EAS470_OS_UserManual

经纬恒润

JINGWEI HIRIAN

文件状态： [ ] 草稿 [√] 正式发布 [ ] 正在修改 Document status: [ ] Draft [√] Released [ ] Modifying
文件起草分工： 1. 杜琳如 2. 赵彦安 Editor: 1. Linru.Du 2. Yanan.zhao1

Change History

Version	Change description	Author	Audit	Approval
1.0	初始版本 Initial version	杜琳如 Linru.Du 2021-5-10	陈宁 Ning.Chen 2021-5-13	于雅琪 Yaqi.Yu 2021-5-30
1.1	更新第三章 Update Chapter 3	杜琳如 Linru.Du 2021-6-3	陈宁 Ning.Chen 2021-6-3	于雅琪 Yaqi.Yu 2021-6-3
1.2	更新第三章的 SC2 部分 Update SC2 SC3	赵彦安 Yanan.zhao1 2021-6-22	陈宁 Ning.Chen 2021-6-22	于雅琪 Yaqi.Yu 2021-6-22
1.3	更新 3.6 和 5.2 节 Update Chapter 3.6, 5.2	赵彦安 Yanan.zhao1 2021-9-9	陈宁 Ning.Chen 2021-9-9	于雅琪 Yaqi.Yu 2021-9-9
1.4	更新第八章内容 Update information of chapter 8	赵彦安 Yanan.zhao1 2021-12-30	陈宁 Ning.Chen 2021-12-30	于雅琪 Yaqi.Yu 2021-12-30
1.5	更新 6.7 章节内容 Update information of chapter 6.7	续开轩 Kaixuan.xu 2023-03-17	赵彦安 Yanan.zhao1 2023-03-21	赵彦安 Yanan.zhao1 2023-03-21
1.6	更新 6.8 章节内容 Update information of chapter 6.8	续开轩 Kaixuan.xu 2023-03-17	赵彦安 Yanan.zhao1 2023-03-21	赵彦安 Yanan.zhao1 2023-03-21
1.7	更新 5.5 和 6.6 章节内容 Update information of chapter 5.5 and 6.6	钱不凡 Bufan.qian 2023-07-10	赵彦安 Yanan.zhao1 2023-07-10	赵彦安 Yanan.zhao1 2023-07-10
1.8	更新 6.3.1 和 6.3.2 和 6.4 章节内容	续开轩 Kaixuan.xu	赵彦安 Yanan.zhao1	赵彦安 Yanan.zhao1

	Update information of chapter 6.3.1 and 6.3.2 and 6.4	2023-07-11	2023-07-11	2023-07-11
1.9	更新 3.3.3 和 6.9 章节内容 Update information of chapter 3.3.3 and 6.9	刘蕊 Rui.liu3 2023-08-25	赵彦安 Yanan.zhao1 2023-09-01	赵彦安 Yanan.zhao1 2023-09-01
1.10	更新 5.5.2 和 5.10 和 5.11 章节内容 Update information of chapter 5.5.2 and 5.10 and 5.11	王宇晗 Yuhan.wang 2023-09-01	赵彦安 Yanan.zhao1 2023-09-05	赵彦安 Yanan.zhao1 2023-09-05
1.11	更新 5.2 、5.11、5.12 和 5.13 章节内容 Update information of chapter 5.2 and 5.11 and 5.12 and 5.13	王宇晗 Yuhan.wang 2023-09-05	赵彦安 Yanan.zhao1 2023-09-05	赵彦安 Yanan.zhao1 2023-09-05
1.12	更新 5.1 、6.1 和 6.5 章节内容 新增 6.10 章节 Update information of chapter 5.1, 6.1 and 6.5 Add chapter 6.10	钱不凡 Bufan.qian 2023-09-19	赵彦安 Yanan.zhao1 2023-09-22	赵彦安 Yanan.zhao1 2023-09-22
1.13	更新 5.2.1 章节中图 14，修改拼写错误 Update figure 14 in chapter 5.2.1, correct spelling errors	马鹤 He.ma1 2024-04-16	赵彦安 Yanan.zhao1 2024-04-17	赵彦安 Yanan.zhao1 2024-04-17
1.14	更新第六章，五章截图为 R21-11 Update figure in chapter 5，6, to R21-11	续开轩 Kaixuan.xu 2024-05-27	赵彦安 Yanan.zhao1 2024-05-27	赵彦安 Yanan.zhao1 2024-05-27
1.15	新增 4.1.38、4.1.39 和 4.1.40 章节。 Add chapter 4.1.38, 4.1.39 and 4.1.40.	马鹤 He.ma1 2024-05-31	赵彦安 Yanan.zhao1 2024-05-31	赵彦安 Yanan.zhao1 2024-05-31

Contents

Keep this line as the end of content.

1 Introduction

1.1 Purpose

本文是 OS 产品通用部分的使用说明，包括如下内容：

This document is user manual of general part of OS, including:

1) Introduction of module
   - Features, software architecture and files
2) Technique Reference
   - State machines
   - Primary job sequence
3) Interface
   - Interfaces between upper and lower layer
4) Configuration
   - Configuration parameters and how to do the configuration
5) Integration
   - How to integrate with user's project
6) Notice
   - Suggestions about modules
7) Examples

1.2 Scope

本文档适用于用户使用和集成阶段。

This document is applicable to the phase of application development and integration.

1.3 Reference Document

Table1-1 Reference Documents

No.	Title	Version
1	OSEK/VDX Operating System	Version 2.2.3
2	Related AUTOSAR specification	V4.7.0
3	EAS470_OS_SafetyManual	V1.4

1.4 Terms and abbreviation

Table1-2 Terms and abbreviation

Name	Description
AUTOSAR	Automotive Open System Architecture
OIL	OSEK Implementation Language

Name	Description
OS	Operating System
OSEK	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug, Open Systems and their interfaces to the Electronics in motor vehicles
VDX	Vehicle Distributed Executive
BSW	Basic Software
EAS	Embedded AUTOSAR Software. Name of Hirain's BSW Product.
MCAL	Microcontroller Abstract Layer
SC	Scalability Class of OS
API	Application Interface
ISR	Interrupt service routine
IOC	Inter-OS-Application-Communicator
MPU	Memory protection unit
C1/C2	Category 1 and Category 2 interrupt. Only C2 interrupt will be processed by OS.
ASR	Short name of AUTOSAR
SchTbl	Schedule Table
Tp	Timing Protection
OS Object	Means Task, ISR, Resource, Schedule Table, Counter, Alarm of OS

2 Overview

2.1 Features

AUTOSAR OS 的主要功能描述如下。

Primary features of AUTOSAR OS are as follows.

SC1:

1) OS start and shutdown
2) ISR
3) Task
4) Event
5) Alarm and Counter
6) Schedule Table
7) Resource
8) Stack Management
9) Error Handling
10) Support multicore
11) Spinlock

SC2:

12) Timing protection

SC3:

13) OS-Application
14) IOC
15) Memory Protection
16) Service Protection

SC4:

Including SC1 ~ SC3.

2.2 Interface diagram

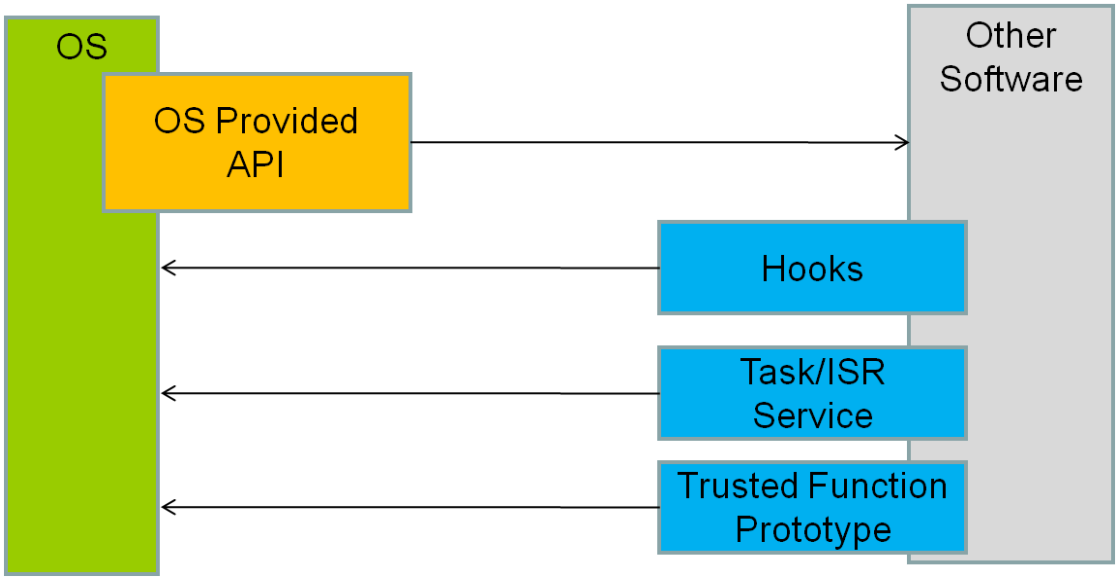


Figure2-1 Interface diagram

OS 提供的 API，参见本文第 4.1 章节的说明。

Hooks，Task/ISR Service，Trusted Function Prototype 均为用户实现，由 OS 调用的接口，具体可参见 4.3 章节的说明。

For OS provided API provided, see the description in Chapter 4.1 of this article.

Hooks, Task/ISR Service, and Trusted Function Prototype are all implemented by users and are called by the OS. For details, please refer to the description in section 4.3.

2.3 File Structure

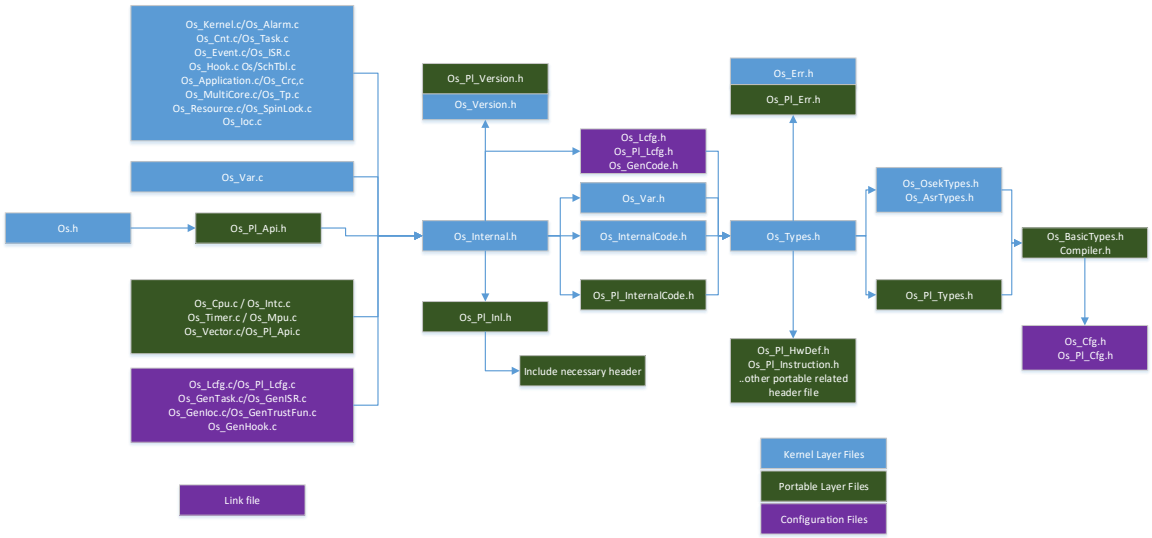


Figure2-2 File Structure

2.3.1 Kernel Files

Kernel files of this module are as follows:

Table2-1 Kernel Layer Non-Generated Source Files

File name	description
Os_Kernel.c	Function implementation of Kernel Module
Os_Alarm.c	Function implementation of Alarm Module
Os_Cnt.c	Function implementation of Counter Module
Os_Task.c	Function implementation of Task Module
Os_Event.c	Function implementation of Event Module
Os_ISR.c	Function implementation of ISR Module
Os_Hook.c	Function implementation of Hook Module
Os_SchTbl.c	Function implementation of SchTbl Module
Os_Application.c	Function implementation of Application Module
Os_Crc.c	Function implementation of CRC Module
Os_Multicore.c	Function implementation of Multicore Module
Os_Tp.c	Function implementation of Tp Module
Os_Resource.c	Function implementation of Resource Module
Os_Spinlock.c	Function implementation of Spinlock Module
Os_Ioc.c	Function implementation of IOC Module
Os_Var.c	Define all global variables of OS
Os_PeripheralArea.c	Function implementation of Peripheral Module

Table2-2 Kernel Layer Non-Generated Header Files

File name	description
Os.h	User only need to include this header file of OS
Os_Internal.h	OS modules only need to include this header file
Os_InternalCode.h	Declare all functions of kernel layer modules
Os_Types.h	Define all non-standard types. Define all global macros, except error related macros.
Os_Err.h	Define all error code and API ID of kernel layer.
Os_OsekTypes.h	Define all OSEK standard types.
Os_AsrTypes.h	Define all AUTOSAR standard types.
Os_Version.h	Define version information of kernel layer.
Os_Var.h	Declare all global variables of OS.

Table2-3 Portable Layer Non-Generated Source Files

File	Description
Os_Cpu.c	Function implementation of Cpu Module

Os_Intc.c	Function implementation of Intc Module
Os_Timer.c	Function implementation of Timer Module
Os_Mpu.c	Function implementation of Mpu Module
Os_Vector.c	定义中断、Trap 向量表中固定的部分。例如，通常 Trap 表是固定的，与用户配置无关。 Define the fixed part of interrupt and trap vector table. For example, usually the Trap table is fixed and has nothing to do with user configuration.
Os_Pl_Api.c	Define all OS API when memory protection is enabled.

Table2-4 Portable Layer Non-Generated Header Files

File	Description
Os_Pl_Api.h	Declare all OS API.
Os_Pl_Inl.h	All inline function implementation of portable layer.
Os_Pl_InternalCode.h	Declare all functions of portable layer modules
Os_Pl_Err.h	Define all error code and API ID of portable layer.
Os_Pl_Types.h	Define non-standard types related to platform. Define global macros related to platform.
Os_BasicTypes.h	Define all basic types. Usually, these types are hardware and compiler related
Os_Pl_HwDef.h	定义硬件相关的宏，例如核的最大数量，寄存器地址等。 Define hardware-related macros, such as the maximum number of cores, register addresses, etc.
Os_Pl_Instruction.h	定义汇编指令相关宏，例如开关全局中断，切换栈指针等。通常，更换编译器时，这些宏需要修改。 Define assembly instructions related macros, such as switch global interrupt, switch stack pointer, etc. Usually, when changing the compiler, these macros need to be modified.
Os_Pl_Version.h	Define version information of portable layer.

2.3.2 Configuration Files

Configuration files of this module are as follows:

Table2-5 Generated Files

File name	description
Os_Cfg.h	Definition and declaration of all configuration parameters of OS kernel layer.

File name	description
Os_Lcfg.c	
Os_Lcfg.h	
Os_Pl_Cfg.h	Definition and declaration of all configuration parameters of OS portable layer.
Os_Pl_Lcfg.c	
Os_Pl_Lcfg.h	
Os_GenTask.c	Generate all task functions. Including IDLE tasks.
Os_GenIsr.c	Generate all interrupt functions. If necessary, generate interrupt registration related content, such as interrupt vector table.
Os_GenHook.c	Generate all hook functions. Such as ErrorHook.
Os_GenIoc.c	Generate all IOC functions. The specific implementation of the IOC function depends on the user's configuration.
Os_GenTrustFun.c	Generate all trusted functions. The specific implementation of the trusted function depends on the user's configuration.
Os_GenCode.h	Declare the content generated above.
Link file	Include some memory sections. User should add memory sections to their own ink file.

3 Technique Reference

3.1 General

3.1.1 Startup

OS 操作系统的启动是通过调用“StartOS()”函数来实现。在标准的 AUTOSAR 项目中，StartOS 函数由 EcuM 模块调用。

The start of the OS operating system is realized by calling the "StartOS()" function. In a standard AUTOSAR project, the StartOS function is called by the EcuM module.

[Notice]

在调用 StartOS()之前，用户应确保一些必要的初始化步骤已经完成。具体请参见《EAS470_OS_SafetyManual》第 3.2 章节。

Before calling StartOS(), the user should ensure that some necessary initialization steps have been completed. For details, please refer to Chapter 3.2 of "EAS470_OS_SafetyManual".

