is in order from Task of the beginning of Eventflag's waiting queue, is satisfied or not, and when a Task satisfying waiting release condition is found, that Task will be released from waiting. Besides, return E_OK as a return value of System Call in waiting status to the Task which has been released from waiting. At this time, if there is TA_CLR attribute specified in Eventflag attribute, then clear all bits in bit pattern of Eventflag, and end process of System Call. In case there is no TA_CLRattribute specified, keep searching for waiting queue.

flgid uses and specifies definition name of Eventflag ID which is created by configurator.

【Recommendation 】

Because set_flg and iset_flg of μC3/Compact are mounted as the same System Call, so it can be the same Usage, regardless Call Context. However, in case of calling from Task Context, it is recommended to use set_flg, or use iset_flg in other cases.

clr_flg Clear Eventflag

【Format】

ER ercd = clr_flg(ID flgid, FLGPTN clrptn) ;

【Parameter】

ID	flgid	ID number of the set Eventflag
FLGPTN	clrptn	The cleared bit pattern(Reversing value of each bit)

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(flgidis incorrect or cannot be used)
E_NOEXS	Object which has not been created(Eventflag is unregistered)
E_PAR	ParameterError(clrptn is incorrect)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Bit pattern of Eventflag specified by flgid is updated by bit pattern before calling System Call and logical junction (AND) of each bit of clrptn value.

In flgid, definition name of Eventflag ID which is created by configurator is used and specified.

【Usage】

```
ER ercd;
        /* Make value which set to 0 for only cleared bit as Parameter and call it */
ercd = clr_flg(ID_Flag1, ~0x0001);
```

wai_flg	**Waiting Eventflag**
pol_flg	**Waiting Eventflag(Polling)**
twai_flg	**Waiting Eventflag(with time-out)**

【Format】

```
ER ercd = wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode,
                  FLGPTN   *p_flgptn) ;
ER ercd = pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode,
                  FLGPTN   *p_flgptn) ;
ER ercd = twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode,
                  FLGPTN   *pflgptn, TMO tmout) ;
```

【Parameter】

ID	flgid	ID number of waiting Eventflag
FLGPTN	waiptn	Waiting bit pattern
MODE	wfmode	Waiting mode
TMO	tmout	Specify time-out(only twai_flg)

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
FLGPTN	flgptn	Bit pattern when release waiting

【Error Code】

E_ID	Incorrect ID number(flgidis incorrect or cannot be used)
E_PAR	ParameterError(waiptn, wfmode is incorrect)
E_ILUSE	System Call is used incorrectly(with waiting Task by Eventflag specified by TA_WSGL attribute)
E_RLWAI	Compulsive release of waiting status(receive rel_wai in waiting status；other than pol_flg)
E_TMOUT	Polling failure or time-out(other than wai_flg)

【Call Context】	**wai_flg**	**pol_flg**	**twai_flg**
Task	Possible	Possible	Possible
Time Event Handler	Impossible	Possible	Impossible
Interrupt Service Routine	Impossible	Impossible	Impossible

【Explanation】

If bit pattern of Eventflag specified by flgid is not satisfied to waiting release condition specified by waiptn and wfmode, connect it to waiting queue till it satisfies the condition, and change to Eventflag waiting status. In case it satisfies the waiting release condition specified by

waiptn and wfmode, end process of System Call without making local Task to waiting status, and return bit pattern which satisfied waiting status to flgptn. At this time, if TA_CLR attribute is specified to Eventflag attribute, then clear all bits of bit pattern of Eventflag.

When TA_WSGL attribute is specified to Eventflag attribute and other Task is connected to waiting queue of Eventflag, it will become E_ILUSEError, regardless condition of waiting release.

In wfmode, it is possible to specify either TWF_ANDW or TWF_ORW. The waiting release condition specified by waiptn and wfmode is a condition when all bits specified by waiptn of bit pattern of Eventflag is ser in case TWF_ANDW is specified in wfmode. If TWF_ORW is specified, it is a condition when either bit specified by waiptn of bit pattern of Eventflag is set.

pol_flg is a System Call running process of wai_flg by polling, and twai_flg is a System Call which is added time-out function to wai_flg. Also, it is possible to specify TMO_POL(=0)or TMO_FEVR(=-1)to tmout. In µC3/Compact, twai_flg which specifies TMO_FEVR to tmout is used as wai_flg, and twai_flg which specifies TMO_POL to tmout is used as pol_flg.

In flgid, definition name of Eventflag ID which is created by configurator is used and specified.

【Usage】

```
FLGPTN waiptn;   /* Secure area storing Flag pattern */
ER ercd;
                        /* Make pointer to storing area to become Parameter and call it */
ercd = wai_flg(ID_Flag1, 0x0003, TWF_ORW, &waiptn);
if (ercd == E_OK) {
     if ((waiptn & 0x0001) != 0) {
          ercd = clr_flg(ID_Flag1, ~0x0001);
     }
}
```

ref_flg Refer status of Eventflag

【Format】

ER ercd = ref_fig(ID flgid, T_RFLG *pk_rflg) ;

【Parameter】

ID	flgid	ID number of refering status of Eventflag
T_RFLG*	pk_rflg	Pointer to packet which returns Eventflag status

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
Content of pk_rflg(T_RFLG type)		
ID	wtskid	ID number of Task in the beginning of waiting queue of Eventflag
FLGPTN	flgptn	The present bit pattern of Eventflag

【Error Code】

E_ID	Incorrect ID number(flgidis incorrect or cannot be used)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Refer status of Eventflag specified by flgid, and return packet specified by pk_rflg.

In wtskid, return ID number of Task in the beginning of waiting queue of Eventflag. In case there is not Task waiting for Event, TSK_NONE(＝0)will be returned.

In flgptn, the present bit pattern of Eventflag will be returned.

In flgid, definition name of Eventflag ID which is created by configurator is used and specified.

5. 3. 3 Data Queues

snd_dtq	**Send to Data Queue**
psnd_dtq	**Send to Data Queue (Polling)**
ipsnd_dtq	
tsnd_dtq	**Send to Data Queue (with time-out)**

【Format】

ER ercd = snd_dtq(ID dtqid, VP_INT data) ;
ER ercd = psnd_dtq(ID dtqid, VP_INT data) ;
ER ercd = ipsnd_dtq(ID dtqid, VP_INT data) ;
ER ercd = tsnd_dtq(ID dtqid, VP_INT data, TMO tmout)

【Parameter】

ID	dtqid	ID number of Data Queues for transmission
VP_INT	data	Data sent to Data Queues
TMO	tmout	Specify time-out(only tsnd_dtq)

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(dtqidis incorrect or cannot be used)
E_RLWAI	Compulsive release of waiting status(Receive rel_wai in waiting status ; only snd_dtq, tsnd_dtq)
E_TMOUT	Polling failure or time-out(other than snd_dtq)

【Call Context】	**snd_dtq**	**psnd_dtq**	**ipsnd_dtq**	**tsnd_dtq**
Task	Possible	Possible	Possible	Possible
Time Event Handler	Impossible	Possible	Possible	Impossible
Interrupt Service Routine	Impossible	Possible	Possible	Impossible

【Explanation】

When there is Task waiting for receiving to Data Queues specified by dtqid, pass data sending to the Task in the beginning of receiving-waiting queue, and release waiting for that Task. Besides, return E_OK to the Task which has been released from waiting as a return value of System Call in waiting status, and return data value as data received from Data Queues. In case there is no Task waiting for receiving, put sending data to the end of Data Queues. If there is no space in Data Queues area, connect local Task to sending-waiting queue and change it to sending-waiting status to Data Queues.