如果用户使能了相关 Hook，StartOS 会在所有初始化完成之后，启动系统定时器、触发第一次任务调度之前，调用用户的 Hook，调用顺序如下。

（1）调用 OsStartupHook。

（2）调用各个 OS-Application 的 OsAppStartupHook。

If the user enables the related Hook, StartOS will call the user's Hook after all initializations are completed, before starting the system timer, and before triggering the first task scheduling. The calling sequence is as follows.

(1) Call OsStartupHook.

(2) Call the OsAppStartupHook of each OS-Application.

StartOS()会对 OS 内部各组件进行初始化，激活自启动任务、报警或调度表。StartOS()的最后，会启动系统定时器，并触发第一次任务调度。

StartOS() initializes various components of OS, and activates auto-start tasks, alarms or schedule tables. At the end of StartOS(), the system timer is started and the first task scheduling is triggered.

调用 StartOS 的示例如下图。

An example of calling StartOS is shown below.

```
#include“OS.h”
void main(void)
{
    /* usercode */
    StartOS(OSDEFAULTAPPMODE);
}
```

Figure3-1　Example of StartOS

3.1.2 Shutdown

系统关闭通过调用“ShutdownOS()”函数实现。“ShutdownOS()”函数会禁能所有中断,停止系统定时器运行，最终进入死循环。

The shutdown of operation system is achieved by calling the "ShutdownOS()" function. The function will disable all interrupts, stop timers, then enter endless loop.

如果用户使能了相关 Hook，ShutdownOS 会在进入结尾处，进入死循环之前，调用用户的 Hook，调用顺序如下。

（1）调用各个 OS-Application 的 OsAppShutdownHook。

（2）调用 OsShutdownHook。

If the user enables the related Hook, ShutdownOS will call the user's Hook at the end and before entering the endless loop. The calling sequence is as follows.

(1) Call the OsAppShutdownHook of each OS-Application.

(2) Call OsShutdownHook.

3.1.3 Auto-Start Features

3.1.3.1 Auto-Start Objects

OS 支持自启动的 Task、Alarm 和 Schedule Table。自启动的对象有如下特性。

The OS supports auto-start Task, Alarm and Schedule Table. The auto-start object has the following characteristics.

1) 在系统启动阶段，自启动的 Task、Alarm 和 Schedule Table 会在 StartOS()中被启动；
2) 自启动 Alarm 需要用户配置其类型（绝对报警，相对报警）、偏移时间和周期。

1) During the system startup phase, the auto-start Task, Alarm and Schedule Table will be started in StartOS();

2) Auto-start Alarm requires the user to configure its type (absolute alarm, relative alarm), offset time and period

3.1.3.2 Application Mode

OSEK 操作系统标准中设立了应用模式的概念。用户可以在不同的应用模式下运行不同的应用程序，从而在不同的条件下实现不同需求。OS 同样实现了 OSEK 操作系统标准的这一功能，系统支持多种应用模式共存，并提供配置选项供用户在系统配置阶段建立和选择所需的应用模式。系统中所有的应用模式必须在系统配置阶段建立完成，每个模式下需要自启动的 Task、Alarm 和 Schedule Table 也必须在系统配置阶段明确指定。

The concept of application mode is established in the OSEK operating system standard. Users can run different applications in different application modes to achieve different requirements under different conditions. The OS also implements this function of the OSEK operating system standard. The system supports the coexistence of multiple application modes and provides configuration options for users to establish and select the required application mode during the system configuration phase. All application modes in the system must be established in the system configuration phase, and the Task, Alarm, and Schedule Table that need to be started automatically in each mode must also be clearly specified in the system configuration phase.

在系统启动阶段，用户程序不能调用任何除“StartOS()”以外的操作系统服务函数。在调用“StartOS()”之前，用户程序必须指定操作系统启动后将采用的应用模式，将应用模式编号作为参数传递给“StartOS()”。

During the system startup phase, the user cannot call any OS service function other than "StartOS()". Before calling "StartOS()", the user must specify the application mode to be used，passing the application mode ID as a parameter to "StartOS()".

用户可配置不同的 AppMode，在不同的 AppMode 中，支持不同的自启动的对象。可参见下图示例。

Users can configure different AppModes, and in different AppModes, different auto-start objects are supported. See the example below.

	Default	APP_MODE1
Task1		√
Task2	√	
Task3	√	
Alarm1	√	
Alarm2		√
Alarm3	√	

Figure3-2 Auto-Start Object and AppMode

在启动 OS 时，需要输入本次启动的 AppMode。示例如下。

When starting the OS, need to input the target AppMode. The example is as follows

```
Main()
{
   /* User code before OS start-up */
  ...........
  ...........
   /* Jump to OS start-up */
   StartOS(OSDEFAULTAPPMODE);
}
```

Figure3-3 Example of startup of OS

[Notice]

OS 中的所有符合类都支持应用模式的设定。应用模式只对系统启动函数有作用。一旦操作系统启动，应用模式就不能再更换，也就是说在系统运行过程中动态切换应用模式是不允许的。

All conforming classes in OS support application mode settings. The application mode only has an effect on the system start function. Once the operating system is started, the application mode can no longer be changed. In other words, it is not allowed to switch application mode when system is running.

3.1.4 Task

嵌入式软件可以根据对实时性的要求分解成多个部分来执行，其中的每一个部分被称作一个任务。任务为用户提供了一个函数执行的框架，并由调度器来决定它们之间的

执行顺序。

Embedded software can be divided into multiple parts according to the requirements of real-time performance, each of which is called a task. Tasks provide users with a framework for function execution, the scheduler determines the order of execution between them.

The framework of the task in OS is shown in figure below.

```
TASK(Task1)
{
    /* User Code */
    TerminateTask();
}
```

Figure3-4 Framework of task in OS

3.1.4.1 Task Priority

OS 中任务的优先级是静态配置的，并且在系统运行过程中不会改变。任务优先级配置时，号码越大所标识的优先级越高，优先级 0 为系统的最低优先级。用户应按照这一原则制定和分配任务优先级。

The priority of the tasks in OS is statically configured and will not change during the system operation. The value 0 is defined as the lowest priority of a task. Accordingly, bigger numbers define higher priorities. Users should configure priorities of tasks according to this principle.

高优先级的任务先执行，相同优先级的任务遵循 FIFO（先进先出）的规则。

High priority tasks are executed first, and tasks with the same priority follow the FIFO (first in first out) rule.

当 OS 符合类为 BCC1 或 ECC1 时，不支持任务的多次激活，不支持为不同的任务分配相同的优先级。因此每个优先级上有且只有一个任务。

When the OS conformance class is BCC1 or ECC1, multiple activations of tasks are not supported, and different tasks assigned to same priority are not supported. Therefore, there is one and only one task per priority.

当 OS 符合类为 BCC2 或 ECC2 时，支持任务的多次激活，支持为不同的任务分配相同的优先级。此时 OS 会为每个优先级创建任务队列，队列深度与该优先级上任务的个数和最大激活次数相关。

例如，Task1, Task2, Task3 的优先级均为 5，Task1 和 Task3 的最大激活次数为 1，Task2 的最大激活次数为 10。则 OS 为优先级 5 创建的任务队列的深度为 12。

所以正常情况下，不会出现队列溢出的情况。

When the OS conformance class is BCC2 or ECC2, multiple activations of tasks are supported, and different tasks assigned to the same priority are supported. At this time, the OS will create a task queue for each priority, and the depth of the queue is related to the number of tasks on that priority and the maximum number of activations.

For example, the priority of Task1, Task2, and Task3 are all 5, the maximum number of activations of Task1 and Task3 is 1, and the maximum number of activations of Task2 is 10. Then the depth of the task queue created by the OS for priority 5 is 12.

Therefore, under normal circumstances, there will be no queue overflow.

下图举例说明了基于优先级的任务调度顺序。

Based on an example, the following figure illustrates the priority-based task scheduling sequence.

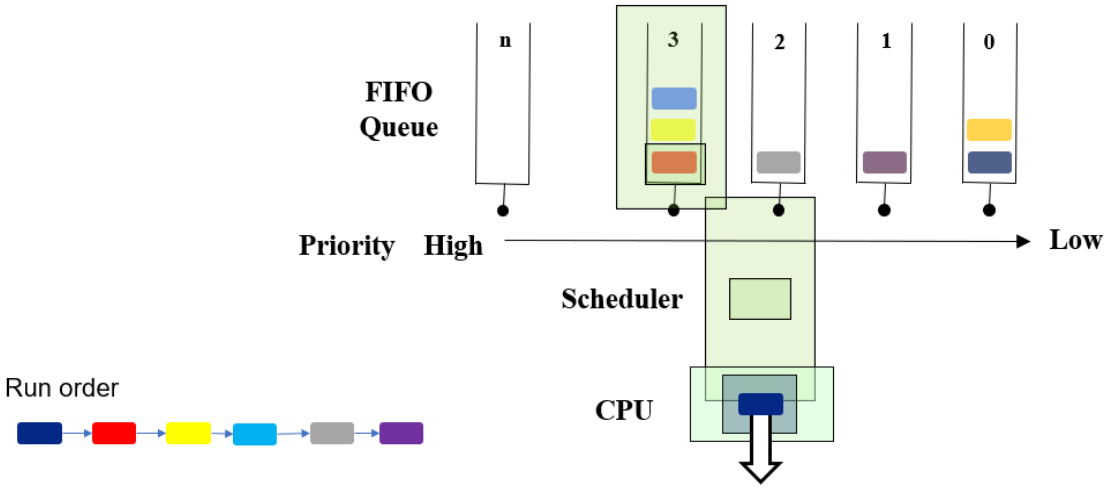


Figure3-5 Examples of Task Scheduling

[Notice]

用户在给任务分配优先级时需要注意以下两点：

Users should note the following two points when assigning priority to tasks:

1)当扩展任务从等待状态（WAITING）进入到就绪状态（READY）时，该任务将会被插入到对应优先级就绪队列的队尾；

A task being released from the waiting state is treated like the last (newest) task in the ready queue of its priority;

2)若位于某个优先级就绪队列队首的任务开始执行，该任务将保持它在该队列中的位置直到该任务完成运行，终止自己。所以如果该任务在执行过程中被其它高优先级的

任务抢占，一旦该任务的优先级恢复成系统中的最高优先级，被抢占的任务仍然是该队列中最先被执行的任务，它将从被抢占的位置继续执行。

If a task at the header of the priority ready queue is executing, the task will maintain the position in the queue until the task completes and terminates itself. After the task is preempted in the process of execution, once the task priority becomes the highest again, the preempted task is still the first task is executed in the queue, it will continue at the location where it was preempted.

3.1.4.2 Scheduling Policy

3.1.4.2.1 Full preemptive scheduling

在完全抢占调度策略下，所有 OS 的任务均为可抢占的。高优先级任务会抢占低优先级任务。

Under the full preemption scheduling strategy, all OS tasks are preemptible. High-priority tasks will preempt low-priority tasks.

3.1.4.2.2 Non preemptive scheduling

在完全非抢占调度策略下，所有 OS 的任务均为不可抢占的。任何任务在运行时，都不会被其他任务抢占。

Under the completely non-preemptive scheduling strategy, all OS tasks are non-preemptible. When any task is running, it will not be preempted by any other tasks.

3.1.4.2.3 Mixed preemptive scheduling

在混合调度策略下，用户可配置任务是否可抢占。

Under the mixed scheduling strategy, users can configure whether tasks can be preempted.

[Notice]在不可抢占的任务中，调用 Schedule 函数会强制触发调度。如果此时有更高优先级的任务在 Ready 状态，则会抢占当前任务。In non-preemptible tasks, calling the Schedule function will force the scheduling to be triggered. If there is a higher priority task in the Ready state at this time, the current task will be preempted.

3.1.4.3 Activate a task

OS 中所有的任务都是静态配置的，在系统启动时就已经存在了。

All tasks in the OS are statically configured, the tasks are already existing when the system is started.

用户在系统运行过程想要执行某个任务只需要将这个任务激活。激活一个任务要使

用系统服务函数“ActivateTask()”或“ChainTask()”，任务也可以配置成自启动的类型，在系统启动阶段由操作系统自动将其激活。

The user can perform a task by activating the task while the system is running by using the operating system services ActivateTask or ChainTask. Tasks can also be configured to auto-start, which are automatically activated by the operating system during system start-up.

3.1.4.3.1 Activate a task by ActivateTask()

The example in figure below shows that Task1 and Task2 exist in the system. Task1 is implemented by calling "ActivateTask()" to activate Task2.

```
TASK(Task1)
{
    /* Activating Task2 */
    ActivateTask(Task2);
    TerminateTask();
}
```

Figure3-6 Activate a task by "ActivateTask()"

系统服务函数“ActivateTask()”将处于挂起状态的任务转换为就绪状态。

The system service function "ActivateTask()" converts the task in the suspended state to the ready state.

如果在某任务运行过程中调用了 ActivateTask()，且当前任务为可抢占任务，则 ActivateTask()会基于优先级进行任务调度。如果当前任务为不可抢占任务，则 ActivateTask()不会触发调度。

If ActivateTask() is called during the running of a task, and the current task is a preemptible task, ActivateTask() will perform task schedule based on the priority. If the current task is a non-preemptible task, ActivateTask() will not trigger the scheduling.

如果在 C2 中断中调用了 ActivateTask()，OS 会在当前中断运行结束后，执行调度。

If ActivateTask() is called in the C2 interrupt, the OS will perform scheduling when the current interrupt end.

3.1.4.3.2 Activate a task by ChainTask()

The example in figure below shows that Task1 and Task2 exist in the system. Task1 is implemented by calling "ChainTask()" to activate Task2.

```
TASK(Task1)
{
    /* Activating Task2 */
    ChainTask(Task2);
}
```

Figure3-7　Activate a task by "ChainTask()"

系统服务函数“ChainTask()”会终止当前正在运行的任务，同时激活一个新任务，然后会触发一次系统调度。

The system service function "ChainTask()" will terminate the currently running task and activate a new task, then trigger a system schedule.

[Notice]

如果新激活的任务与调用“ChainTask()”的任务是同一个任务，这种情况不属于任务的多次激活。

If the new activated task is the same task as the one calling "ChainTask()", this situation does not belong to multiple activations of the task.

3.1.4.4 Terminate a task

用户需要调用系统服务函数“TerminateTask()”或“ChainTask()”终结当前运行的任务。示例如下。

The user needs to call the system service function "TerminateTask()" or "ChainTask()" to terminate the currently running task. The example is as follows.

```
TASK(Task1)
{
    /* User code*/
    TerminateTask();
}
```

Figure3-8　Terminate a task by "TerminateTask()"

```
TASK(Task1)
{
    /* Terminate Task1 and  successively activate Task2*/
    ChainTask(Task2);
}
```

Figure3-9　Terminate a task by "ChainTask()"

用户在调用“TerminateTask()”或“ChainTask()”终结任务之前，应释放所有被占用的资源和自旋锁，恢复被禁能的中断状态。

Before calling "TerminateTask()" or "ChainTask()" to terminate the task, the user should release all occupied resources and spinlocks, and restore the disabled interrupt state.

[Notice]

在 AUTOSAR 的服务保护中，增加了如下特性：

如果用户没有终结任务，OS 会自动终结任务，并自动释放资源、自旋锁，自动恢复中断状态。

In the service protection of AUTOSAR, the following features have been added:

If the user does not terminate the task, the OS will automatically terminate the task, automatically release the resource and spinlock, and automatically restore the interrupted state.

3.1.4.5 Multiple Task Activation

如果操作系统的符合类被配置成 BCC2 或者 ECC2，那么系统将支持任务的多次激活。系统中只有基本任务（Basic Task）能够进行多次激活，扩展任务（Extended Task）不能进行多次激活。

Depending on the conformance class of BCC2 or ECC2, the system will support multiple activation of task. Multiple task activation for extended tasks is not supported, only basic tasks support multiple activation.

3.1.5 Counter

OS 中有两类计数器：SystemCounter 和 UserDefinedCounter。SystemCounter 指的是操作系统所使用的系统计数器；UserDefinedCounter 指的是用户自行设定的、通常用于应用软件的计数器。

There are two types of counters in OS: SystemCounter and UserDefinedCounter. SystemCounter refers to the system timer used by the operating system. UserDefinedCounter is

a user-defined counter that is typically used for applications.

3.1.5.1 SystemCounter

OS 中始终存在一个默认的计数器 SystemCounter，该计数器只能通过 SystemTick 进行触发，用户不可修改。

SystemCounter is a default counter in OS, which can only be triggered by SystemTick and cannot be modified by the user.

3.1.5.1.1 TickTime

Tick 是指 SystemCounter 的计数基值。硬件定时器多久产生一个 Tick 是由 TickTime 确定，TickTime 的单位为 μs，默认值为 1000μs。用户可以在配置工具中根据需求对 TickTime 的大小进行配置。

A counter is represented by a counter value, measured in “ticks”, and the default value is 1000.Users can configure the size of TickTime in Configurator as required.

3.1.5.1.2 Hardware interrupt source

为实现 SystemCounter，OS 会占用一个硬件定时器来周期性地产生 System Tick。OS 会自动创建该硬件定时器的中断，用户可以更改定时器中断的优先级。

To implement SystemCounter, the OS will occupy a hardware timer to periodically generate System Tick. The OS will automatically create the interrupt of the hardware timer, and the user can change the priority of the timer interrupt.

3.1.5.2 User-defined counters

除 SystemCounter 外， OS 支持用户自定义的 counter。

Except SystemCounter, OS supports user-defined counter.

3.1.5.2.1 Trigger of user-defined counters

用户自定义的 counter 没有固定触发源，用户需要调用 IncrementCounter 触发自定义 Counter 增加。例如在周期任务或其他的定时中断中调用。

The user-defined counter does not have a fixed trigger source, and the user needs to call the IncrementCounter to trigger the increase of the user-defined counter. For example, it is called in periodic tasks or other timer interrupts.

用户也可使用 Alarm 触发自定义 Counter，Alarm 的动作中可选择 IncrementCounter。

The user can also use the Alarm to trigger a user-defined counter, and the IncrementCounter can be selected in the action list of the Alarm.

下图为在中断中调用 IncrementCounter 的示例。

The following figure is an example of calling IncrementCounter in an interrupt.

```
ISR(TestISR)
{
    IncrementCounter(TestCounter);
}
```

Figure3-10　trigger user-defined counters by ISR

3.1.6 Alarm

3.1.6.1 Alarm Concept

用户可以使用报警进行定时，定时时间到后，可以执行相应的动作。

The user can use the alarm for timing, and when the timing is up, the corresponding action can be executed.

3.1.6.2 Alarm Actions

The OS alarm supports following actions:

1) Activate tasks;
2) Set events;
3) Call an alarm-callback routine;
4) Trigger a user-defined counter.

用户必须在配置阶段明确设置报警发生时的动作，以及动作实施的对象。若选择激活任务的操作，则必须确定被激活的任务 ID；若选择设置事件的操作，则需要确定所设置的事件以及该事件对应的任务；若报警动作选择是调用报警回调函数，则需要确定回调函数的函数名；若报警动作是触发软件计数器值，则需要确定要触发的计数器。

The user must clearly set the action and target object of action when the alarm expires during configuration phase. If the alarm action of activate task is selected, the activated task ID must be determined. If the user chooses the alarm action of set event, the user need to determine both event and task. If the alarm action is to call the alarm callback function, the function name needs to be determined. If the alarm action is to trigger a user-defined counter, the counter to be triggered needs to be determined.

在多核情况下，Alarm 没有属于哪个核的配置项，其归属与其关联的 Counter 一致。

In the case of Multi-Core, there is no configuration item about which core the Alarm belongs to, and its ownership is consistent with the configuration of its associated counter.

[Notice]

报警回调函数只在裁剪类 SC1 中提供。

Alarm callback function is provided only in SC1.

在 OS 中，每一个报警回调函数均是按下图中的方式进行实现的。下图中的实例将报警回调函数的名称设置为“TestCallback”。用户可以在回调函数中集成应用程序。

In OS, each alarm callback function is implemented as shown in figure below. The example in figure below sets the name of the callback function to "TestCallback". Users can integrate applications in callback functions.

```
ALARMCALLBACK(TestCallback)
{
    /*user code*/
}
```

Figure3-11 Example of callback function

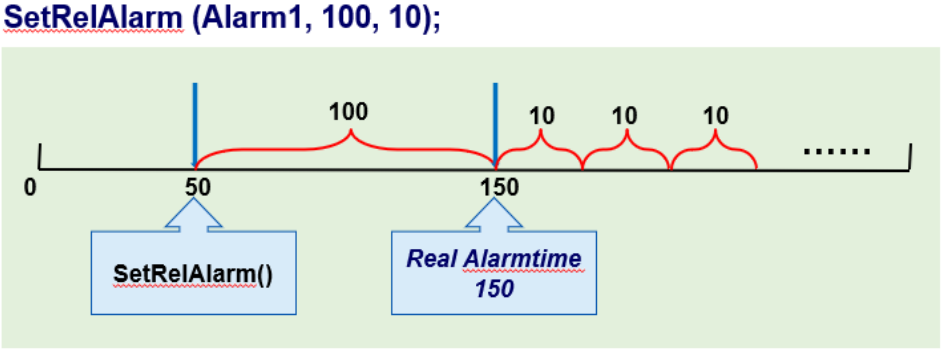
3.1.6.3 Alarm Activation

有两种方法可以启动 Alarm，自启动的 Alarm，可参见“Auto-Start Features”章节。

对于非自启动的 Alarm，用户需要调用 SetAbsAlarm()或 SetRelAlarm()启动报警，设置报警的启动时间和周期。下图的示例展示了相对报警和绝对报警的区别。

There are two ways to start the Alarm. For auto-start Alarm, please refer to chapter” Auto-Start Features”.

For non-auto-start alarms, the user needs to call SetAbsAlarm() or SetRelAlarm() to start the alarm, also set the start time and period of the alarm. The example in the figure below shows the difference between relative alarms and absolute alarms.



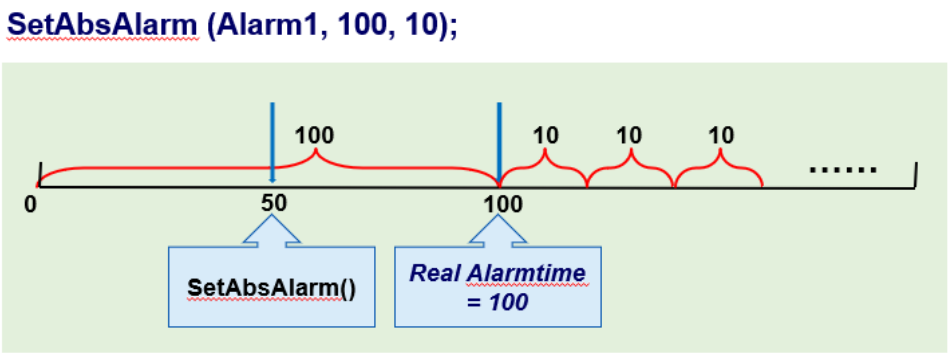


Figure3-12 Example of Set Alarms

3.1.7 Resource

OS 提供了资源管理功能，用于解决任务或中断同时访问共享资源时的协调问题。共享资源可以是应用程序段、内存空间或者硬件外设等。OS 的资源管理机制可以防止资源死锁，任务优先级反转等问题。

OS provides a resource management function to solve the coordination problem when tasks or interrupts are simultaneously accessing shared resources. Shared resources can be application program segments, memory space, hardware peripherals and so on. The OS's resource management mechanism can prevent problems such as resource deadlock and task priority reverse.

3.1.7.1 Service of Resource

OS provides 2 system service functions for resource management:

StatusType GetResource(ResourceType ResID)

StatusType ReleaseResource(ResourceType ResID)

The example in figure below shows how to occupy and release a resource in OS. In this example, the resource Res1 is assigned to Task1.

```
TASK(Task1)
{
    /* Getting Res1*/
    GetResource (Res1);

    /* Operations within Res1 */

    /* Releasing Res1*/
    ReleaseResource (Res1);

    TerminateTask();
}
```

Figure3-13　Example of resource handle at task level of OS

3.1.7.2 Resource Ceiling Priority

在配置阶段，用户需要配置 Resource 可以被哪些 Task 或二类中断访问。基于用户配置，Resource 的天花板优先级由配置工具计算获得。任务的天花板优先级为软件优先级，是所有访问该资源的任务中的最高优先级。中断的天花板优先级为硬件件优先级，是所有访问该资源的中断中的最高优先级。

In the configuration phase, the user needs to configure which Task or C2 interrupts can access this Resource. Based on user configuration, the ceiling priority of Resource is calculated by the configuration tool. The ceiling priority of the task is the software priority, which is the highest priority among all tasks that access the resource. The ceiling priority of the interrupt is the hardware priority, which is the highest priority among all interrupts that access the resource.

用户调用 GetResource 获取资源时，OS 会将任务和中断的优先级提升至该资源的天花板优先级。之后，访问该资源的其他任务或中断将无法运行，直到该资源被释放。资源被释放时，OS 将会恢复获取该资源之前的任务和中断的优先级。

When the user calls GetResource to lock a resource, the OS will raise the priority of the task and interrupt to the ceiling priority of the resource. After that, other tasks or interrupts that access the resource will not be able to run until the resource is released. When the resource is released, the OS will resume the priority of the task and interrupt before locking the resource.

[Notice]—应该可以参考

1.资源的获取和释放，须遵循 LIFO（后进先出）的原则。

2.如果资源是任务和中断共享的，当该资源被任务获取后，OS 将会锁定调度（所有任务都不能抢占当前任务，包括与当前资源无关的任务），这样做是为了防止使用该资源的中断被锁定的时间过长。

3.如果一个任务或中断占用了多个资源，则天花板优先级为所有被占用资源中最高的优先级。

1. The get and release of resources must follow the principle of LIFO (last in, first out).

2. If the resource is shared by tasks and interrupts, when the resource is got by the task, OS will lock the scheduling (all tasks cannot preempt the current task, including tasks unrelated to the current resource). This is to prevent the interrupt who use this resource is locked for too long.

3. If a task or interrupt get multiple resources, the ceiling priority is the highest priority among all occupied resources.

3.1.8 Event

OS 完全支持标准中规定的事件机制，用于实现系统内不同的实体之间的同步，包括任务与任务进行同步或是任务与中断服务函数进行同步。

OS is fully Conform to the standard event mechanism which is used to implement the system synchronization between different entities, including synchronization between tasks and synchronization between task and interrupt.

只有扩展任务可以调用 WaitEvent 等待事件，调用 WaitEvent 后，任务不再运行，进入 WAIT 状态。

Only extended tasks can call WaitEvent to wait for an event. After calling WaitEvent, the task will no longer run and enter the WAIT state.

任务，二类中断中，用户可以调用 SetEvent 设置事件。Alarm，Schedule Table 也可以配置设置事件的动作。设置事件后，等待该事件的任务将恢复至 READY 状态。然后 OS 会基于当前系统的抢占策略，任务优先级的情况，确定何时恢复任务的运行。

In the task, C2 interrupt, the user can call SetEvent to set the event. Action of Alarm and Schedule Table can also be configured to set event. After setting the event, the task waiting for the event will return to the READY state. Then the OS will determine when to resume the task based on the current system's preemption strategy and task priority.

每个扩展任务支持最多 32 个事件。WaitEvent 和 SetEvent 均可以等待或设置多个事件。

Each extended task supports up to 32 events. Both WaitEvent and SetEvent can wait or set multiple events.

下图的实例显示了 OS 中两个任务之间的同步实现。推荐用户按照图中的代码示例运用 OS 的同步机制。

The example shown in figure below shows the synchronization implementation between two tasks in OS. It is recommended that users follow the example code in the figure to use the synchronization mechanism in OS.

The Synchronized Task

```
TASK(Task1)
{
 EventMaskType event;
 ..................
 for(;;)
 {
   /* Synchronization
      Request */
   WaitEvent(Event1|Event2);
   /* Synchronization
      Completed*/
   GetEvent(Task1,&event);
   ClearEvent(event);
   if(event & Event1)
   {
   /* Operation for Event1 */
   }
   if(event & Event2)
   {
   /* Operation for Event2 */
   }
 }
}
```

The Synchronizing Task/ISR

```
TASK(Task2)/* Or ISR(Isr1)*/
{
 ..................
 ..................
 ..................
 /* Synchronizing point */
 SetEvent(Event1);
 ..................
 ..................
}
```

Figure3-14 Example of event synchronization

3.1.9 Interrupt

OS 的中断分为一类中断和二类中断。中断的很多特性是和硬件强相关的，请参见不同芯片的 OS 的使用说明。本文只描述和硬件无关的内容。

OS interrupts are divided into C1 interrupt and C2 interrupt. Many characteristics of interrupts are strongly related to the hardware, please refer to the user manual of the OS of different chips. This article only describes content that has nothing to do with hardware.

用户可以配置中断处理函数的名称，在中断触发后，OS 经过预处理之后，会调用该处理函数。中断处理函数由用户实现。可参见下图示例。

The user can configure the name of the interrupt processing function. After the interrupt is triggered, the OS will call the processing function after preprocessing. The interrupt processing function is implemented by the user. See the example below.

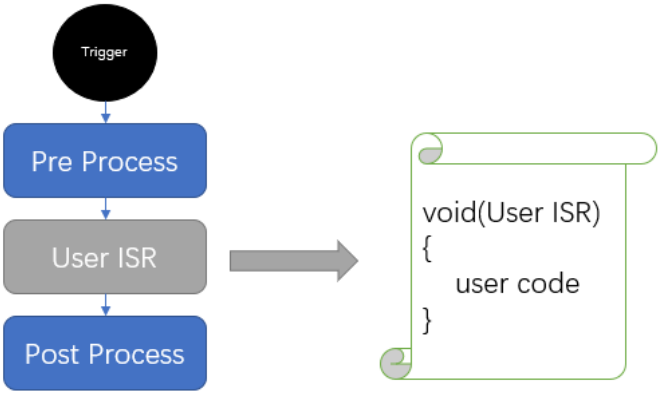


Figure3-15 Interrupt Process

[Notice]

使用中断的一些通用注意事项，请参见《EAS470_OS_SafetyManual》第 3.4 章节。

For some general notice for using interrupts, please refer to chapter 3.4 of "EAS470_OS_SafetyManual".

3.1.9.1 C1 Interrupt

针对 C1 Interrupt，OS 仅添加了如下处理。

- 保存恢复现场。
- 切换 OS Context。
- 处理计算 CPU 负载。
- 调用用户的处理函数。

For C1 Interrupt, OS only adds the following processing.

- Save and restore the context of the program.
- Switch OS Context.
- Processing and calculating CPU load.
- Call user process function.

[Notice]—可以参考

（1）一类中断中，用户能调用的 OS 的 API 很少，具体参见“OS Context”章节。

（2）用户应尽量缩短一类中断的执行时间。

（3）一类中断执行结束后，只能恢复至程序被打断的位置，不会触发任务调度。

（4）一类中断运行时，其内存访问权限和 OS 内核是一致的（最高权限）。

（5）一类中断没有独立的栈，其使用的是被其打断的任务或二类中断的栈。

（6）如果任务或二类中断使能了时间保护，一类中断不会暂停时间保护的计时。

（7）一类中断中可以调用下述中断开关的接口,用户需确保这些接口成对调用。OS的服务保护不涉及一类中断，如果用户忘记了恢复中断状态，可能导致系统运行不正常。

- DisableAllInterrupts /EnableAllInterrupts
- SuspendAllInterrupts/ResumeAllInterrupts
- SuspendOSInterrupts/ResumeOSInterrupts

(1) In C1 interrupt, there are few OS APIs that users can call. For details, see Chapter “OS Context”.

(2) The user should try to shorten the execution time of C1 interrupt.

(3) After the execution of the C1 interrupt is completed, it can only be restored to the position where the program was interrupted, and the task scheduling will not be triggered.

(4) When C1 interrupt is running, its memory access authority is consistent with the OS kernel (the highest authority).

(5) The C1 interrupt has no independent stack, it uses the stack of task or C2 interrupt which interrupted by it.

(6) If the task or the C2 interrupt has time protection enabled, the C1 interrupt will not suspend the timing of the time protection.

(7) The following interrupt switch interfaces can be called in C1 interrupt, and the user needs to ensure that these interfaces are called in pairs. The service protection of the OS does not involve C1 interrupt. If the user forgets to restore the interrupted state, it may cause the system to operate abnormally.