In case there is no Task in psnd_dtq and ipsnd_dtq waiting for receiving to Data Queues, and

there is no space in Data Queues area, then return E_TMOUTError.

psnd_dtq and ipsnd_dtq is System Call running process of snd_dtq by polling, and tsnd_dtq is System Call which added time-out function to snd_dtq. Also, it is possible to specify TMO_POL(＝0)or TMO_FEVR(＝－1)to tmout. In μC3/Compact, tsnd_dtq specifying TMO_FEVR to tmout is used as snd_dtq, and tsnd_dtq specifying TMO_POL to tmout is used as psnd_dtq.

In dtqid, definition name of Data Queues ID which is created by configurator is used and specified.

【Recommendation 】

Because psnd_dtq and ipsnd_dtq of μC3/Compact are mounted as the same System Call, so it can be the same Usage, regardless Call Context. However, in case of calling from Task Context, it is recommended to use psnd_dtq, or use ipsnd_dtq in other cases.

fsnd_dtq	**Forced Send to Data Queue**
ifsnd_dtq	

【Format】

ER ercd = fsnd_dtq(ID dtqid, VP_INT data) ;

ER ercd = ifsnd_dtq(ID dtqid, VP_INT data) ;

【Parameter】

ID	dtqid	ID number of Data Queues transmission
VP_INT	data	Data sent to Data Queues

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(dtqidis incorrect or cannot be used)
E_ILUSE	System Call is used incorrectly(Capacity of Data Queues area of the target Data Queues is 0)

【Call Context】	**fsnd_dtq**	**ifsnd_dtq**
Task	Possible	Possible
Time Event Handler	Possible	Possible
Interrupt Service Routine	Possible	Possible

【Explanation】

When there is Task waiting for receiving by Data Queues specified by dtqid, pass sending data to Task in the beginning of receiving-waiting queue, and release that Task from waiting status. Also, return E_OK as return value of System Call in waiting status to the Task which has been released from waiting, and return data value as data received from Data Queues. In case there is no Task in waiting for receiving, then put sending data to the end of Data Queues. Here, if there is no space in Data Queues area, then delete the beginning data of Data Queues, secure necessary area for Data Queues, and put sending data to the end of Data Queues. In other words, the oldest data is deleted. In these System Calls, when compulsive transmission of data is tried form Data Queues in which capacity of Data Queues area is 0, it will return E_ILUSEError.

In dtqid, definition name of Data Queues ID which is created by configurator is used and specified.

【Recommendation 】

Because fsnd_dtq and ifsnd_dtq µC3/Compact are mounted as the same System Call, so it can be the same Usage, regardless Call Context. However, in case of calling from Task Context, it is recommended to use fsnd_dtq, or use ifsnd_dtq in other cases.

rcv_dtq	Receive from Data Queue
prcv_dtq	Receive from Data Queue(Polling)
trcv_dtq	Receive from Data Queue(with time-out)

【Format】

```
ER ercd = rcv_dtq(ID dtqid, VP_INT   *p_data) ;
ER ercd = prcv_dtq(ID dtqid, VP_INT   *p_data) ;
ER ercd = trcv_dtq(ID dtqid, VP_INT   *p_data, TMO tmout) ;
```

【Parameter】

ID	dtqid	ID number of receiving target of Data Queues
TMO	tmout	Specify time-out(only trcv_dtq)

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
VP_INT	data	Data received from Data Queues

【Error Code】

E_ID	Incorrect ID number(dtqid is incorrect or cannot be used)
E_RLWAI	Compulsive release of waiting status(Receive rel_wai in waiting status ; other than prcv_dtq)
E_TMOUT	Polling failure or time-out(other than rcv_dtq)

【Call Context】	rcv_dtq	prcv_dtq	trcv_dtq
Task	Possible	Possible	Possible
Time Event Handler	Possible	Possible	Impossible
Interrupt Service Routine	Impossible	Impossible	Impossible

【Explanation】

When there is data in Data Queues specified by dtqid, the beginning data will be taken out and returned to data. In case there is Task waiting for sending by Data Queues, then put data of Task in the beginning of transmission waiting queue for sending to the end of Data Queues, and release that Task from waiting status. Also, return E_OK as a return value of System Call in waiting status to Task which had been release from waiting.

When there is Task waiting for transmission by Data Queues but in a status which there is no data, receive data of Task for sending from Task in the beginning of transmission waiting queue, and release that Task from waiting status. Also, return E_OK to the Task which has been

release from waiting as a return value of System Call in waiting status. Return the received data to data.

In case there is no either data or Task waiting for transmission, local Task will be connected to receiving-waiting queue, and it is changed to receiving-waiting status from Data Queues.

prcv_dtq is System Call running process of rcv_dtq by polling, and trcv_dtq is System Call which added time-out function to rcv_dtq. Also, it is possible to specify TMO_POL(＝0)or TMO_FEVR(＝－1)to tmout. In µC3/Compact, trcv_dtq specifying TMO_FEVR to tmout is used as rcv_dtq, and trcv_dtq specifying TMO_POL to tmout is used as prcv_dtq.

In dtqid, definition name of Data Queues ID which is created by configurator is used and specified.

ref_dtq	**Refer to Data Queues status**

【Format】

ER ercd = ref_dtq(ID dtqid, T_RDTQ *pk_rdtq) ;

【Parameter】

ID	dtqid	ID number of Data Queues for status reference
T_RDTQ*	pk_rdtq	Pointer to packet which returns Data Queues status

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
Content of pk_rdtq(T_RDTQ type)		
ID	stskid	ID number of Task in the beginning of transmission waiting queue of Data Queues
ID	rtskid	ID number of Task in the beginning of receiving- waiting queue of Data Queues
UINT	sdtqcnt	Number of data in Data Queues

【Error Code】

E_ID	Incorrect ID number(dtqidis incorrect or cannot be used)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Refer status of Data Queues specified by dtqid, and return to packet specified by pk_rdtq.

Return ID number of Task in the beginning of transmission waiting queue of Data Queues to stskid. In case there is no Task waiting for transmission, then return TSK_NONE(=0).

Return ID number of Task in the beginning of receiving-waiting queue of Data Queues to rtskid. In case there is no Task waiting for receiving, then return TSK_NONE(=0).

Return number of data existing currently in Data Queues to sdqcnt.

In dtqid, definition name of Data Queues ID which is created by configurator is used and specified.