- DisableAllInterrupts /EnableAllInterrupts
- SuspendAllInterrupts/ResumeAllInterrupts
- SuspendOSInterrupts/ResumeOSInterrupts

3.1.9.2 C2 Interrupt

针对 C2 Interrupt，OS 需要处理的内容如下：

（1）保存和恢复程序上下文。

（2）切换 OS Context。

（3）记录中断嵌套的信息。

（4）处理被打断任务或中断，以及当前中断的时间保护。

（5）切换内存访问权限。

（6）切换堆栈。

（7）调用 PreISRHook 和 PostISRHook。

（8）调用用户服务函数。

（9）退出中断。

（9-1 如果当前有中断嵌套，则恢复程序上下文，恢复至被打断的中断。

（9-2）如果无中断嵌套，且没有更高优先级的任务被激活，则恢复程序上下文，恢复至被打断的任务。

（9-3）如果无中断嵌套，但有更高优先级的任务被激活，则切换至新的任务。

For C2 Interrupt, the OS needs to deal with the following:

(1) Save and restore the program context.

(2) Record interrupt nesting information.

(3) Switch OS Context.

(4) Deal with the time protection of interrupted task or interrupt, and current interrupt.

(5) Switch memory access permissions.

(6) Switch the stack.

(7) Call PreISRHook and PostISRHook.

(8) Call the user service function.

(9) Exit interrupt.

(9-1) If there is interrupt nesting currently, the program context will be restored and the interrupted interrupt will be restored.

(9-2) If there is no interrupt nesting and no higher priority task is activated, the program context is restored and the interrupted task is restored.

(9-3) If there is no interrupt nesting, but a higher priority task is activated, then switch to a new task.

二类中断中可以调用的 OS 的 API，参见参见“OS Context”章节。

For the OS APIs that can be called in the C2 interrupt, please refer to the "OS Context" chapter.

在系统配置阶段用户在配置中断时，需要将各个中断服务函数与对应的硬件中断源进行关联。

The user needs to associate each interrupt service function with the corresponding hardware interrupt source when configuring the interrupt.

3.1.9.3 Interrupt Nest

OS 支持中断嵌套，用户可在工具上配置哪些中断支持嵌套。

The OS supports interrupt nesting, and the user can configure which interrupts support nesting on the tool.

3.1.9.4 Trap

OS 会为硬件的 Trap 提供 Hook 函数。Trap 是硬件强相关的内容，更多的描述需要参见特定芯片的 OS 的使用说明。

OS will provide Hook function for hardware Trap. Trap is strongly related to the hardware. For more description, please refer to the user manual of the specific chip's OS.

3.1.10 Schedule Table

3.1.10.1 Schedule Table Concept

用户可在 Schedule Table 配置多个 EP，每个 EP 具有不同的时间偏移，每个 EP 可执行多个不同的动作。调度表可以帮助用户更方便的实现一系列具有时序要求的动作。

调度表可以配置为周期运行的调度表。支持同时启动多个调度表。支持自启动的调度表。

The user can configure multiple EPs in the Schedule Table, each EP has a different time offset, and each EP can perform multiple different actions. The schedule table can help users more conveniently implement a series of actions with timing sequence.

The schedule table can be configured as periodically. Support to start multiple schedule tables at the same time. Support auto-start schedule table.

下图是调度表的示例。

The figure below is an example of a schedule table.

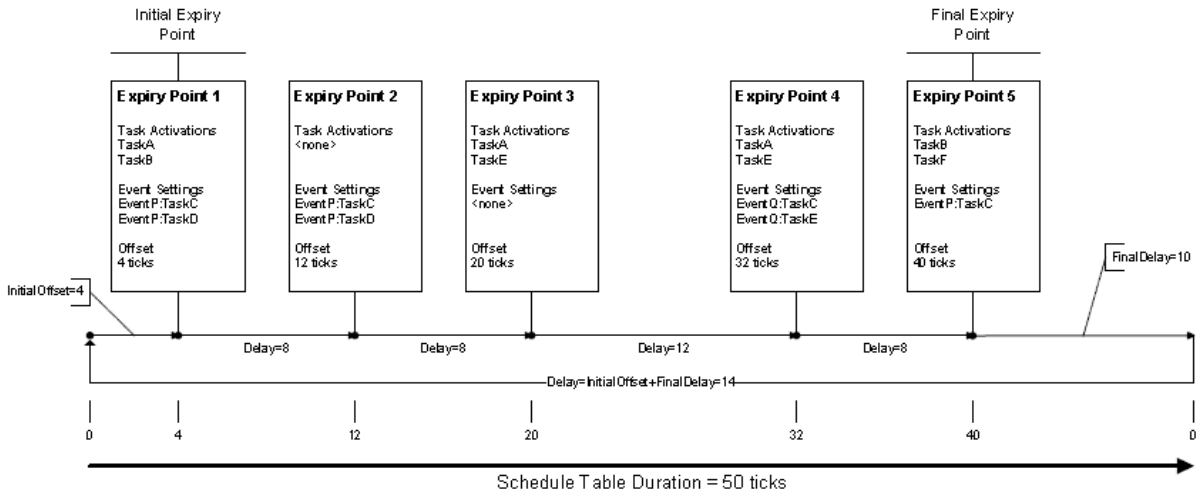


Figure3-16 Example of Schedule Tables

上图中，EP 的动作，Offset，Final Delay 等都需要在配置阶段进行静态配置。

In the above figure, EP actions, Offset, Final Delay, etc. all need to be statically configured during the configuration phase.

3.1.10.2 EP Actions

OS 中，EP 到期发生时可以进行以下两种操作：

In OS, the following two operations can be performed when the EP expires:

1) Activate task
2) Set event

两种操作的个数不限，但不能都为空。用户必须在配置阶段明确设置每个 EP 发生时的动作，以及动作实施的对象。若选择激活任务的操作，则必须确定被激活的任务 ID；若选择设置事件的操作，则需要确定所设置的事件以及该事件对应的任务。若既配置了激活任务又配置了设置事件，则所有激活任务都发生在设置事件之前。

There are no restrictions on the number of the two operations, but they cannot all be empty. The user must explicitly set the actions that occur when each EP occurs during the configuration phase, as well as the objects that the actions are performed on. If the user chooses to activate the task, they must determine the task ID; if the user chooses the operation of setting an event, they need to determine events to be set and the task corresponding to the event. If both the activating tasks and the setting events are configured, all activating tasks will occur before the setting event.

3.1.10.3 Activation of Schedule Table

有两种方法可以启动 Schedule Table，自启动的 Schedule Table，可参见“Auto-Start Features”章节。

对于非自启动的 Schedule Table，用户需要调用 StartScheduleTableAbs ()或 StartScheduleTableRel ()启动 Schedule Table，设置 Schedule Table 的启动时间。相对时间和绝对时间与 Alarm 中的描述是相同的。

There are two ways to start the Schedule Table. For the auto-start Schedule Table, please refer to the "Auto-Start Features" chapter.

For a non-auto-start Schedule Table, the user needs to call StartScheduleTableAbs () or StartScheduleTableRel () to start the Schedule Table and set the start time of the Schedule Table. The relative time and absolute time are the same as the description in Alarm.

3.1.11 Stack Monitor

堆栈检测将进行栈初始化，并且当发生栈溢出的错误现象时，可以及时告知用户，

并最终执行关闭 OS。

The stack detection function will initialize the stack, and when the stack overflow error occurs, it can inform the user in time, and finally shut down the OS.

当退出二类中断、退出任务时，系统会进行堆栈溢出检测，即检查当前中断或任务是否将预先配置的栈全部使用耗尽。

When exiting C2 interrupt or task, the system will detect stack overflow, that is, check whether the current interrupt or task will use up all the pre-configured stacks.

3.1.12 CPU Load

系统提供 CPU 负载计算功能，可计算特定时间内（OS_CPULOAD_BASE_TIME）系统的 CPU 负载率，系统提供了 GetCpuLoadValue 接口函数，用户应以固定周期（OS_CPULOAD_BASE_TIME）调用该函数，以便获取最近一次该周期内的系统负载率，详细配置方法请参考配置章节。

The EAS_OS provides CPU load calculation function, can calculate CPU load rate of the system in a specific time (OS_CPULOAD_BASE_TIME). The OS provides GetCpuLoadValue interface function, User should call it in fixed cycle (OS_CPULOAD_BASE_TIME) to obtain the system load rate in the last period. Please refer to the configuration section for detailed configuration method.

3.1.13 Error Handling

3.1.13.1 Error Record

当 OS 内部检测到故障后，OS 会将错误信息记录在变量 Os_ErrRecord 中，包含下述内容。

Table3-1 Error Format

Name	Description
errCore	发生错误的核。 The core that occurs error.
serviceId	服务函数的 ID。 格式为 OSServiceId_xx。 The ID of service function. The format is OSServiceId_xx.
error	错误码。包含如下类型。 Error code. Including following format. E_OS_xx, OSEK or AUTOSAR Standard error code. E_OS_SYS_xx, EAS.OS self-defined error code.

	E_OS_SYSFATA_xx, Fatal error code.
errDescription	进一步的错误描述，格式为 OS_IE_xx。 Further error description, the format is OS_IE_xx.
errPar	记录的错误参数。 The recorded error parameter.

例如，用户调用 SetEvent，传入了错误的 TaskID，serviceId = OSServiceId_SetEvent，error = E_OS_ID， errDescription = OS_IE_TASK_ID，errPar[0] 记录了错误的 TaskID。

For example, the user calls SetEvent and input wrong TaskID, serviceId = OSServiceId_SetEvent, error = E_OS_ID, errDescription = OS_IE_TASK_ID, errPar[0] records the wrong TaskID.

用户可以在 Os_Err.h 中找到 error, serviced, errDescription 相关的宏定义。

Users can find macro definitions related to error, serviced and errDescription in Os_Err.h.

第 4 章 API 部分，详细介绍了每个 API 可以检测到的错误，以及上报的信息。

The API part of Chapter 4 describes in detail the errors that can be detected by each API and the reported information.

在 AUTOSAR 规范中，为 IOC 的接口定义了专门的错误码，因此 OS 也在 Os_Err.h 定义了专门的宏。描述如下。

In the AUTOSAR specification, special error codes are defined for the IOC interface, so the OS also defines a special macro in Os_Err.h. Described as follows.

Table3-2 IOC Error Code

Name	**Value**	**Description**
IOC_E_OK	0	No error.
IOC_E_NOK	1	Common IOC error.
IOC_E_LIMIT	130	IOC queue is full.
IOC_E_LOST_DATA	64	Not used.
IOC_E_NO_DATA	131	IOC queue is empty.

更多的描述参见 IOC 部分的 API。

For more description, please refer to the API in the IOC section.

3.1.13.2Error Level

Table3-3 Error Level

Level	**Description**

Standard	标准错误，OS 始终会检查标准错误。 检测到标准错误后，会调用 ErrorHook，用户本次请求会被拒绝，不影响系统继续运行。 Standard error, OS will always check standard error. After the standard error is detected, ErrorHook will be called, and the user's request will be rejected, system can continue running.
Extended	扩展错误，只有当用户配置 OsStatus = Extended 时候，才会检查这些错误。 检测到扩展错误后，会调用 ErrorHook，用户本次请求会被拒绝，不影响系统继续运行。 Extended errors, these errors will only be checked when the user configures OsStatus = Extended. After the extended error is detected, ErrorHook will be called, and the user's request will be rejected, system can continue running.
Fatal	致命错误。 检测到致命错误后，调用 ErrorHook，然后 Shutdown OS。 Fatal error. After a fatal error is detected, ErrorHook is called, and then the OS is shut down.

第 9 章的附表 1 中，详细介绍了所有的致命错误。

Table1 of Chapter 9 details all fatal errors.

3.1.13.3ErrorHook

用户可以配置是否使能 ErrorHook。如果使能，OS 检测到任意错误时，都会调用该函数报告给用户。用户可在该 Hook 函数中添加自定义的代码。

Users can configure whether to enable ErrorHook. If enabled, when the OS detects any error, it will call this function to report it to the user. Users can add custom code in the Hook function

```
void ErrorHook(StatusType Error)
{
    /*usercode*/
}
```

Figure3-17　ErrorHook

3.1.13.4Other

时间保护和内存保护相关的错误检测，参见 3.2 和 3.3 章节的描述。

For error detection related to time protection and memory protection, see the descriptions in chapters 3.2 and 3.3.

3.1.14OS Context

OS Context 指的是当前 OS 运行的状态， 在不同的 OS Context 下，允许调用的 OS API 是不同的，参见下表。

OS Context refers to the current operating state of the OS, under different OS Contexts, the OS APIs allowed to be called are different, see the table below.、

相对 AUTOSAR 规范，EAS 的 OS 增加了 PreISRHook 和 PostISRHook 两个 OS Context。他们允许调用的 API 和 PreTaskHook 和 PostTaskHook 是一致的。

Compared with the AUTOSAR specification, the EAS OS has added two OS Contexts, PreISRHook and PostISRHook. The API they allow to call is the same as PreTaskHook and PostTaskHook.

Service	Task	Cat1 ISR	Cat2 ISR	Error Hook	PreTask Hook	PostTask Hook	Startup Hook	Shutdown Hook	Alarm Callback	Protection Hook
ActivateTask	✓		✓							
TerminateTask	✓		C							
ChainTask	✓		C							
Schedule	✓		C							
GetTaskID	✓		✓	✓	✓	✓				✓
GetTaskState	✓		✓	✓	✓	✓				
DisableAllInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EnableAllInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SuspendAllInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ResumeAllInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SuspendOSInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ResumeOSInterrupts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GetResource	✓		✓							
ReleaseResource	✓		✓							
SetEvent	✓		✓							
ClearEvent	✓		C							
GetEvent	✓		✓	✓	✓	✓				
WaitEvent	✓		C							
GetAlarmBase	✓		✓	✓	✓	✓				
GetAlarm	✓		✓	✓	✓	✓				
SetRelAlarm	✓		✓							
SetAbsAlarm	✓		✓							
CancelAlarm	✓		✓							
GetActiveApplicationMode	✓		✓	✓	✓	✓	✓	✓		
StartOS										
ShutdownOS	✓		✓	✓			✓			

Service	Task	Cat1 ISR	Cat2 ISR	Error Hook	PreTask Hook	PostTask Hook	Startup Hook	Shutdown Hook	Alarm Callback	Protection Hook
GetApplicationID	✓		✓	✓	✓	✓	✓	✓		✓
GetISRID	✓		✓	✓						✓
CallTrustedFunction	✓		✓							
CheckISRMemoryAccess	✓		✓	✓						✓
CheckTaskMemoryAccess	✓		✓	✓						✓
CheckObjectAccess	✓		✓	✓						✓
CheckObjectOwnership	✓		✓	✓						✓
StartScheduleTableRel	✓		✓							
StartScheduleTableAbs	✓		✓							
StopScheduleTable	✓		✓							
NextScheduleTable	✓		✓							
StartScheduleTableSynchron	✓		✓							
SyncScheduleTable	✓		✓							
GetScheduleTableStatus	✓		✓							
SetScheduleTableAsync	✓		✓							
IncrementCounter	✓		✓							
GetCounterValue	✓		✓							
GetElapsedValue	✓		✓							
TerminateApplication	✓		✓	✓[2]						
AllowAccess	✓		✓							
GetApplicationState	✓		✓	✓	✓	✓	✓	✓		✓
ControlIdle	✓		✓							
GetCurrentApplicationID	✓		✓	✓	✓	✓	✓	✓		✓

Service	Task	Cat1 ISR	Cat2 ISR	Error Hook	PreTask Hook	PostTask Hook	Startup Hook	Shutdown Hook	Alarm Callback	Protection Hook
GetNumberOfActivatedCores	✓		✓							
GetCoreID	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
StartCore										
StartNonAutosarCore										
GetSpinlock	✓		✓							
ReleaseSpinlock	✓		✓							
TryToGetSpinlock	✓		✓							
ShutdownAllCores	✓		✓	✓			✓			

Figure3-18 Allowing Service in OS Context

注：上图中只有√表示支持在该 OS_CONTEXT 下调用。空白和 C 均表示不支持。

Note: Only √ in the above figure indicates support for calling under this OS_CONTEXT. Blank and C both mean not supported.

3.2 SC2 Timing Protection

3.2.1 Main Function

时间保护主要功能如下：

1）任务/二类中断执行时间的保护

2）任务/二类中断最小执行时间间隔的保护

3）任务/二类中断锁定时间的保护

Main feathers are as follows:

1) Execution time protection of Task or C2 ISR
2) Inter-arrival time protection of Task or C2 ISR
3) Locking time protection of Task or C2 ISR

时间保护的实现与芯片的定时器模块强相关，关于使用了哪些定时器，请参考芯片与芯片相关的 OS 使用说明部分。

The implementation of timing protection is strongly related to the timer module of the chip. For which timers are used, please refer to the OS User Manual related to the chip.