5. 3. 4 Mailboxes

snd_mbx	Send to Mailbox	
【Format】		
ER ercd = snd_mbx(ID mbxid, T_MSG *pk_msg) ;		
【Parameter】		
ID	mbxid	ID number of Mailbox for transmission
T_MSG*	pk_msg	The beginning number of message packet sent to Mailbox
【Return value】		
ER	ercd	Successful completion(E_OK)or Error Code
【Error Code】		
E_ID	Incorrect ID number(mbxidis incorrect or cannot be used)	
【Call Context】		
Task	Possible	
Time Event Handler	Possible	
Interrupt Service Routine	Impossible	

【Explanation】

When there is Task waiting for receiving by Mailbox specified by mbxid, pass the beginning number of message packet specified by pk_msg to Task in the beginning of waiting queue, and release that Task from waiting status. Also, return E_OK to the Task which has been released from waiting as a return value of System Call in waiting status, and return value of pk_msg as the beginning number of message packet received from Mailbox.

In case there is no Task in waiting for receiving, then put message packet, in which pk_msg is made as the beginning number, to the end of Message cue. At this time, sending message packet must be already flowing in Message cue of Mailbox.

In mbxid, definition name of Mailbox ID which is created by configurator is used and specified.

【Usage】

```
T_MSGPKT* pk_msgpkt;    /* Secure area storing the beginning number of message packet */
ER ercd;
ercd = get_mpf(ID_Mpf1, &pk_msgpkt);
if (ercd == E_OK) {
        /* Edit message packet */
        /* Make pointer to storing area to become Parameter and call it */
    ercd = snd_mbx(ID_Mbx1, (T_MSG*)pk_msgpkt);
}
```

rcv_mbx	Receive from Mailbox
prcv_mbx	Receive from Mailbox(Polling)
trcv_mbx	Receive from Mailbox(with time-out)

【Format】

ER ercd = rcv_mbx(ID mbxid, T_MSG **ppk_msg) ;

ER ercd = prcv_mbx(ID mbxid, T_MSG **ppk_msg) ;

ER ercd = trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout) ;

【Parameter】

ID	mbxid	ID number of receiving Mailbox
TMO	tmout	Specify time-out(only trcv_mbx)

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
T_MSG*	pk_msg	The beginning number of message packet received from Mailbox

【Error Code】

E_ID	Incorrect ID number(mbxidis incorrect or cannot be used)
E_RLWAI	Compulsive release of waiting status(Receive rel_wai in waiting status ; other than prcv_mbx)
E_TMOUT	Polling failure or time-out(other than rcv_mbx)

【Call Context】	rcv_mbx	prcv_mbx	trcv_mbx
Task	Possible	Possible	Possible
Time Event Handler	Impossible	Possible	Impossible
Interrupt Service Routine	Impossible	Impossible	Impossible

【Explanation】

If there is message in Message cue of Mailbox specified by mbxid, that message packet in the beginning will be taken out, and that beginning number is returned to pk_msg. When there is no message, local Task will be connected to waiting queue, and it is changed to receiving status from Mailbox.

prcv_mbx is System Call running process of rcv_mbx by Polling, and trcv_mbx is System Call which added time-out function to rcv_mbx. Also, it is possible to specify TMO_POL(＝0) or TMO_FEVR(＝－1)to tmout. In μC3/Compact, trcv_mbx specifying TMO_FEVR to tmout is used as rcv_mbx, and trcv_mbx specifying TMO_POL to tmout is used as prcv_mbx.

In mbxid, definition name of Mailbox ID which is created by configurator is used and specified.

【Usage】

```
T_MSGPKT* pk_msgpkt;    /* Secure area storing the beginning number of message packet*/
ER ercd;

ercd = rcv_mbx(ID_Mbx1, (T_MSG    **)&pk_msgpkt);
if (ercd == E_OK) {
                    /* Confirm message packet */
                    /* Make number of memory block as Parameter and call it */
      ercd = rel_mpf(ID_mpf1, pk_msgpkt);
}
```

ref_mbx　　Reference Mailbox State

【Format】

ER ercd = ref_mbx(ID mbxid, T_RMBX *pk_rmbx) ;

【Parameter】

ID	mbxid	ID number of Mailbox for status reference
T_RMBX*	pk_rmbx	Pointer to packet which returns Mailbox status

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

Content of pk_rmbx(T_RMBX type)

ID	wtskid	ID number of Task in the beginning of waiting queue of Mailbox
T_MSG*	pk_msg	The beginning number of message packet of Message cue

【Error Code】

E_ID	Incorrect ID number(mbxidis incorrect or cannot be used)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Refer to status of Mailbox specified by mbxid, and return to packet specified by pk_rmbx.

Return ID number of Task in the beginning of Mailbox waiting queue to wtskid. In case there is no Task waiting for receiving, then return TSK_NONE(＝0).

Return the beginning number of message packet in the beginning of Message cue of Mailbox to pk_msg. If there is no message in Message cue, return NULL(＝0).

In mbxid, definition name of Mailbox ID which is created by configurator is used and specified.

5. 4 Memory Pool Management Functions

5. 4. 1 Fixed-Sized Memory Pools

get_mpf	**Acquire Fixed-Sized Memory Block**

pget_mpf	Acquire Fixed-Sized Memory Block(Polling)
tget_mpf	Acquire Fixed-Sized Memory Block(with time-out)

【Format】

ER ercd = get_mpf(ID mpfid, VP *p_blk) ;

ER ercd = pget_mpf(ID mpfid, VP *p_blk) ;

ER ercd = tget_mpf(ID mpfid, VP *p_blk, TMO tmout) ;

【Parameter】

ID	mpfid	ID number of Fixed-Sized memory pool of memory block acquisition
TMO	tmout	Specify time-out(only tget_mpf)

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
VP	blk	The beginning number of the acquired memory block

【Error Code】

E_ID	Incorrect ID number(mpfidis incorrect or cannot be used)
E_RLWAI	Compulsive release of waiting status(Receive rel_wai in waiting status ; other than pget_mpf)
E_TMOUT	Polling failure or time-out(other than get_mpf)

【Call Context】	get_mpf	pget_mpf	tget_pmf
Task	Possible	Possible	Possible
Time Event Handler	Impossible	Possible	Impossible
Interrupt Service Routine	Impossible	Impossible	Impossible

【Explanation】

In case there is empty memory block in memory area of Fixed-Sized memory pool specified by mpfid, select some in them and return that beginning number to blk.

If there is no empty memory block, then connect local Task to waiting queue, and changed it to waiting status of Fixed-Sized memory pool acquisition. pget_mpf is System Call running process of get_mpf by Polling, and tget_mpf is System Call which added time-out function to get_mpf. Also, it is possible to specify TMO_POL(＝0) or TMO_FEVR(＝－1)to tmout. In μC3/Compact, tget_mpf specifying TMO_FEVR to tmout is used as get_mpf, and tget_mpf specifying TMO_POL to tmout is used as pget_mpf. In mpfid, definition name of Fixed-Sized memory pool ID which is created by configurator is used and specified.

rel_mpf Release Fixed-Sized Memory Block

【Format】

ER ercd = rel_mpf(ID mpfid, VP blk) ;

【Parameter】

ID	mpfid	ID number of Fixed-Sized memory pool for returning memory block
VP	blk	The beginning number of memory block for returning

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(mpfidis incorrect or cannot be used)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

In case there is no Task waiting for acquiring memory block by Fixed-Sized memory pool which is specified by mpfid, then memory block in which blk is made as the beginning number will be returned to memory area of that Fixed-Sized memory pool.

If there is Task waiting for acquisition, then acquire the returned mamory block in Task in the beginning of waiting queue, and release that Task from waiting. Also, return E_OK to Task which was released from waiting as a return value of System Call in waiting status, and return value of blk as the beginning number of memory block acquired from Fixed-Sized memory block.

The beginning number of the returned memory block, which is retuned, is the one of acquired memory block from Fixed-Sized memory pool specified by mpfid, so it must be the one which has not been returned.

In mpfid, definition name of Fixed-Sized memory pool ID which is created by configurator is used and specified.

ref_mpf Reference Fixed-Sized Memory Pool State

【Format】

ER ercd = ref_mpf(ID mpfid, T_RMPF *pk_rmpf) ;

【Parameter】

ID	mpfid	ID number of Fixed-Sized memory pool for status reference
T_RMPF*	pk_rmpf	Pointer to oacket which returns status of Fixed-Sized memory pool

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

Content of pk_rmpf(T_RMPF type)

ID	wtskid	ID number of Task in the beginning of waiting queue of Fixed-Sized memory pool
UINT	fblkcnt	Empty memory block number of Fixed-Sized memory pool(Number)

【Error Code】

E_ID	Incorrect ID number(mpfidis incorrect or cannot be used)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Refer to status of Fixed-Sized memory pool specified by mpfid, and return to packet specified by pk_rmpf.

Return ID number of Task in the beginning of waiting queue of Fixed-Sized memory pool to wtskid. If there is no Task waiting for acquiring memory block, TSK_NONE(＝0) will be returned.

Return number of empty memory block in area of Fixed-Sized memory pool to fblkcnt.

In mpfid, definition name of Fixed-Sized memory pool ID which is created by configurator is used and specified.

5. 5 Time Management Functions

5. 5. 1 System Time Management

set_tim	Set System Time

【Format】

ER ercd = set_tim(SYSTIM *p_systim) ;

【Parameter】

SYSTIM	systim	Set up time to System time

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

There is no Error that should be mentioned specially.

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Set up the present System time so that it will display time in system. Also, Time-out time of System which already has been called will not be changed by the change of System time.

get_tim Reference System Time

【Format】

ER ercd = get_tim(SYSTIM *p_systim) ;

【Parameter】

No

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
SYSTIM	systim	System time at the present

【Error Code】

There is no Error that should be mentioned specially.

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Call out the present System time and return to system.

isig_tim Supply Time Tick

【Format】

ER ercd = isig_tim( ) ;

【Parameter】

No

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

There is no Error that should be mentioned specially.

【Call Context】

Task	Impossible
Time Event Handler	Impossible
Interrupt Service Routine	Possible

【Explanation】

Tick time is added to System Time.

5. 5. 2 Cyclic Handlers

sta_cyc	Start Cyclic Handler Operation

【Format】

ER ercd = sta_cyc(ID cycid) ;

【Parameter】

ID	cycid	ID number of operation starting Cycle Handler

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(cycidis incorrect or cannot be used)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

When there is no TA_PHS attribute specified in Cycle handler which is specified by cycid, it will be changed to operating status. Also, when System Call is called, Time adding starting-up cycle of Cycle Handler is made as time which should start the next Cycle Handler. At this time, if it has already been in operation, then change only time which should nextly start-up. In case TA_PHS attribute is specified, change status which has not been in operation to operating status, and do nothing if it's in operating status.

In cycid, definition name of Cycle Handler ID which is created by configurator is used and specified.

stp_cyc Stop Cyclic Handler Operation

【Format】

ER ercd = stp_cyc(ID cycid) ;

【Parameter】

ID	cycid	ID number of Cycle handler for stopping operation

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(cycidis incorrect or cannot be used)
E_NOEXS	Object which has not been created(The target Cycle Handler has not been registered yet)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

In case Cycle Handler specified by cycid is in operation status, change it to non-operation status. And do nothing if it is in non-operation status.

In cycid, definition name of Cycle Handler ID which is created by configurator is used and specified.

ref_cyc Reference Cyclic Handler State

【Format】

ER ercd = ref_cyc(ID cycid, T_RCYC *pk_rcyc) ;

【Parameter】

ID	cycid	ID number of Cycle Handler for status reference
T_RCYC*	pk_rcyc	Pointer to packet which returns status of Cycle Handler

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

Content of pk_rcyc(T_RCYC type)

STAT	cycstat	Operation status of Cycle Handler
RELTIM	lefttim	Time till next starting-up of Cycle Handler

【Error Code】

E_ID	Incorrect ID number(cycidis incorrect or cannot be used)

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Refer to status of Cycle Handler specified by cycid, and return to packet which is specified by pk_rcyc.

In cycstat, it will return to either in the following value depending on operation status or non-operation status of Cycle Handler.

TCYC_STP	0x00	Cycle Handler is not in operation
TCYC_STA	0x01	Cycle Handler is in operation

In lefttim, if the target Cycle Handler is in operation status, time that is till next starting-up of the target Cycle Handler will be returned. However, the return value of lefttim is the time which secures till next starting-up of Cycle Handler. Therefore, if Cycle Handler is started up in the next Time Tick, 0 will be returned to lefttim. When the target Cycle Handler is in non-operation status, an irregular value will be returned to lefttim.

In cycid, definition name of Cycle Handler ID which is created by configurator is used and specified.

5. 6 System State Management Functions

rot_rdq	Rotate Task Precedence
irot_rdq	

【Format】

ER ercd = rot_rdq(PRI tskpri) ;

ER ercd = irot_rdq(PRI tskpri) ;

【Parameter】

PRI	tskpri	Priority Level of object that rotates Priority Level

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_NOSPT	The Task in the beginning of Priority Order of the target Priority Level is Restriction Task Attribute

【Call Context】	rot_rdq	irot_rdq
Task	Possible	Possible
Time Event Handler	Possible	Possible
Interrupt Service Routine	Possible	Possible

【Explanation】

Rotate Priority Order of Task of Priority Level specified by tskpri. In other words, make Task, which is in Tasks having Priority Level and in runnable state, become the one of highest Priority Order; and make Task which has the same Priority Level become the one of lowest Priority Order. If specifying TPRI_SELF(=0)to tskpri, base Priority Level of local Task will become the target Priority Level.