3.2.2 Execution time protection

执行时间保护包括任务预期执行时间（OsTaskExecutionBudget）的保护和中断预期执行时间（OsIsrExecutionBudget）的保护。

任务预期执行时间/中断预期执行时间：允许任务或中断可以执行的最大时间。如果任务或中断运行时间超出了其预期时间, OS 将会执行 ProtectionHook()函数，参数为 E_OS_PROTECTION_TIME。

Execution time protection includes OsTaskExecutionBudget and OsIsrExecutionBudget.

OsTaskExecutionBudget/ OsIsrExecutionBudget: The maximum allowed execution time of the task/C2 ISR. If a task's/ISR's OsTaskExecutionBudget/ OsIsrExecutionBudget is reached, then the OS will call the ProtectionHook() with E_OS_PROTECTION_TIME.

当任务或中断被打断时，系统会停止任务或中断的执行时间计时，直到该任务或中断被恢复，系统继续对其执行时间计时。对于扩展任务，其状态由 RUNNING 转换成

WAITING 时，系统会停止其执行时间计时，当其等待的事件被成功设置后，重新开始该任务的执行时间计时。

When a task or interrupt is interrupted, the system will stop timing the execution time of the task or interrupt until the task or interrupt is restored, and the system will continue timing its execution time. For an extended task, when its state changes from RUNNING to WAITING, the system will stop its execution time timing. When the waiting event is successfully set, the system will restart the execution time timing of the task.

3.2.3 Inter-arrival time protection

到达间隔时间保护包括任务间隔时间（OsTaskTimeFrame）的保护和中断间隔时间（OsTaskTimeFrame）的保护。

任务间隔时间：任务运行的最小时间间隔。

中断间隔时间：中断运行的最小时间间隔。

Inter-arrival time protection includes OsTaskTimeFrame and OsIsrTimeFrame.

OsTaskTimeFrame: The minimum inter-arrival time between task running.

OsIsrTimeFrame: This minimum inter-arrival time between interrupts running.

如果尝试在小于任务时间间隔内再次激活任务，OS 将会调用 ProtectionHook()函数，其参数为 E_OS_PROTECTION_ARRIVAL。如果用户在 Hook 函数中返回 PRO_IGNORE，那么 OS 将会再次执行激活任务，否则不会执行真正的激活。

If an attempt is made to activate a task before the end of an OsTaskTimeFrame then the OS module shall not perform the activation (unless returns PRO_IGNORE) AND shall call the ProtectionHook() with E_OS_PROTECTION_ARRIVAL.

如果用户尝试设置一个事件，但是相应的扩展任务运行间隔小于任务间隔时间，那么 OS 会调用 ProtectionHook()函数，其参数为 E_OS_PROTECTION_ARRIVAL。如果用户在 Hook 函数中返回 PRO_IGNORE，这个事件将会被设置成功，OS 将执行该扩展任务。

If user attempts to set an event, but the corresponding extended task running interval is less than the task interval time, then the OS will call the ProtectionHook() function with the parameter E_OS_PROTECTION_ARRIVAL. If the user returns PRO_IGNORE in the Hook function, this event will be set successfully, and the OS will execute the extended task.

如果一个二类中断发生的频次小于中断间隔时间，那么 OS 将不会执行该中断的用户处理函数，同时调用 ProtectionHook()函数，其参数为

E_OS_PROTECTION_ARRIVAL。如果用户在 Hook 函数中返回 PRO_IGNORE，该中断仍可被执行。

If C2 ISR occurs before the end of the OsIsrTimeFrame then the OS shall not execute the user provided ISR AND shall call the ProtectionHook() with E_OS_PROTECTION_ARRIVAL. If the user returns PRO_IGNORE in the Hook function, the interrupt can still be executed.

[Notice]

只有 Inter-arrival time protection 类型的错误，用户可在 ProtectionHook()中返回 PRO_IGNORE。具体请参见 ProtectionHook 章节。

Only Inter-arrival time protection type errors, the user can return PRO_IGNORE in ProtectionHook(). For details, please refer to the ProtectionHook chapter.

3.2.4 Locking time protection

锁定时间保护包括：

Locking time protection includes：

OsTaskResourceLockBudget, Task lock resource time protection.
OsIsrResourceLockBudget, ISR lock resource time protection.
OsTaskAllInterruptLockBudget, Task disable all interrupts time protection.
OsTaskOSInterruptLockBudget, Task disable all C2 interrupts time protection.
OsIsrAllInterruptLockBudget, ISR disable all interrupts time protection.
OsIsrOSInterruptLockBudget, ISR disable all C2 interrupts time protection.

任务预期锁定资源时间/中断预期锁定资源时间：任务或中断允许锁定资源的最大时间。

OsTaskResourceLockBudget/OsIsrResourceLockBudget: the maximum time the task/C2 ISR is allowed to lock the resource.

任务预期锁定所有中断时间/二类中断中预期锁定所有中断的时间：任务或二类中断中允许锁定所有中断（通过调用 SuspendAllInterrupts()或 DisableAllInterrupts()）的最大时间。

OsTaskAllInterruptLockBudget/OsIsrAllInterruptLockBudget: The maximum time for which the task/C2 ISR is allowed to lock all interrupts (via SuspendAllInterrupts() or

DisableAllInterrupts()).

任务预期锁定所有二类中断时间/二类中断中预期锁定所有二类中断的时间：任务或二类中断中允许锁定所有中断（通过调用 SuspendOSInterrupts ()）的最大时间。

OsTaskOSInterruptLockBudget/OsIsrOSInterruptLockBudget: The maximum time for which the task/C2 ISR is allowed to lock all Category 2 interrupts (via SuspendOSInterrupts()).

如果任务或中断被打断，那么相应任务或中断的锁定时间计时将会暂停，当系统重新回到该任务或中断时，锁定时间计时将会继续。

If the task or C2 ISR is preempted/interrupted, the lock time timer responding to the task or C2 ISR will be suspended. When the system returns to the task or C2 ISR, the lock time timer will continue.

如果任务或二类中断超出了锁定时间保护，那么 OS 将会调用 ProtectionHook()，参数为 E_OS_PROTECTION_LOCKED。

If a Task or C2 ISR exceeds Locking time protection, the OS will call the ProtectionHook() with E_OS_PROTECTION_LOCKED.

3.2.5 Time protection nested

在任务和中断执行时，Execution time protection 和不同的 Locking time protection 可能会同时启动，称之为 Time protection nested。当这种情况发生时，OS 将以最短的超时时间为准进行监控。

When tasks and interrupts are executed, Execution time protection and different Locking time protection may be activated at the same time, which is called Time protection nested. When this happens, the OS will monitor with the shortest timeout budget.

下图展示了一个 nested 的例子。

The figure below shows a nested example.

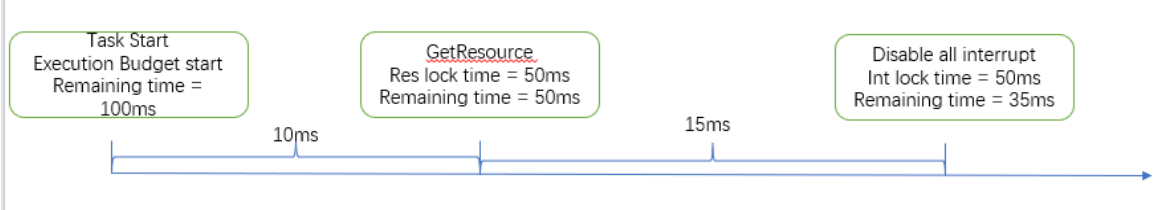


Figure3-19 TP nested example

上图描述如下：

（1）任务启动时，Execution time protection 启动计时，Remain Time = 100ms。

（2）10ms 后，任务调用了 GetResource，资源 Lock time protection 的时间配置为 50ms，当前任务的 Execution Budget 还剩余 90ms，因此，OS 当前监控的剩余时间更新为较短的 50ms。

（3）15ms 后，任务调用了 DisableAllInterrupts，中断的 Lock time protection 的时间配置为 50ms，此时资源的 Lock time protection 还剩余 50-15 = 35ms。因此 OS 当前监控的剩余时间仍为较短的 35ms。

The picture above is described as follows:

(1) When the task is started, the Execution time protection starts timing, Remain Time = 100ms.

(2) After 10ms, the task calls GetResource, the resource Lock time protection time is configured to 50ms, and the Execution Budget of the current task has 90ms left. Therefore, the remaining time of the OS currently monitored is updated to a shorter 50ms.

(3) After 15ms, the task calls DisableAllInterrupts, and the interrupted Lock time protection time is configured as 50ms. At this time, the lock time protection of the resource still has 50-15 = 35ms. Therefore, the remaining time currently monitored by the OS is still 35ms.

3.3 SC3

3.3.1 OS-Application

OSEK OS 允许任何 task/ISR 设置 event，也允许任何 task/ISR 操作 alarm。因此某个任务出错时，该错误可能会传播到同一处理器上的其他任务中。为了实现更可靠的保护，AUTOSAR OS 引入 OS-Application 来解决此问题。

OSEK OS allows any task/ISR to set events and also allows any task/ISR to operate alarm. Therefore, when a task fails, the error may be propagated to other tasks on the same processor. In order to achieve more reliable protection, AUTOSAR OS introduces OS-Application to solve this problem.

Task, ISR, Counter, Alarm, Schedule Table, Resource 称为 OS 的 Object。用户需要配置 2 类属性：

- Object 属于哪个 OS-Application。
- Object 可以被哪个 OS-Application 访问。

Task, ISR, Counter, Alarm, Schedule Table, Resource are called OS Objects. The user needs

to configure 2 types of attributes:

- Which OS-Application the object belongs to.
- Which OS-Application can access the Object.

注 1： Resource 不需要配置其所属的 OS-Application，但需要配置那些 OS-Applications 可以访问该 Resource。

注 2：属于同一 OS-Applications 的所有对象都可以相互访问。

注 3：Event 所属的扩展任务如果可以访问，则 Event 也可以被访问。

注 4：中断发生的时间不确定，因此中断可以被任意的 OS-Applications 访问。

注 5：同一 Object 必须属于且只能属于唯一的 OS-Applications。

注 6：如果尝试配置 Alarm 或 ScheduleTable 关联一个它们没有访问权限的 Task，那么配置工具将会报错。

Note 1: Resource does not need to configure the OS-Application to which it belongs, but it is necessary to configure which OS-Applications can access the Resource.

Note 2: All objects belonging to the same OS-Applications can access each other.

Note 3: If the extended task to which the Event belongs can be accessed, the Event can also be accessed.

Note 4: The time when the interrupt occurs is uncertain, so the interrupt can be accessed by any OS-Applications.

Note 5: One Object must belong to and can only belong to the only OS-Applications.

Note 6: If you try to configure Alarm or ScheduleTable to associate a Task that they do not have access to, the configuration tool will report an error.

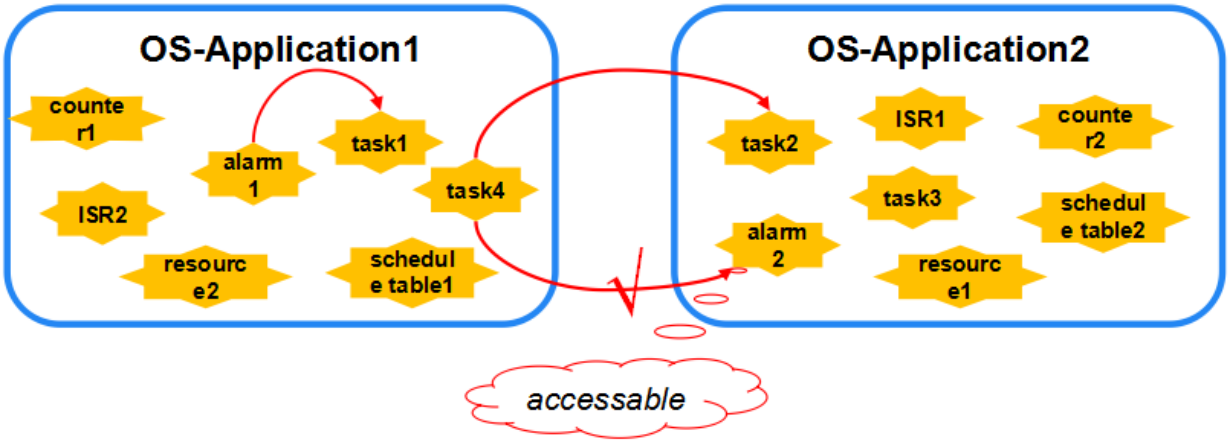


Figure3-20 Access rights of OS-Application

OS-Application 分为两类：可信的和不可信的。可信的 OS-Application 可以不受限制

地访问内存。不可信的 OS-Application，用户需要配置其可访问的内存范围。

There are two classes of OS-Application, Trusted and Non-Trusted. Trusted OS-Application can access memory without restriction. For untrusted OS-Applications, users need to configure the memory range that they can access.

系统中如何划分 OS-Application，取决于功能安全的分析。通常来说，不同的安全等级划分为不同的 OS-Application。多核系统中，不同的核必须划分为不同的 OS-Application。

The division of OS-Application in the system depends on the functional safety analysis. Generally, different safety levels are divided into different OS-Applications. In a multi-core system, different cores must be divided into different OS-Applications.

OS-Applications 的状态转换图如下所示。

The state transition of OS-Applications diagram is shown below.

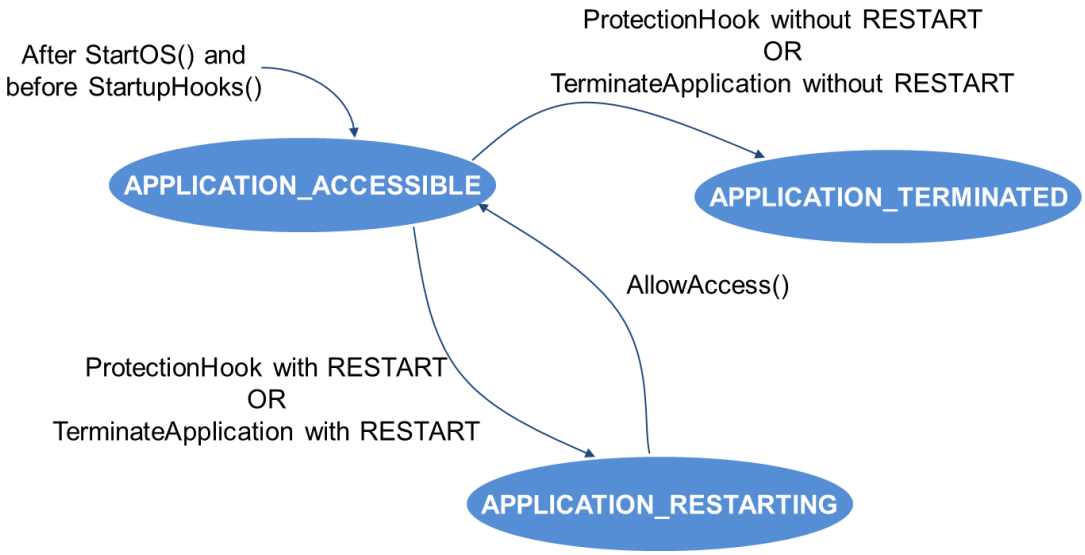


Figure3-21　OS-Application state transition diagram

OS 启动后，默认进入 APPLICATION_ACCESSIBLE 状态。每个操作系统应用程序中的对象是根据配置的权限相互操作和访问的。在正常情况下，操作系统应用程序总是处于这种状态。

After the OS is started, the APPLICATION_ACCESSIBLE state is entered by default. The objects in each OS-Application are mutually operated and accessed according to the configured permissions. Under normal circumstances, the OS-Application is always in this state.

有两种情况会导致 OS-Application 的状态发生更改。一种是 OS-Application 中的对象触发错误，另一种是调用 TerminateApplication()终止 OS-Application。

There are two situations that can cause the state of the OS-Application to change. One is

when an object in OS-Application triggers an error, and the other is when TerminateApplication() is called to terminate an OS-Application.