【Recommendation 】

Because rot_rdq and irot_rdq of μC3/Compact are mounted as the same System Call, so it can be the same Usage, regardless Call Context. However, in case of calling from Task Context, it is recommended to use rot_rdq, or use irot_rdq in other cases.

get_tid	Reference Task ID in the RUNNING State
iget_tid	

【Format】

ER ercd = get_tid(ID *p_tskid) ;

ER ercd = iget_tid(ID *p_tskid) ;

【Parameter】

No

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
ID	tskid	ID number of Task in execution status

【Error Code】

E_PAR	ParameterError(p_tskid is incorrect)

【Call Context】	**get_tid**	**iget_tid**
Task	Possible	Possible
Time Event Handler	Possible	Possible
Interrupt Service Routine	Possible	Possible

【Explanation】

Refer to ID number of Task in execution status, and return it to tskid. When there is no Task in execution status if it is called from Non-Task Context, return TSK_NONE(＝0)to tskid.

【Recommendation 】

Because get_tid and iget_tid of µC3/Compact are mounted as the same System Call, so it can be the same Usage, regardless Call Context. However, in case of calling from Task Context, it is recommended to use get_tid, or use iget_tid in other cases.

loc_cpu　　Lock the CPU
iloc_cpu

【Format】

ER ercd = loc_cpu( ) ;

ER ercd = iloc_cpu( ) ;

【Parameter】

No

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

There is no Error that should be mentioned specially.

【Call Context】	**loc_cpu**	**iloc_cpu**
Task	Possible	Possible
Time Event Handler	Possible	Possible
Interrupt Service Routine	Possible	Possible

【Explanation】

Change to CPU Lock status. And do nothing if it is called by CPU Lock status.
CPU Lock status is depending on processor, so please refer to “processor dependence part Manual” for more explanation.

【Recommendation 】

Because loc_cpu and iloc_cpu of μC3/Compact are mounted as the same System Call, so it can be the same Usage, regardless Call Context. However, in case of calling from Task Context, it is recommended to use loc_cpu, or use iloc_cpu in other cases.

unl_cpu　　Unlock the CPU

iunl_cpu

【Format】

ER ercd = unl_cpu( ) ;

ER ercd = iunl_cpu( ) ;

【Parameter】

No

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

There is no Error that should be mentioned specially

【Call Context】	**unl_cpu**	**iunl_cpu**
Task	Possible	Possible
Time Event Handler	Possible	Possible
Interrupt Service Routine	Possible	Possible

【Explanation】

Change to release CPU Lock status. And do nothing if it is called by release CPU Lock status. CPU Lock status release is depending on processor, so please refer to "processor dependence part Manual" for more explanation.

【Recommendation 】

Because unl_cpu and iunl_cpu of µC3/Compact are mounted as the same System Call, so it can be the same Usage, regardless Call Context. However, in case of calling from Task Context, it is recommended to use unl_cpu , or use iunl_cpu in other cases.

.

dis_dsp Disable Dispatching

【Format】

ER ercd = dis_dsp( ) ;

【Parameter】

No

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

There is no Error that should be mentioned specially.

【Call Context】

Task	Possible
Time Event Handler	Impossible
Interrupt Service Routine	Impossible

【Explanation】

Change to Dispatch pendig status. And do nothing if it is called from Dispatch pending status.

ena_dsp Enable Dispatching

【Format】

ER ercd = ena_dsp( ) ;

【Parameter】

No

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

There is no Error that should be mentioned specially.

【Call Context】

Task	Possible
Time Event Handler	Impossible
Interrupt Service Routine	Impossible

【Explanation】

Change to Dispatch permission status. And do nothing if it is called from Dispatch permission status.

sns_ctx Reference Contexts

【Format】

BOOL state = sns_ctx( ) ;

【Parameter】

No

【Return value】

BOOL	state	Context

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Possible

【Explanation】

Return to TRUE if it is called from Non-Task Context, and return to FALSE if it is called from Task Context.

sns_loc Reference CPU State

【Format】

BOOL state = sns_loc( ) ;

【Parameter】

No

【Return value】

BOOL	state	CPU Lock status

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Possible

【Explanation】

Return to TRUE if System is in CPU Lock status, and return to FALSE if it is in CPU release status.

sns_dsp Reference Dispatching Disabled State

【Format】

BOOL state = sns_dsp( ) ;

【Parameter】

No

【Return value】

BOOL	state	Dispatch Pending status

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Possible

【Explanation】

Return to TRUE if System is in Dispatch Pending status, and return to FALSE if it is in Dispatch permission status.

sns_dpn Reference Dispatch Pending State

【Format】

```
BOOL state = sns_dpn( ) ;
```

【Parameter】

No

【Return value】

BOOL	state	Dispatch reservation status

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Possible

【Explanation】

Return to TRUE if System is in Dispatch reservation status, and return to FALSE for other caes. In other words, return to TRUE if it is either in CPU Lock status, or Dispatch Pending status, or when Interrupt level is higher than Task level.

ref_sys Reference System State

【Format】

ER ercd = ref_sys(T_RSYS *pk_rsys) ;

【Parameter】

T_RSYS*	pk_rsys	Pointer to packet which returns System status

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

Content of pk_rsys(T_RSYS type)

There is no field that should be mentioned specially.

【Error Code】

There is no Error that should be mentioned specially.

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Impossible

【Explanation】

Refer to System status, and return to packet specified by pk_rsys.

【Complement】

At the time of creating this Manual, there is still no information for referring to System status.

5. 7 Interrupt Management Functions

chg_ims Change Interrupt Mask

【Format】

ER ercd = chg_ims(IMASK imask) ;

【Parameter】

IMASK	imask	Interrupt mask after change

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

There is no Error that should be mentioned specially.

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Possible

【Explanation】

Change Interrupt level of processor to value specified by imask. This System Call is depending on processor, so please refer to "Processor dependence part Manual" for more explanation.

get_ims Reference Interrupt Mask

【Format】

ER ercd = get_ims(IMASK *p_imask) ;

【Parameter】

No

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
IMASK	imask	The present Interrupt mask

【Error Code】

There is no Error that should be mentioned specially.

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Possible

【Explanation】

Refer to Interrupt mask of processor and return to imask.

5. 8 System Configuration Management Functions

ref_cfg Reference Configuration Information

【Format】

```
ER ercd = ref_cfg(T_RCFG   *pk_rcfg) ;
```

【Parameter】

T_RCFG*	pk_rcfg	Pointer to packet which returns configuration information

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
Content of pk_rcfg(T_RCFG type)		
UH	tick	Cycle time of Time Tick
UH	tskpri_max	Upper TaskPriority Level
UH	id_max	Maximum ID number

【Error Code】

There is no Error that should be mentioned specially.

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Possible

【Explanation】

Refer to information specified by configuration of System and return to packet specified by pk_rcfg.

In Tick, Cycle time of Time Tick specified as Tick time is returned.

In tskpri_max, upper TaskPriority Level specified as Task Priority Level number is returned.

In id_max, return maximum ID number in ID number used in System.

ref_ver Reference Version Information

【Format】

ER ercd = ref_ver(T_RVER *pk_rver) ;

【Parameter】

T_RVER*	pk_rver	Pointer to packet which returns Version information

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code
Content of pk_rver(T_RVER type)		
UH	maker	Maker code of kernel
UH	prid	Identified number of kernel
UH	spver	Version number of ITRON specification
UH	prver	Version number of kernel
UH	prno[4]	Information management of kernel product

【Error Code】

There is no Error that should be mentioned specially.

【Call Context】

Task	Possible
Time Event Handler	Possible
Interrupt Service Routine	Possible

【Explanation】

Refer to Version information of kernel in use and return to packet specified by pk_rver.

【Complement】

At the time of creating this Manual, maker code has not been acquired yet. Therefore, 0x000 will be returned.

CHAPTER 6 Explanation of standard COM port driver

6. 1 Outline of standard COM port driver

Using method of using COM port in µC3/Compact is regulated, and that driver is called standard COM port driver. Here is an explanation of service call of standard COM port driver.

Service call is only corresponding from Task Context, and it is impossible to use by Dispatch reservation status.

6. 2 Service call of standard COM port driver

ini_com Initialization of COM port

【Format】

ER ercd = ini_com(ID DevID, T_COM_SMOD const *pk_SerialMode) ;

【Parameter】

ID	DevID	ID number of device
T_COM_SMOD const *	pk_SerialMode	Pointer to packet of initial information

Content of pk_SerialMode(T_COM_SMOD type)

UW	baud	Baud rate
UB	blen	Data bit
UB	par	Parity
UB	sbit	Stop bit
UB	flow	Flow control

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(DevIDis incorrect or cannot be used)
E_PAR	ParameterError

【Explanation】

Initialize device specified in DecID by the content of packet of initial information. Definition name of deviceID created by configurator in DevID is used and specified.

In baud, specify Baud rate of serial device.

In blen, specify either of these data bit:

BLEN8	8-bit data length
BLEN7	7-bit data length
BLEN6	6-bit data length
BLEN5	5-bit data length

In par, specify either of these parities:

PAR_NONE	Parity bit invalidity
PAR_EVEN	Even number parity bit validity
PAR_ODD	Odd number parity bit validity

In sbit, specify either of these Stop bit:

SBIT1	1 bit stop
SBIT15	1．5-bit data length
SBIT2	2-bit data length

In flow, specify either of these flow controls:

FLW_NONE	Flow control invalidity
FLW_XON	Software flow control validity
FLW_HARD	Hardware flow control validity

ctr_com Control COM port

【Format】

ER ercd = ctr_com (ID DevID, UH command, TMO tmout) ;

【Parameter】

ID	DevID	ID number of device
UH	command	Control command
TMO	tmout	Specify time-out

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(DevIDis incorrect or cannot be used)
E_TMOUT	Polling failure or time-out
E_PAR	ParameterError

【Explanation】

Control device specified in DecID by the content specified by command. Definition name of device ID created by configurator in DevID is used and specified.

There are following types in Command, and it is possible to specify various command by logical disjunction(OR). In this case, the order will be from the upper command.

RST_COM	0xF800	Reset COM port
CLN_TXBUF	0x8000	Waiting for sending of transmission buffer
RST_BUF	0x6000	Clear sending-receiving buffer
RST_TXBUF	0x4000	Clear sending buffer
RST_RXBUF	0x2000	Clear receiving buffer
STP_COM	0x1800	Prohibit sending, receiving
STP_TX	0x1000	Prohibit sending
STP_RX	0x0800	Prohibit receiving
SND_BRK	0x0400	Sending of break character
STA_COM	0x0300	Permit sending, receiving
STA_TX	0x0200	Permit sending
STA_RX	0x0100	Permit receiving
LOC_TX	0x0080	Transmission lock
LOC_RX	0x0040	Receiving lock

UNL_TX	0x0020	Release transmission lock
UNL_RX	0x0010	Release receiving lock

tmout will specify Time-out time in case of CLN_TXBUF, and sending time in case of SND_BRK. In that case, it will be ignored.

putc_com　　Sending character to COM port

【Format】

ER ercd = **putc_com** (ID DevID, VB chr, TMO tmout) ;

【Parameter】

ID	DevID	ID number of device
VB	chr	Character transmission
TMO	tmout	Specify time-out

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(DevIDis incorrect or cannot be used)
E_TMOUT	Polling failure or time-out

【Explanation】

Send the character transmission chr from device which is specified by DecID. Definition name of deviceID created by configurator in DevID is used and specified.

tmout will specify Time-out time till sending.

puts_com Character string transmission of COM port

【Format】

ER ercd = **puts_com** (ID DevID, VB const *p_schr, UINT *p_scnt, TMO tmout) ;

【Parameter】

ID	DevID	ID number of device
VB const *	schr	Character string transmission
UINT *	scnt	Character number transmission
TMO	tmout	Specify time-out

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(DevIDis incorrect or cannot be used)
E_TMOUT	Polling failure or time-out

【Explanation】

Send the character string transmission schr from device which is specified by DecID and only the character number transmission scnt. Definition name of deviceID created by configurator in DevID is used and specified.

tmout will specify Time-out time till sending.

getc_com　　Receive 1 character from COM port

【Format】

ER ercd = **getc_com**(ID DevID, VB　*p_rbuf, UB　*p_sbuf, TMO tmout) ;

【Parameter】

ID	DevID	ID number of device
VB *	rbuf	Receive character
UB *	sbuf	Sending status
TMO	tmout	Specify time-out

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(DevIDis incorrect or cannot be used)
E_TMOUT	Polling failure or time-out

【Explanation】

Return the received character to rbuf, and return receiving status to sbuf from device specified by DecID. At this time, if receiving status is unnecessary, then specify 0 to p_sbuf. Definition name of deviceID created by configurator in DevID is used and specified.

tmout will specify Time-out time till sending.

gets_com Receive character string from COM port

【Format】

ER ercd = **gets_com**(ID DevID, VB *p_rbuf, UB *p_sbuf, INT eos, UINT *p_rcnt, TMO tmout) ;

【Parameter】

ID	DevID	ID number of device
VB *	rbuf	Array row of receiving character
UB *	sbuf	Array of receiving status
INT	eos	Ending character
UINT	rcnt	Receiving character number
TMO	tmout	Specify time-out

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

【Error Code】

E_ID	Incorrect ID number(DevIDis incorrect or cannot be used)
E_TMOUT	Polling failure or time-out

【Explanation】

From device specified by DecID, return the character which has been received to rbuf, and return receiving status to sbuf. Specify size of storing area of receiving data to rcnt, and return the received character to rcnt. At this time, if receiving status is unnecessary, then specify 0 to p_sbuf. In DevID, definition name of deviceID which is created by configurator is used and specified.

Make the receiving to Successful completion when storing area is filled, or ending character is received, or there is Error in case receiving status is valid.

tmout will specify Time-out time till sending.

ref_com Refer to COM port status

【Format】

ER ercd = ref_com(ID tskid, T_COM_REF *pk_SerialRef) ;

【Parameter】

ID	DevID	ID number of device
T_COM_REF *	pk_SerialRef	COM port status

【Return value】

ER	ercd	Successful completion(E_OK)or Error Code

Content of pk_SerialRef(T_COM_REF type)

UH	rxcnt	Character number which has been received
UH	txcnti	Character number which has not been sent
UH	status	Status

【Error Code】

E_ID	Incorrect ID number(tskidis incorrect or cannot be used)

【Explanation】

Refer to status of device which is specified by DecID, and return to packet specified by pk_SerialRef. Return Character number which has been received in driver to rxcnt, and return Character number which has not been sent to txcnt.