当 OS 检测到时间保护错误或内存保护错误时，会调用 ProtectionHook()。如果用户在 ProtectionHook()中返回 PRO_TERMINATEAPPL，OS 会终结当前的 OS-Application。如果用户返回 PRO_TERMINATEAPPL_RESTART，OS 会重启当前的 OS-Application。

When the OS detects a time protection error or a memory protection error, it will call ProtectionHook(). If the user returns PRO_TERMINATEAPPL in ProtectionHook(), the OS will terminate the current OS-Application. If the user returns to PRO_TERMINATEAPPL_RESTART, the OS will restart the current OS-Application.

用户也可通过调用 TerminateApplication()终止或重启指定的 OS-Application。

The user can also terminate or restart the specified OS-Application by calling TerminateApplication().

已经被终止的 OS-Application，其所包含的任务、中断、Alarm、Counter、资源、Schedule Table 都不可用。除非重新启动系统，否则无法改变此状态。

For the OS-Application that has been terminated, the tasks, interrupts, Alarms, Counters, resources, and Schedule Tables contained in it are unavailable. This status cannot be changed unless the system is restarted.

支持重启的 OS-Application，用户需要为其配置一个 RestartTask，重启 OS-Application 时，指定 OS-Application 进入 APPLICATION_RESTARTING 状态，并激活 RestartTask。OS-Application 中包含的任务、中断、Alarm、Counter、资源、Schedule Table 在此状态下都不可用。在 RestartTask 中，用户调用 AllowAccess（）后，操作系统应用程序将返回到 APPLICATION_ACCESSIBLE 状态，所包含的对象将恢复正常使用。

For OS-Applications that support restarting, users need to configure a RestartTask for it. When restarting the OS-Application, specify the OS-Application to enter the APPLICATION_RESTARTING state and activate the RestartTask. The tasks, interrupts, alarms, counters, resources, and schedule tables included in the OS-Application are not available in this state. In RestartTask, after the user calls AllowAccess(), the operating system application will return to the APPLICATION_ACCESSIBLE state, and the contained objects will resume normal use.

在单核系统上，OS-Applications 仅适用于 SC3 和 SC4。在多核系统中，OS-Applications 始终可用。

On single-core systems OS-Applications are available only for SC3 and SC4. In Multi-core

systems OS-Applications are always available.

3.3.2 Service Protection

由于 OS-Applications 可以通过服务与 OS 进行交互，因此服务调用必须不会损坏 OS 模块本身。服务保护在运行时防止此类损坏。

As OS-Applications can interact with the OS module through services, it is essential that the service calls will not corrupt the Operating System module itself. Service Protection guards against such corruption at runtime.

对于服务保护，考虑了以下几种情况：

- 参数无效或超出范围；
- 在不合理的上下文调用 API;
- 具有未定义行为的服务;
- 不可信应用中的服务限制;
- 不同 OS-Applications 之间的 Objects 调用。

There are a number of cases to consider with Service Protection:

- Invalid Object Parameter or Out of Range Value.
- Service Calls Made from Wrong Context.
- Services with Undefined Behavior.
- Service Restrictions for Non-Trusted OS-Applications.
- Service Calls on Objects in Different OS-Applications.

参数无效或超出范围：系统 API 的对象无效（比如，该对象未被配置），则返回 E_OS_ID，参数超出范围（比如，设置一个 Alarm 但 cycle 参数小于 OsCounterMinCycle）则返回 E_OS_VALUE。

Invalid Object Parameter or Out of Range Value: The OS' service calls return E_OS_ID on invalid objects (i.e. objects not defined in the OIL file) and E_OS_VALUE for out of range values (e.g. setting an alarm cycle time less than OsCounterMinCycle).

在不合理的上下文调用 API：在 Specification of Operating System AUTOSAR Release 4.2.2 Tab. 1 中定义了，各个 API 允许在哪些场景下调用，如果不符合即在错误的上下文下调用则 API 返回 E_OS_CALLEVEL。

Service Calls Made from Wrong Context: In Specification of Operating System AUTOSAR Release 4.2.2 Tab. 1 shows Allowed Calling Context for OS Service Calls. When these operating system APIs are called in an illegal context, the OS does not perform functions, and an API with

a return value will return an E_OS_CALLEVEL error.

具有未定义行为的服务：

1) 在任务结束时未调用 TerminateTask()或 ChainTask()，系统会自动执行终结任务。
2) 如果配置了 PostTaskHook，如果系统执行 ShutdownOS()就不会再执行 PostTaskHook()。
3) 如果在任务或二类中断中调用了 EnableAllInterrupts() / ResumeAllInterrupts() / ResumeOSInterrupts()，但是在这之前并没有调用相对应的 DisableAllInterupts() / SuspendAllInterrupts() / SuspendOSInterrupts()，那么这些接口将不会真正执行。
4) 如果 Task/ISR/Hook 中在中断被挂起或关闭情况下调用系统 API，那么这些 API 将不会被执行，如果 API 可以返回状态的话就返回 E_OS_DISABLEDINT。
5) 如果任务或中断中占用了一个资源，但执行系统调度时检测到资源需要被释放但未被释放，则 OS 会自动释放该资源。

Services with Undefined Behavior：

1) If Tasks ends without calling a TerminateTask() or ChainTask(), the OS will automatically execute the termination task.
2) If the PostTaskHook() is configured, the OS will not call the hook if ShutdownOS() is called.
3) If EnableAllInterrupts() / ResumeAllInterrupts() / ResumeOSInterrupts() are called and no corresponding DisableAllInterupts() / SuspendAllInterrupts() / SuspendOSInterrupts() was done before, the OS module will not perform this OS service.
4) If interrupts are disabled/suspended by a Task/ISR/Hook and the Task/ISR/Hook calls any OS service (excluding the interrupt services) then the OS module shall ignore the service AND shall return E_OS_DISABLEDINT if the service returns a StatusType value.
5) If a resource is occupied in a task or interrupt, but it is detected that the resource needs to be released but has not been released during system scheduling, the OS will automatically release the resource.

不可信应用中的服务限制：OS 会忽略来自不可信应用对 ShutdownOS()的调用。

Service Restrictions for Non-Trusted OS-Applications: The OS will ignore calls to ShutdownOS() from non-trusted OS-Applications.

不同 OS-Applications 之间的 Objects 调用：如果 OS API 的参数是一个系统对象，但是该对象未被配置响应任务或中断的访问权限，那么系统 API 将返回 E_OS_ACCESS。

Service Calls on Objects in Different OS-Applications: If an OS-object identifier is the parameter of an OS module's system service, and no sufficient access rights have been assigned to this OS-object at configuration time (Parameter Os[...]AccessingApplication) to the calling Task/C2 ISR, the OS module's system service shall return E_OS_ACCESS.

3.3.3 Memory Protection

每个任务或二类中断都需要访问一些数据，正常运行时的访问数据范围可以事先确定。当任务或二类中断发生错误时，可能会跨越访问范围，从而导致其他任务或二类中断出现问题。

Each task or C2 ISR needs to access some data, and the range of access data when running normally can be determined in advance. When an error occurs in a task or C2 ISR, it is possible to cross the access range, causing problems with other tasks or C2 ISR.

为此，OS 提供内存保护功能，为每个任务/二类中断/OS-Application 配置指定的内存区域，并在访问越界时触发保护中断，从而确保系统的正常运行。

For this purpose, the OS provides the Memory Protection function, which configures the specified memory area for each task/C2 ISR/OS-Application, and triggers a protection interrupt when the access is out of bounds, thus ensuring the normal operation of the system.

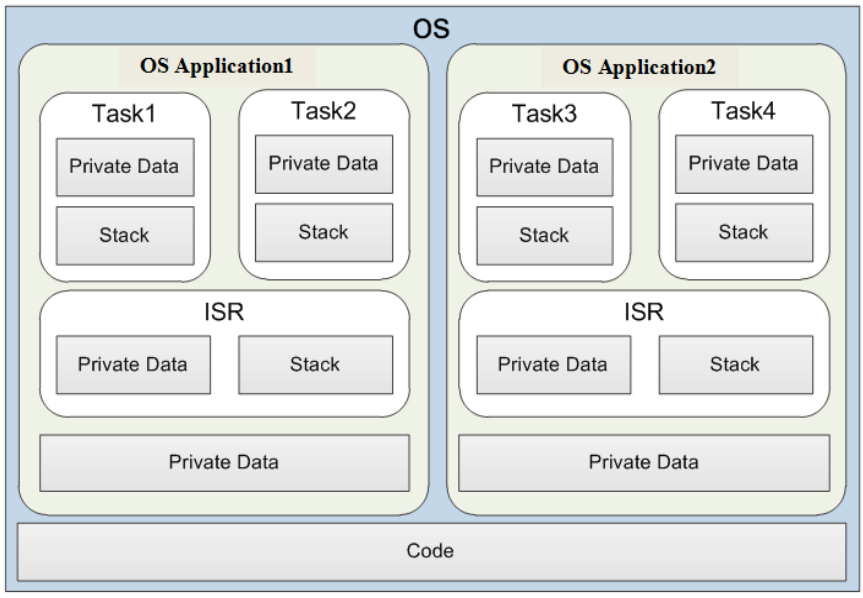


Figure3-1 Memory type

每个任务/二类中断都有一个私有数据和堆栈空间，其他任务/中断无法访问这些数据和堆栈空间。OS-Application 可以配置多个私有数据区域，当前 OS-Application 中的任务/中断可以访问这些区域，其他 OS-Application 的任务/中断无法访问。OS-Application 还可以配置多个共享数据区域，当前 OS-Application 中的任务/中断可以访问这些区域，配

置了相同共享区域的其他 OS-Application 的任务/中断可以访问这些共享区域。

Each task/type 2 interrupt has a private data and stack space, and other tasks/interrupts cannot access these data and stack space. The OS-Application can have multiple private data areas, which can be accessed by tasks/interrupts in the current OS-Application, but cannot be accessed by other OS-Application tasks/interrupts. The OS-Application can also have multiple shared data areas, which can be accessed by tasks/interrupts in the current OS-Application, and can also be accessed by tasks/interrupts in other OS-Application which have the same shared code areas.

支持代码执行保护功能。OS-Application 可以配置多个私有代码保护区，当前 OS-Application 中的任务/中断可以执行这些区域中的代码，其他 OS-Application 的任务/中断无法执行。OS-Application 还可以配置多个共享代码保护区，当前 OS-Application 中的任务/中断可以执行这些区域中的代码，配置了相同共享区域的其他 OS-Application 的任务/中断也可以执行这些区域中的代码。若未配置代码保护区（私有或共享），则不对代码的执行做任何限制。

Code protection is supported. The OS-Application can have multiple private code areas, which can be executed by tasks/interrupts in the current OS-Application, but cannot be executed by other OS-Application tasks/interrupts. The OS-Application can also have multiple shared code areas, which can be executed by tasks/interrupts in the current OS-Application, and can also be executed by tasks/interrupts in other OS-Application which have the same shared code areas.

注 1：可信的 OS-Application 不受内存保护的限制。

注 2：用户调用 OS API 时，不受内存保护的限制。

Note 1: Trusted OS-Application is not restricted by memory protection.

Note 2: When users call OS API, they are not restricted by memory protection.

内存保护的具体实现，与硬件的 MPU 单元及一些特权模式相关，更多信息可参见具体芯片的使用说明。

The specific implementation of memory protection is related to the hardware MPU unit and some privileged modes. For more information, please refer to the specific user manual of different hardware chips.

[Notice]

根据 AUTOSAR 的要求，一类中断不在 OS 管理范围内。因此一类中断不受内存保护的限制，与 OS 内核具有相同的权限。用户应为一类中断分配最高的安全等级。

According to the requirements of AUTOSAR, C1 interrupt is not within the scope of OS

management. Therefore, C1 interrupt is not restricted by memory protection and has the same permission with the OS kernel. The user should assign the highest safety level to a class of interrupts.

3.3.4 IOC

由于内存保护的存在，不同的 OS-Application 之间传输数据是受限制的。但在某些系统中，仍需要在不同的 OS-Application 间进行数据交互。因此，OS 提供了 IOC 通道，实现 OS-Application 间的数据传递。

Due to the existence of memory protection, data transfer between different OS-Applications is restricted. However, in some systems, data interaction between different OS-Applications is still required. Therefore, the OS provides an IOC channel to realize data transfer between OS and Application.

在一些多核处理器上，每个核包含不同的 OS-Applications，因此核间通信是跨操作系统 OS-Applications 通信的一种情况。操作系统将核心间通信能力整合到 IOC 中。

On some multi-core processors, each core contains different OS-Applications, so inter-core communication is a case of cross-OS-Application communication. The operating system incorporates inter-core communication capabilities into the IOC.

用户可配置 IOC 通道的个数，每个通道中传递的数据的个数、长度和类型。一般的 IOC 通道支持任意数量的发送端和接收端。

Users can configure the number of IOC channels, the number, length and type of data transferred in each channel. The general IOC channel supports any number of senders and receivers.

IOC 通道支持队列。使能队列后，仅支持 1:1 或 N：1，也就是仅支持一个数据接收端。

The IOC channel supports queues. After the queue is enabled, only 1:1 or N:1 is supported, that is, only one data receiver is supported.

IOC 内部会采取下述机制保证传递数据的一致性。

- 单核内传递数据时，采用开关全局中断的方式。
- 多核间传递数据时，会使用自旋锁。

The following mechanisms will be adopted within IOC to ensure the consistency of the transmitted data.

- When transferring data within a single core, the switch global interrupt mode is adopted.

- When transferring data between multiple cores, spinlocks are used.

IOC 的通知需要用户自行实现，比如激活一个任务或触发一个中断。

The notification of IOC needs to be implemented by users, such as activating a task or triggering an interrupt.

OS 的工具会为每个 IOC 的通道生成独立的读写接口，具体可参见 4.2 章节的 IOC 相关的 API。

The OS tool will generate an independent read and write interface for each IOC channel. For details, please refer to the IOC-related API in Section 4.2.

3.3.5 Trusted Function

不可信 OS-Application 不允许访问其他 OS-Application 的数据，但在某些系统中，有一些不可信的 OS-Application 需要使用其他 OS-Application 服务。操作系统为可信应用提供了一种将其某些服务导出到不可信应用的方法。这称为可信函数。

The non-trusted OS-Application does not allow access to the data of the trusted OS-Application. In some systems, there are also non-trusted OS-Applications that need to use services within the trusted OS-Application. The OS provides a way for a trusted OS-Application to export some of its services to a non-trusted OS-Application. This is called a trusted function.

用户可在配置工具上定义可信函数的名称，入参及返回值等信息。OS 提供了统一的 CallTrustedFunction 接口，该函数会进行权限切换的操作，并调用用户自定义的可信函数。用户需自行实现可信函数的功能并确保其可靠性。

The user can define the name of the trusted function on the configuration tool, input parameters and return value and other information. The OS provides a unified CallTrustedFunction interface, this function will perform permission switching operations and call user-defined trusted functions. The user needs to implement the function of the trusted function and ensure its reliability.

3.4 MultiCore

3.4.1 MultiCore Startup

内核的启动方式在很大程度上取决于硬件。通常，硬件只启动一个核心，称为主核心，而其他核心（从核）保持在停止状态，直到它们被软件激活。

The way cores are started depends heavily on the hardware. Typically, the hardware only starts one core, referred as the master core, while the other cores (slaves) remain in halt state

until they are activated by the software.

AUTOSAR 多核操作系统规范要求系统具有主从启动行为，可以由硬件直接支持，也可以由软件模拟。主核被定义为不需要软件激活的核，而从核则需要软件激活。

The AUTOSAR Multi-Core OS specification requires a system with master-slave start-up behavior, either supported directly by the hardware or emulated in software. The master core is defined to be the core that requires no software activation, whereas a slave core requires activation by software.

在多核中，在核上执行 StartOS 之前，必须激活 AUTOSAR 使用的每个从核。根据硬件的不同，可能只能从主设备激活可用内核的一个子集。从核可能会在调用 StartOS 之前激活其他核心。所有属于 AUTOSAR 系统的内核都必须由指定的 AUTOSAR API 函数激活。此外，必须在所有这些核上调用 StartOS 函数。

In Multi-Core configurations, each slave core that is used by AUTOSAR must be activated before StartOS is entered on the core. Depending on the hardware, it may be possible to only activate a subset of the available cores from the master. The slave cores might activate additional cores before calling StartOS. All cores that belong to the AUTOSAR system have to be activated by the designated AUTOSAR API function. Additionally, the StartOS function has to be called on all these cores.

如果核心被激活，它会在调用“main”函数之前执行一些硬件和编译器特定的操作。如果在每个核心上执行相同的“main”函数，则必须通过函数中的特定核心 Id 来区分核心。

If a core is activated it executes some HW and compiler specific operations, before the "main" function is called. In case the same "main" function is executed on each core, the cores have to be differentiated by their specific core Id within the function.