Depending on status, return the following status by value of logical disjunction (OR):

T_COM_EROVB	0x0001	FIFO Overrunning
T_COM_EROR	0x0002	Overrunning Error
T_COM_ERP	0x0004	ParityError
T_COM_ERF	0x0008	Flaming Error
T_COM_BRK	0x0010	Receving Break character
T_COM_TXOFF	0x0020	Transmission XOFF reception
T_COM_RXOFF	0x0040	Receiving XOFF transmission
T_COM_RTS	0x0080	RTS Signal active
T_COM_CTS	0x0100	CTS Signal active
T_COM_DTR	0x0200	DTR Signal active
T_COM_DSR	0x0400	DSR Signal active
T_COM_CD	0x0800	CD Signal active
T_COM_RI	0x1000	RI Signal active
T_COM_ENARX	0x2000	Permitted receiving status
T_COM_ENATX	0x4000	Permitted sending status

CHAPTER 7 Appendix

7. 1 Data type

Data types which have been regulated in µITRON4.0 specification are as following:(excluding data type of packet)

B	8-bit integer with sign
H	16- bit integer with sign
W	32- bit integer with sign
UB	8-bit integer without sign
UH	16-bit integer without sign
UW	32--bit integer without sign
VB	8-bit value of which data type is not decided
VH	16-bit value of which data type is not decided
VW	32-bit value of which data type is not decided
VP	Pointer to the one that data type is not decided
FP	Starting-up number of program(pointer)
INT	Integer in processor with natural size sign
UINT	Integer in processor without natural size sign
BOOL	True/false value(TRUE or FALSE)
FN	Function code(Integer with sign)
ER	Error Code(Integer with sign)
ID	ID number of object(Integer with sign)
ATR	Attribute of object(Integer without sign)
STAT	Status of object(Integer without sign)
MODE	Operation mode of Service call(Integer without sign)
PRI	Priority Level(Integer with sign)
SIZE	Size of memory area(Integer without sign)
TMO	Specify time-out(Integer with sign, time unit is 1 mili-second)
RELTIM	Corresponding time(Integer without sign, time unit is 1 mili-second)

SYSTEM	System time(Integer without sign, time unit is 1 mili-second)
VP_INT	Pointer to the one that data type is not decided or an integer in the processor with the natural size sign
ER_BOOL	Error Code or True/false value
ER_ID	Error Code or ID number(Negative ID number is not expressible)
ER_UINT	Error Code or Integer without sign(Valid bit number of Integer without sign is 1 bit shorter than UINT)
FLGPTN	Bit pattern of Eventflag(Integer without sign)
T_MSG	Message header to Mailbox
INTNO	Interrupt number
IMASK	Interrupt mask

【Complement】

INT,UINT,VP_INT,FLGPTN are depending on processor, so please refer to "Processor dependence part mounting Manual" for more detail.

7. 2 Form of packet

(1)Task management functions

Packet form of Task status

```
typedef struct t_rtsk {
        STAT        tskstat ;       /* Taskstatus */
        PRI         tskpri ;        /* The Current Priority Level of Task */
        PRI         tskbpri ;       /* Base Priority Level of Task */
        STAT        tskwait ;       /* Waiting factor */
        ID          wobjid ;        /* ID number of Object for waiting*/
        TMO         lefttmo         /* Time till time-out */
        UINT        actcnt          /* Start-up request queuing number */
        UINT        wupcnt          /* Get-up request queuing number */
        UINT        suscnt          /* Waiting control request nest number */
} T_RTSK ;
```

Packet form of Task status(simple edition)

```
typedef struct t_rtst {
        STAT        tskstat ;       /* Task status */
        STAT        tskwait ;       /* Waiting factor */
} T_RTST ;
```

(2)Synchronization and Communication Functions

Packet form of Semaphore status

```
typedef struct t_rsem {
        ID          wtskid ;        /* ID number of Task in the beginning of waiting
                                       queue of Semaphore*/

        UINT        semcnt ;        /* The present resource number of  Semaphore
                                       */
} T_RSEM ;
```

Packet form of Eventflag status

```
typedef struct t_rflg {
        ID          wtskid ;        /* ID number of Task in the beginning of waiting
                                       queue of Eventflag*/

        FLGPTN      flgptn ;        /* The present bit pattern of Eventflag */
} T_RFLG ;
```

Packet form of Data Queues status

```
typedef struct t_rdtq {
        ID          stskid      /* ID number of Task in the beginning of
                                   transmission waiting queue of Data Queues*/

        ID          rtskid      /* ID number of Task in the beginning of
                                   receiving-waiting queue of Data Queues*/

        UINT        sdtqcnt     /* Number of data in Data Queues */
} T_RDTQ ;
```

Packet form of Mailbox status

```
typedef struct t_rmbx {
        ID          wtskid ;    /* ID number of Task in the beginning of waiting
                                   queue */
        T_MSG*      pk_msg ;    /* The beginning number of message packet in
                                   the beginning of Message cue*/

} T_RMBX ;
```

(3)Memory Pool Management Functions

Packet form of status of Fixed-Sized memory pool

```
typedef struct t_rmpf {
        ID          wtskid ;    /* ID number of Task in the beginning of waiting
                                   queue of Fixed-Sized memory pool*/

        UINT        fblkcnt ;   /* Empty memory block number (number) of
                                   Fixed-Sized memory pool*/

} T_RMPF ;
```

(4)Time Management Functions

Cycle Handlerstatus Packet form of

```
typedef struct t_rcyc {
        STAT        cycstat ;   /*Operation status of Cycle Handler */
        RELTIM      lefttim ;   /* Time till next starting-up of Cycle Handler*/

} T_RCYC ;
```

(5)System State Management Functions

Packet form of System status

```
typedef struct t_rsys {
        /* No field */
} T_RSYS ;
```

(6)System Configuration Management Functions

Packet form of Configuration information

```
typedef struct t_rcfg {
        UH      tick        Cycle time of Time Tick
        UH      tskpri_max  Upper limit of Task Priority Level
        UH      id_max      Maximum ID number
} T_RCFG ;
```

Version information Packet form of

```
typedef struct t_rver {
        UH      maker ;     /* Maker code of kernel */
        UH      prid ;      /* Identified number of kernel */
        UH      spver ;     /*Version number of ITRON specification*/
        UH      prver ;     /*Version number of kernel */
        UH      prno[4] ;   /*Management information of kernel product*/
} T_RVER ;
```