如果调用 StartCore 但不调用 StartOS 可能会导致系统挂起。集成时要避免此类行为。

StartCore but do not call StartOS may cause the system to hang. It is in the responsibility of the integrator to avoid such behavior.

StartCore 激活的所有核心都应调用 StartOS 函数。不允许从已经调用 StartOS 的核心调用 StartCore。

The StartOS function shall be called on all cores that have been activated by StartCore. It is not allowed to call StartCore from a core that has already called StartOS.

在 StartOS 的最后，会进行多核启动的同步。所有核都启动后，才会同步触发每个核的首次的任务调度。

At the end of StartOS, the synchronization of multi-core startup will be performed. After all cores are started, the first task scheduling of each core will be triggered synchronously.

3.4.2 MultiCore Shutdown

AUTOSAR 支持两种系统关闭概念，同步关闭和单独关闭概念。同步关闭由 API 函数 ShutdownAllCores()触发，而单个关闭由 API 函数 ShutdownOS()调用。

AUTOSAR supports two shutdown concepts, the synchronized shutdown and the individual shutdown concept. While the synchronized shutdown is triggered by the API function ShutdownAllCores(), the individual shutdown is invoked by the API function ShutdownOS().

如果具有适当权限的任务调用“ShutdownAllCores”，则会向所有其他 Core 发送一个信号，以引导关闭过程。一旦内核上的关闭过程启动，中断和任务就不会被进一步处理，也不会进行任何调度，因此激活任何任务是没有意义的，但是不会生成错误。在调用“ShutdownAllCores”之前，应用程序开发人员有责任确保在应用程序和基本软件级别完成任何关闭准备(比如通过 ECU 管理模块）。

If a TASK with the proper rights calls “ShutdownAllCores”, a signal is sent to all other cores to induce the shutdown procedure. Once the shutdown procedure has started on a core, interrupts and TASKs are not further processed, and no scheduling will take place, therefore it makes no sense to activate any TASK, however no error will be generated. It is in the responsibility of the application developer to make sure that any preparations for shutdown on application and basic software level are completed before calling “ShutdownAllCores”. (e.g. by means of the ECU state manager).

在单核情况下，如果一个任务调用了 ShutdownOS，那么内核上的操作系统将被关闭。多核情况下，不支持通过调用 ShutdownOS 单独关闭一个核。

In the case of single core, if a TASK calls ShutdownOS the OS will be shut down on the core. In the case of multi-core, it is not supported to close a single core by calling ShutdownOS.

3.4.3 SpinLock

在多核概念下，需要一种新的机制来支持不同核心上的任务互斥。这种新机制不应在同一核心上的任务之间使用，因为这是毫无意义的。

With the Multi-Core concept, a new mechanism is needed to support mutual exclusion for TASKS on different cores. This new mechanism shall not be used between TASKs on the same

core because it makes no sense.

Spinlock 是一种繁忙的等待机制，它轮询（lock）变量直到它变为可用。

A Spinlock is a busy waiting mechanism that polls a (lock) variable until it becomes available.

OS 提供了一个 GetSpinlock()函数，该函数占用一个 Spinlock。如果 Spinlock 已经被占用，GetSpinlock()将继续尝试占用 Spinlock，直到成功为止。

The OS provide a GetSpinlock() function which occupies a Spinlock. If the Spinlock is already occupied, GetSpinlock() shall keep on trying to occupy the Spinlock until it succeeds.

OS 提供了一个 TryToGetSpinlock()函数，该函数占用一个 Spinlock。如果 Spinlock 已经被占用，TryToGetSpinlock()将返回。

The OS provide a TryToGetSpinlock() function which occupies a Spinlock. If the Spinlock is already occupied, TryToGetSpinlock() will return.

OS 提供了一个 ReleaseSpinlock()函数，用于释放被占用的自旋锁。如果自旋锁未被占用，则返回一个错误。

The OS provide a ReleaseSpinlock() function which releases an occupied Spinlock. If the Spinlock is not occupied an error will be returned.

如果多个 Spinlock 存在嵌套使用的情况，需要在工具中配置其使用顺序，否则可能造成死锁的风险。

If multiple Spinlocks are nested, users need to configure their use order in the tool, otherwise it may cause the risk of deadlock.

Spinlock 的实现与硬件强相关，具体使用了硬件哪种功能，请参考芯片相关的 OS 使用说明。

The implementation of Spinlock is strongly related to the hardware. Please refer to the OS User Manual related to the chip for specific functions of the hardware.

3.4.4 Cross Core Service

OS 中的跨核操作需基于芯片实现，一般使用核间中断，具体请参考芯片相关的 OS 使用说明。

The cross-core operation in OS needs to be implemented based on chip, and inter core interrupt is generally used. For details, please refer to the OS User Manual related to the chip.

支持跨核操作的接口如下所示：

The interfaces supporting cross core operations are as follows:

- ActivateTask
- CancelAlarm
- CheckObjectAccess
- CheckObjectOwnership
- ControlIdle
- GetAlarm
- GetAlarmBase
- StartNonAutosarCore
- GetEvent
- TerminateApplication
- GetTaskState
- SetAbsAlarm
- SetEvent
- SetRelAlarm
- ShutdownAllCores
- StartCore

如果跨核动作超时，OS 会执行 ErrorHook()。

If the cross core action timed out, the OS will execute ErrorHook ().

3.5 Performance Parameters

OS 资源占用量，运行时间等性能参数，需要在指定的平台和环境下测得，参见芯片相关的 OS 的使用说明。

Performance parameters such as OS resource occupation and running time need to be measured under the specified platform and environment. Refer to the user manual of the OS related to the chip.

3.6 Unsupported Features

当前 OS 不支持的功能包括：

- 不支持内部资源、链接资源。
- 不支持调度表的同步。
- 不支持 SuspendOSInterrupts/ResumeOSInterrupts 和

SuspendAllInterrupts/ResumeAllInterrupts 这两对 API 的互相嵌套。

- 不支持添加 Hardware 类型的 Counter。
- 不支持 OsTrustedApplicationWithProtection 配置项，即可信应用具有最高权限，不支持内存保护。
- 不支持在从核上调用 StartCore()。
- 除 3.4.4 之外的系统 API 都不支持支持跨核操作。

The functions that the current OS does not support include：

- Internal and linked resources are not supported.
- The synchronization function of schedule table is not supported.
- The nesting of two pairs of APIs, SuspendOSInterrupts/ResumeOSInterrupts and SuspendAllInterrupts/ResumeAllInterrupts, is not supported.
- Adding counter of hardware type is not supported.
- The OsTrustedApplicationWithProtection configuration item is not supported, that is, the Trusted-Application has the highest permission and does not support memory protection.
- Calling StartCore() on a slave core is not support.
- The system APIs except 3.4.4 do not support cross core operations.

3.7 Memory Section

OS 本身包含的 Memory Section 如下图。用户应基于实际项目需求，对 OS 的代码和变量进行定位。

The Memory Section contained in the OS itself is shown in the figure below. Users should locate the OS code and variables based on actual project requirements.

Section	Description
CODE	该段用于存放 OS 的核心代码。 This section is used for storing the core code of the OS.
ISR	该段用于存放 OS 的中断服务函数。 This section is used for storing the ISR.
Callout	该段用于存放 OS 的 Callout 函数 This section is used for storing the callout function of the OS.
CONFIG_DATA	该段用于存放 OS 的配置数据。 This section is used for storing the configuration data of the OS.
RAM	该段用于存放 OS 的内核变量。

	This section is used for storing core variables of the OS.
CONST	该段用于存放 OS 的常量。 This section is used for storing constant of the OS.
STACK	该段用于存放任务栈和中断栈。 This section is used for storing stack of task and ISR.

4 API

4.1 Service API

4.1.1 StartOS

Syntax			
void StartOS(AppModeType Mode)			
Parameters			
Mode	应用模式名称 application mode		
Return Value			
void	none		
Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
当前< CoreID >为无效 ID Current< CoreID > is invalid Mode is not valid. 多核时，传入的 Mode 不一致。 When multi-core, the input Mode is inconsistent	当前 OS 还未启动，无法记录错误信息。若此时 OS 检查到错误，OS 会进入死循环。 Now OS has not been started, and error messages cannot be recorded. If the OS detects an error at this time, the OS will enter an endless loop.		
Description			
用户通过调用该项系统服务按某种设定的应用模式启动操作系统。 The user can call this system service to start the operating system in a specific mode.			
Call Opportunity			
系统启动时调用。如果是多核，每个核都需要调用 StartOS。 Called when the system starts. If it is multi-core, each core needs to call StartOS.			
Remark			
1.See 3.1.1 for more details. 2.多核启动时，在 StartOS 的最后会进行同步。如果无法同步，OS 会进入死循环。 When multi-core is started, it will be synchronized at the end of StartOS. If it cannot be synchronized, the OS will enter an endless loop.			

4.1.2 ShutdownOS

Syntax			
void ShutdownOS (StatusType Error)			
Parameters			
Error	发生错误的错误标识 error occurred		
Return Value			
void	none		
Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			

None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par =Os_CurrentApplicationId
Current OS-Application is not Trusted	E_OS_ACCESS	OS_IE_SHUTDOWN_BY_UNTRUST_APP	None

Description
用户可以通过调用该系统服务来关闭整个系统；一旦系统发生严重错误无法继续正常运行或是运行到未知的状态，操作系统会自行调用该项系统服务函数来关闭整个系统。 The user can call this system service to abort the overall system. The operating system also calls this function internally, if it has reached an undefined internal state or fatal error detected.
Call Opportunity
None
Remark
如果对此服务函数配置了ShutdownHook，那么每次系统关闭之前都将先运行ShutdownHook然后再关闭系统； If a ShutdownHook is configured the hook routine ShutdownHook is always called before shutting down the operating system;

4.1.3 ActivateTask

Syntax	
StatusType ActivateTask (TaskType TaskID)	
Parameters	
TaskID	任务名称 Task reference
Return Value	
StatusType	若成功调用，返回 E_OS_OK; No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
若超出<TaskID>的多次激活上限	E_OS_LIMIT	OS_IE_TASK	Par= TaskID

Too many task activations of <TaskID>		_ACTIVE_TIMES_ERROR	
当前任务所属的 Application 已经被终结 The application to which the current task belongs has been killed	E_OS_ACCESS	OS_IE_OBJECT_KILLED	Par= TaskID
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
若<TaskID>为无效 ID Task <TaskID> is invalid	E_OS_ID	OS_IE_TASK_ID	Par= TaskID
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par =Os_Current ApplicationId
当前的 Application 对该 TASK 没有访问权限 The current application does not have access permission to the task	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par =Os_Current ApplicationId Par1= TaskID

Description

激活指定任务。

如果当前任务可抢占，且新激活的任务优先级更高，则会触发任务调度，切换至新任务运行。

Activate the specified task.

If the current task can be preempted and the newly activated task has a higher priority, the task scheduling will be triggered and the new task will be switched to run.

Call Opportunity

None

Remark

--

4.1.4 TerminateTask

Syntax	
StatusType TerminateTask (void)	
Parameters	
void	none
Return Value	
StatusType	若成功调用，不返回调用层； No return to call level; 其他情况参考 error report。

	For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
当前任务激活次数未 0。 The number of current task activations is 0.	E_OS_LIMIT	OS_IE_TASK_ACTIVE_TIMES_ERROR	Par= tempTaskId
自动恢复全局中断状态 Auto resume global interrupt state	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
自动释放 Task 的资源 Auto release task resource	E_OS_RESOURCE	OS_IE_NOMORE_INFO	None
自动释放被 Task 锁定的 Spinlock Auto release spinlock locked by Task	E_OS_SPINLOCK	OS_IE_NOMORE_INFO	None
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_TASK_ID	Par= TaskID
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
当前任务激活次数未 0。 The number of current task activations is 0.	E_OS_LIMIT	OS_IE_TASK_ACTIVE_TIMES_ERROR	None

Description

此服务函数终止当前运行的任务。触发任务调度，切换至下一个优先级最高的任务运行。
This service terminates the running task. Trigger task scheduling and switch to the next highest priority task to run.

Call Opportunity

At the end of user task function.

Remark

1.该 API 会自动释放被任务占用的资源、自旋锁，恢复全局中断状态。
The API will automatically release the resources and spin locks occupied by the task, and restore the global interrupt state.
2.任务运行结束时，必须调用 TerminateTask 或者 ChainTask 终结当前任务。如用户忘记调用，OS 内部会调用 TerminateTask。
When the task is finished, user must call TerminateTask or ChainTask to terminate the current task. If the user forgets to call, the OS will call TerminateTask internally.

4.1.5 ChainTask

Syntax

StatusType ChainTask (TaskType TaskID)

Parameters

TaskID	后续需要激活的任务名称 Reference to the sequential succeeding task to be activated.

Return Value	
StatusType	若成功调用，不返回调用层； No return to call level; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
若超出<TaskID>的多次激活上限 Too many task activations of <TaskID>	E_OS_LIMIT	OS_IE_TASK_ACTIVE_TIMES_ERROR	Par= TaskID
<TaskID>所属的 Application 已经被终结 The application to which <TaskID> belongs has been killed	E_OS_ACCESS	OS_IE_OBJECT_KILLED	Par= TaskID
当前任务激活次数未 0。 The number of current task activations is 0.	E_OS_LIMIT	OS_IE_TASK_ACTIVE_TIMES_ERROR	Par= tempTaskId
自动恢复全局中断状态 Auto resume global interrupt state	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
自动释放 Task 的资源 Auto release task resource	E_OS_RESOURCE	OS_IE_NOMORE_INFO	None
自动释放被 Task 锁定的 Spinlock Auto release spinlock locked by Task	E_OS_SPINLOCK	OS_IE_NOMORE_INFO	None
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
若<TaskID>为无效 ID Task <TaskID> is invalid	E_OS_ID	OS_IE_TASK_ID	Par=TaskID
若< Os_CurrentTaskId >为无效 ID Task < Os_CurrentTaskId > is invalid	E_OS_ACCESS	OS_IE_TASK_ID	Par=TaskID
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par=TaskID
当前任务激活次数未 0。 The number of current task activations is 0.	E_OS_LIMIT	OS_IE_TASK_ACTIVE_TIMES_ERROR	None
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par =Os_Current ApplicationId
当前的 Application 对该 TASK 没有	E_OS_ACCESS	OS_IE_NO_A	Par

访问权限 The current application does not have access to the task		CCESS_RIGHT	=Os_CurrentApplicationId Par1= TaskID

Description
该项系统服务终止当前运行的任务，同时任务<TaskID>将被激活。 This service terminates of the running task, and activate task <TaskID>. 该API会触发任务调度，切换至下一个优先级最高的任务运行。 This service will trigger task scheduling and switch to the next highest priority task to run.
Call Opportunity
None
Remark
1.该 API 会自动释放被任务占用的资源、自旋锁，恢复全局中断状态。 The API will automatically release the resources and spin locks occupied by the task, and restore the global interrupt state. 2.任务运行结束时，必须调用 TerminateTask 或者 ChainTask 终结当前任务。如用户忘记调用，OS 内部会调用 TerminateTask。 When the task is finished, user must call TerminateTask or ChainTask to terminate the current task. If the user forgets to call, the OS will call TerminateTask internally. 3.若后续任务的 ID 和当前要终止的任务一致，这不属于任务的多次激活。 If the ID of the subsequent task is the same as the current task to be terminated, this does not belong to multiple activations of the task.

4.1.6 Schedule

Syntax	
StatusType Schedule(void)	
Parameters	
void	none
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用	E_OS_DISABLE	OS_IE_NOM	None

API is called between interrupt processing APIs	DINT	ORE_INFO	

Description

该函数强制触发一次任务调度。
This function forces a task scheduling to be triggered.

Call Opportunity

None

Remark

如果当前运行的任务是不可抢占任务，该函数同样会触发调度并切换至更高优先级的任务。
If the currently running task is a non-preemptible task, this function will also trigger the scheduling and switch to a higher priority task.

4.1.7 GetTaskID

Syntax

StatusType GetTaskID(TaskRefType TaskID)

Parameters

TaskID	Output the current running TaskID

Return Value

StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若<TaskID>为空指针 <TaskID> is null	E_OS_PARAM_POINTER	OS_IE_NOMORE_INFO	None

Description

该项系统服务返回当前正在运行的任务的ID信息。
GetTaskID returns the information about the TaskID of the task which is currently running.