7. 3 Constant and macro

(1)General

NULL	0	Invalid pointer
TRUE	1	True
FALSE	0	False
E_OK	0	Successful completion

(2)Specify time-out

TMO_POL	0	Polling
TMO_FEVR	-1	Permanent wait
TMO_NBLK	-2	Non-blocking

(3)Operation mode of Service call

TWF_ANDW	0x00	AND waiting of Eventflag
TWF_ORW	0x01	OR waiting of Eventflag

(4)Object status

TTS_RUN	0x01	Execution status
TTS_RDY	0x02	possible execution status
TTS_WAI	0x04	Waiting status
TTS_SUS	0x08	Compulsive waiting status
TTS_WAS	0x0c	Double waiting status
TTS_DMT	0x10	Dormant status
TTW_SLP	0x0001	Get-up waiting status
TTW_DLY	0x0002	Time-passing waiting status
TTW_SEM	0x0004	Status of waiting for acquiring Semaphore resource
TTW_FLG	0x0008	Waiting status of Eventflag
TTW_SDTQ	0x0010	Waiting status of transmission to Data Queues
TTW_RDTQ	0x0020	Waiting status of receiving from Data Queues
TTW_MBX	0x0040	Waiting status of receiving from Mailbox
TTW_MPF	0x2000	Acquisition waiting status of Fixed-Sized memory block
TCYC_STP	0x00	Cycle Handler is in non-operation
TCYC_STA	0x01	Cycle Handler is in operation

(5)Other constants

TSK_SELF	0	Specify local Task
TSK_NONE	0	No appropriateTask
TPRI_SELF	0	Specify Base Priority Level of local Task
TPRI_INI	0	Specify Priority Level when starting up Task

7．4 Composition constant and macro

(1)Scope of Priority Level

TMIN_TPRI	Minimum value of TaskPriority Level(＝1)

(2)Version information

TKERNEL_MAKER	Maker code of kernel
TKERNEL_PRID	Identified number of kernel
TKERNEL_SPVER	Version number of ITRON specification
TKERNEL_PRVER	Version number of kernel

(3)The maximum value of queuing/frequency of nest

TMAX_ACTCNT	Maximum value of Task’s Start-up request queuing number
TMAX_WUPCNT	Maximum value of Task’s get-up request queuing number

(4)Bit number of bit pattern

TBIT_FLGPTN	Bit number of Eventflag

(5)Others

TMAX_MAXSEM	Maximum value of Semaphore’s maximum resource number

7. 5 List of Error Code

E_SYS	-5	0xFFFFFFFB	SystemError
E_NOSPT	-9	0xFFFFFFF7	Non-support function
E_RSFN	-10	0xFFFFFFF6	Reservation function code
E_RSATR	-11	0xFFFFFFF5	Reservation Attribute
E_PAR	-17	0xFFFFFFEF	ParameterError
E_ID	-18	0xFFFFFFEE	Incorrect ID number
E_CTX	-25	0xFFFFFFE7	ContextError
E_MACV	-26	0xFFFFFFE6	Memory access violation
E_OACV	-27	0xFFFFFFE5	Object access violation
E_ILUSE	-28	0xFFFFFFE4	Incorrect use of Service call
E_NOMEM	-33	0xFFFFFFDF	Memory shortage
E_NOID	-34	0xFFFFFFDE	ID number shortage
E_OBJ	-41	0xFFFFFFD7	Object status error
E_NOEXS	-42	0xFFFFFFD6	Object which has not been created
E_QOVR	-43	0xFFFFFFD5	Queuing overflow
E_RLWAI	-49	0xFFFFFFCF	Compulsive release of waiting status
E_TMOUT	-50	0xFFFFFFCE	Polling failure or time-out
E_DLT	-51	0xFFFFFFCD	Delete status of waiting Object
E_CLS	-52	0xFFFFFFCC	Change status of waiting Object
E_WBLK	-57	0xFFFFFFC7	Non-blocking reception
E_BOVR	-58	0xFFFFFFC6	Buffer overflow

7. 6 List of System Call

System Call name	Task	Time Event Handler	Interrupt Service Routine
A) Task Management Functions			
act_tsk／iact_tsk	○	○	○
can_act	○	○	×
sta_tsk	○	○	○
ext_tsk	○	×	×
ter_tsk	○	×	×
chg_pri	○	○	×
get_pri	○	○	×
ref_tsk	○	○	×
ref_tst	○	○	×
B) Task Dependent Synchronization Functions			
slp_tsk	○	×	×
tslp_tsk	○	×	×
wup_tsk／iwup_tsk	○	○	○
can_wup	○	○	×
rel_wai／irel_wai	○	○	○
dly_tsk	○	×	×
C) Synchronization and Communication Functions(Semaphores)			
sig_sem／isig_sem	○	○	○
wai_sem	○	×	×
pol_sem	○	○	×
twai_sem	○	×	×
ref_sem	○	○	×
D) Synchronization and Communication Functions(Eventflags)			
set_flg／iset_flg	○	○	○
clr_flg	○	○	×
wai_flg	○	×	×
pol_flg	○	○	×
twai_flg	○	×	×
ref_flg	○	○	×

System Call name	Task	Time Event Handler	Interrupt Service Routine
E) Synchronization and Communication Functions(Data Queues)			
snd_dtq	○	×	×
psnd_dtq／ipsnd_dtq	○	○	○
tsnd_dtq	○	×	×
fsnd_dtq／ifsnd_dtq	○	○	○
rcv_dtq	○	○	×
prcv_dtq	○	○	×
trcv_dtq	○	×	×
ref_dtq	○	○	×
F) Synchronization and Communication Functions(Mailboxes)			
snd_mbx	○	○	×
rcv_mbx	○	×	×
prcv_mbx	○	○	×
trcv_mbx	○	×	×
ref_mbx	○	○	×
G) Memory Pool Management Functions (Fixed-Sized Memory Pools)			
get_mpf	○	×	×
pget_mpf	○	○	×
tget_mpf	○	×	×
rel_mpf	○	○	×
ref_mpf	○	○	×
H) Time Management Functions(System Time Management)			
set_tim	○	○	×
get_tim	○	○	×
isig_tim	×	×	○
I)Time Management Functions (Cycle Handlers)			
sta_cyc	○	○	×
stp_cyc	○	○	×
ref_cyc	○	○	×

System Call name	Task	Time Event Handler	Interrupt Service Routine
J) System State Management Functions			
rot_rdq／irot_rdq	○	○	○
get_tid／iget_tid	○	○	○
loc_cpu／iloc_cpu	○	○	○
unl_cpu／iunl_cpu	○	○	○
dis_dsp	○	×	×
ena_dsp	○	×	×
sns_ctx	○	○	○
sns_loc	○	○	○
sns_dsp	○	○	○
sns_dpn	○	○	○
ref_sys	○	○	×
K) Interrupt Management Functions			
chg_ims	○	○	○
get_ims	○	○	○
L) System Configuration Management Functions			
ref_cfg	○	○	○
ref_ver	○	○	○

○：Possible use

×：Impossible use

Index

L

M

N

O

P

Q

R

S

T

U

W

µC3/Compact Users Guide

2008 May	1st Edition
2008 August	2nd Edition
2009 March	3rd Edition
2010 June	4th Edition
2012 October	5th Edition

eForce Co., Ltd. http://www.eforce.co.jp/

Contact us : info@eforce.co.jp