Call Opportunity

None

Remark

None

4.1.8 GetTaskState

Syntax	
StatusType GetTaskState (TaskType TaskID, TaskStateRefType State)	
Parameters	
TaskID(in)	任务名称 Task reference
State(out)	<TaskID> 所标识的任务状态 Reference to the state of the task <TaskID>
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若<TaskID>为无效 ID Task <TaskID> is invalid	E_OS_ID	OS_IE_TASK_ID	Par=TaskID
若<State>为空指针 Task<State> is null	E_OS_PARAM_POINTER	OS_IE_NOMORE_INFO	None
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par=TaskID
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par =Os_Current ApplicationId
当前的 Application 对该 TASK 没有访问权限 The current application does not have access to the task	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par =Os_Current ApplicationId Par1= TaskID

Description
返回<TaskID>所标识的任务在此函数被调用时刻的状态 (running, ready, waiting, suspended)

Returns the state of a task (running, ready, waiting, suspended) at the time of calling GetTaskState.
Call Opportunity
None
Remark
None

4.1.9 EnableAllInterrupts

Syntax	
void EnableAllInterrupts (void)	
Parameters	
void	none
Return Value	
StatusType	none

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
中断 API 的调用顺序错误。 The calling sequence of the interrupt API is wrong.	E_OS_DISABLEDINT	None	None
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None

Description
此服务函数重新装载由DisableAllInterrupts保存的系统中断状态。 This service restores the state saved by DisableAllInterrupts.
Call Opportunity
None
Remark
1. 该项系统服务要和 DisableAllInterrupts 配对使用； This service is a counterpart of DisableAllInterrupts service, which has to be called before, and its aim is the completion of the critical section of code; 2. 不允许在 DisableAllInterrupts 和 EnableAllInterrupts 之间调用任何系统服务函数； No API service calls are allowed within this critical section; 3. DisableAllInterrupts 和 EnableAllInterrupts 这对 API 不支持嵌套调用。 DisableAllInterrupts/EnableAllInterrupts does not support nesting.

4.1.10 DisableAllInterrupts

Syntax	
void DisableAllInterrupts(void)	
Parameters	
void	none

Return Value	
StatusType	none

Description

此服务函数关闭所有中断，关闭前系统的中断状态将被保存。
This service function disable all interrupts, the interrupt status of the system will be saved before disable.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
中断 API 的调用顺序错误。 The calling sequence of the interrupt API is wrong.	E_OS_DISABLEDINT	None	None
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None

Call Opportunity

None

Remark

1. 该项系统服务要和 EnableAllInterrupts 配对使用；
   This service is intended to start a critical section of the code. This section shall be finished by calling the EnableAllInterrupts service;
2. 不允许在 DisableAllInterrupts 和 EnableAllInterrupts 之间调用任何系统服务函数；
   No API service calls are allowed within this critical section;
3. DisableAllInterrupts 和 EnableAllInterrupts 这对 API 不支持嵌套调用。
   DisableAllInterrupts/EnableAllInterrupts does not support nesting.

4.1.11 ResumeAllInterrupts

Syntax

void ResumeAllInterrupts(void)

Parameters

void	none

Return Value	
StatusType	none

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
中断 API 的调用顺序错误。 The calling sequence of the interrupt API is wrong.	E_OS_DISABLEDINT	None	None
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None

Description

此服务函数被调用时将装载由SuspendAllInterupts保存的系统中断状态信息。

This service restores the status of all interrupts saved by the SuspendAllInterrupts service.
Call Opportunity
None
Remark
1. 该项系统服务要和 SuspendAllInterrupts 配对使用； This service is the counterpart of SuspendAllInterrupts service, which has to have been called before; 2. SuspendAllInterrupts 和 ResumeAllInterrupts 这对 API 支持嵌套调用。 SuspendAllInterrupts/ResumeAllInterrupts can be nested.

4.1.12 SuspendAllInterrupts

Syntax	
void SuspendAllInterrupts(void)	
Parameters	
void	none
Return Value	
StatusType	none

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
中断 API 的调用顺序错误。 The calling sequence of the interrupt API is wrong.	E_OS_DISABLEDINT	None	None
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None

Description
此服务函数关闭所有中断，关闭前系统的中断状态将被保存。 This service function disable all interrupts, the interrupt status of the system will be saved before disable.
Call Opportunity
None
Remark
1. 该项系统服务要和 ResumeAllInterrupts 配对使用； This service is the counterpart of ResumeAllInterrupts service, it shall be finished by calling the ResumeAllInterrupts service; 2. SuspendAllInterrupts 和 ResumeAllInterrupts 这对 API 支持嵌套调用。 SuspendAllInterrupts/ResumeAllInterrupts can be nested.

4.1.13 ResumeOSInterrupts

Syntax	
void ResumeOSInterrupts(void)	
Parameters	
void	none

Return Value	
StatusType	none

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
中断 API 的调用顺序错误。 The calling sequence of the interrupt API is wrong.	E_OS_DISABLEDINT	None	None
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None

Description

此服务函数被调用时将装载由SuspendOSInterrupts保存的系统中断状态信息。
This service restores the status of interrupts saved by the SuspendOSInterrupts service.

Call Opportunity

None

Remark

1. 此服务函数要和 SuspendOSInterrupts 配对使用；
   This service is the counterpart of SuspendOSInterrupts service, which has to have been called before;
2. ResumeOSInterrupts 和 SuspendOSInterrupts 这对 API 支持嵌套调用。
   SuspendOSInterrupts/ResumeOSInterrupts can be nested.

4.1.14 SuspendOSInterrupts

Syntax

void SuspendOSInterrupts(void)

Parameters

void	none

Return Value

StatusType	none

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
中断 API 的调用顺序错误。 The calling sequence of the interrupt API is wrong.	E_OS_DISABLEDINT	None	None
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None

Description

此服务函数用来关闭系统内所有第二类中断，关闭前系统的中断状态将被保存。
This service saves the recognition status of interrupts of category 2 and disables the recognition of these interrupts.

Call Opportunity

None
Remark
1. 此服务函数要和 ResumeOSInterrupts 配对使用； This service is the counterpart of ResumeOSInterrupts service, it shall be finished by calling the ResumeOSInterrupts service; 2. ResumeOSInterrupts 和 SuspendOSInterrupts 这对 API 支持嵌套调用。 SuspendOSInterrupts/ResumeOSInterrupts can be nested.

4.1.15 GetResource

Syntax	
StatusType GetResource (ResourceType ResID)	
Parameters	
ResID	资源 ID Reference to resource ID
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
资源已经被占用。 Resources have been occupied.	E_OS_ACCESS	None	None
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若<ResID>为无效 ID <ResID> is invalid	E_OS_ID	OS_IE_RES_ID	Par=ResID
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par=ResID
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par =Os_CurrentApplicationId
当前的 Application 对该 RES 没有访问权限 The current application does not have	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par= ResID Par1=Os_CurrentApplicati

access to the RES			onId
在任务中获取中断资源 Obtain ISR resources in task	E_OS_ACCESS	OS_IE_ISRRES_ACCBY_TASK	Par=ResID Par1=tempTaskId
在中断中获取任务资源 Obtain task resources in ISR	E_OS_ACCESS	OS_IE_TASKRES_ACCBY_ISR	Par=ResID Par1=tempIsrId

Description
此服务函数用于占用<ResID>。 This service function is used to occupy <ResID>.
Call Opportunity
None.
Remark
1.占用资源的时间应尽可能短。 Generally speaking, critical sections should be short。 2.Refer to 3.1.7 for more details.

4.1.16 ReleaseResource

Syntax	
StatusType ReleaseResource (ResourceType ResID)	
Parameters	
ResID	资源名称 Reference to resource
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若<ResID>为无效 ID <ResID> is invalid	E_OS_ID	OS_IE_RES_ID	Par=ResID
API 不支持跨核调用。	E_OS_SYS_SER	OS_IE_NOM	Par=ResID

The API does not support cross-core calls.	VICE_NOTSUPPORT_CROSSCORE	ORE_INFO	
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par =Os_CurrentApplicationId
当前的 Application 对该 RES 没有访问权限 The current application does not have access to the RES	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par= ResID Par1=Os_CurrentApplicationId
尝试释放一个未被任何任务或中断占用的资源，或者该资源之前已经被释放了 Attempt to release a resource which is not occupied by any task or ISR, or the resource shall be released before.	E_OS_NOFUNC	OS_IE_NOMORE_INFO	Par=ResID
资源释放的顺序错误 The order of resource release is wrong.	E_OS_ACCESS	OS_IE_RES_LIFO_ERROR	Par=ResID Par1= tempTaskId
资源释放的顺序错误 The order of resource release is wrong.	E_OS_ACCESS	OS_IE_RES_LIFO_ERROR	Par=ResID Par1= tempIsrId

Description
此服务函数用于释放<ResID>。 This service function is used to release <ResID>.
Call Opportunity
None
Remark
Refer to 3.1.7.

4.1.17 SetEvent

Syntax	
StatusType SetEvent (TaskType TaskID, EventMaskType Mask)	
Parameters	
TaskID	单个或多个事件所属的扩展任务名称 Reference to the task for which one or several events are to be set.
Mask	需要设置的事件，可以是单个事件的名称或多个事件的名称相或，如 SetEvent（Task1, Event1 | Event 2 | Event3） Mask of the events to be set.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若被标识的任务<TaskID>为无效 ID Referenced task <TaskID> is invalid	E_OS_ID	OS_IE_TASK_ID	Par=TaskID
若被标识的任务不是扩展任务 Referenced task is no extended task	E_OS_ACCESS	OS_IE_EVENT_WORK_WITH_BT	Par=TaskID
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par =Os_CurrentApplicationId
当前的 Application 对该 Task 没有访问权限 The current application does not have access to the task	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par= TaskID

Description

设置扩展任务的事件。如果指定扩展任务正在等待该事件，则扩展任务将会进入就绪态。如果当前任务可抢占，且指定扩展任务的优先级更高，则会触发任务调度，切换至新任务运行。

Activate the specified task. If the specified extended task is waiting for this event, the extended task will enter the ready state.

If the current task can be preempted and the newly activated task has a higher priority, the task scheduling will be triggered and the new task will be switched to run.

Call Opportunity

None

Remark

4.1.18 ClearEvent

Syntax	
StatusType ClearEvent (EventMaskType Mask)	
Parameters	
Mask	需要清除的事件，可以是单个事件的名称或多个事件的名称相或，如 ClearEvent（Task1, Event1 | Event 2 | Event3） Mask of the events to be cleared.

Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
当前任务无效。 Current task is invalid.	E_OS_ACCESS	OS_IE_EVENT_ERRTASK	Par=Os_CurrentTaskId
当前任务不是扩展任务 Current task is not extended task	E_OS_ACCESS	OS_IE_EVENT_WORK_WITH_BT	Par=Os_CurrentTaskId

Description

扩展任务调用该项系统服务重置该任务所关联的一个或多个事件。
The extended task calls this system service to clear one or more events associated with the task.

Call Opportunity

None

Remark

None

4.1.19 GetEvent

Syntax	
StatusType GetEvent (TaskType TaskID, EventMaskRefType Event)	
Parameters	
TaskID(in)	需要获取所有事件状态的扩展任务名称 Task whose event mask is to be returned.
Event(out)	任务<TaskID>的所有事件状态掩码 Reference to the memory of the return data.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若被标识的任务<TaskID>为无效 ID Referenced task <TaskID> is invalid	E_OS_ID	OS_IE_TASK_ID	Par=TaskID
若< Event >为空指针 Task< Event > is null	E_OS_PARAM_POINTER	OS_IE_NOMORE_INFO	None
若被标识的任务不是扩展任务 Referenced task is no extended task	E_OS_ACCESS	OS_IE_EVENT_WORK_WITH_BT	Par=TaskID
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_TASK_ID	Par=TaskID
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par =Os_Current ApplicationId
当前的 Application 对该 Task 没有访问权限 The current application does not have access to the task	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par= TaskID
如果被标识的任务处于终止状态，则返回 E_OS_STATE If the identified task is in the suspended state, return E_OS_STATE	E_OS_STATE	OS_IE_TASK_ID	Par=TaskID

Description

此服务函数将返回任务<TaskID>的所有事件的当前状态，不仅仅是该任务正在等待的事件；任务<TaskID>的事件状态将通过掩码的形式复制到<Event>当中。

This service function will return the current status of all events of task <TaskID>, not just the events that the task is waiting for; the event status of task <TaskID> will be copied to <Event> in the form of a mask.

Call Opportunity

None

Remark

--

4.1.20 WaitEvent

Syntax	
StatusType WaitEvent (EventMaskType Mask)	
Parameters	
Mask	需要等待的事件，可以是单个事件的名称或多个事件的名称相或，如 WaitEvent（Event1 | Event 2 | Event3） Mask of the events waited for.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
自动恢复全局中断状态 Auto resume global interrupt state	E_OS_DISABLEDINT	OSServiceId_WaitEvent	None
自动释放 Task 的资源 Auto release task resource	E_OS_RESOURCE	OSServiceId_WaitEvent	None
自动释放被 Task 锁定的 Spinlock Auto release spinlock locked by Task	E_OS_SPINLOCK	OSServiceId_WaitEvent	None
Extended			
若< CoreID >为无效 ID < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
当前任务不是扩展任务 Current task is not extended task	E_OS_ACCESS	OS_IE_EVENT_WORK_WITH_BT	Par=TaskID
当前的 Task 无效。 Current task is invalid.	E_OS_ACCESS	OS_IE_EVENT_ERRTASK	Par =CurrentTaskId

Description

调用此服务函数的扩展任务将从运行状态转移到等待状态，触发任务调度，切换至下一个优先级最高的任务运行。

The extended task that calls this service function will move from the running state to the waiting state. Trigger task scheduling and switch to the next highest priority task to run.

若需要等待的一个或多个事件中，至少有一个事件已经被设置，那么此扩展任务将继续执行，不进行状态转换。

If at least one event to be waited for has been set, then this extended task will continue to execute without state transition.

Call Opportunity

None

Remark
1.该 API 会自动释放被任务占用的资源、自旋锁，恢复全局中断状态。 The API will automatically release the resources and spin locks occupied by the task, and restore the global interrupt state.

4.1.21 GetAlarmBase

Syntax	
StatusType GetAlarmBase (AlarmType AlarmID, AlarmBaseRefType Info)	
Parameters	
AlarmID(in)	报警名称 Reference to alarm.
Info(out)	用于存储与报警<AlarmID>相关联的计数器属性的变量 Reference to structure with constants of the alarm base.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若<AlarmID>为无效 ID <AlarmID> is invalid	E_OS_ID	OS_IE_ALARM_ID	Par=AlarmID
若< Info>为空指针 Alarm< Info> is null	E_OS_PARAM_POINTER	OS_IE_NOMORE_INFO	None
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Alarm 没有访问权限 The current application does not have access to the alarm	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=AlarmID Par1=Os_CurrentApplicationId

Description
此服务函数读取与报警<AlarmID>相关联的计数器的基本属性，作为输出参数的变量

<Info>数据类型应为AlarmBaseType。 The system service GetAlarmBase reads the alarm base characteristics. The return value <Info> is a structure in which the information of data type AlarmBaseType is stored.
Call Opportunity
None
Remark
None

4.1.22 GetAlarm

Syntax	
StatusType GetAlarm (AlarmType AlarmID, TickRefType Tick)	
Parameters	
AlarmID(in)	报警名称 Reference to an alarm.
Tick(out)	当前时刻到报警<AlarmID>下次发生，需要经过的 Tick 值 Tick value from the current moment to the next occurrence of the alarm <AlarmID>
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
当前报警所属的 Application 已经被终结 The application to which the current alarm belongs has been killed	E_OS_ACCESS	OS_IE_OBJECT_KILLED	Par=AlarmID
报警<AlarmID>未运行 Alarm <AlarmID> is not running	E_OS_NOFUNC	OS_IE_NOMORE_INFO	Par=AlarmID
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若<AlarmID>为无效 ID <AlarmID> is invalid	E_OS_ID	OS_IE_ALARM_ID	Par=AlarmID
若< Tick >为空指针 Alarm< Tick > is null	E_OS_PARAM_POINTER	OS_IE_NOMORE_INFO	None
API 不支持跨核调用。	E_OS_SYS_SER	OS_IE_NOM	Par=AlarmID

The API does not support cross-core calls.	VICE_NOTSUPPORT_CROSSCORE	ORE_INFO	
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Alarm 没有访问权限 The current application does not have access to the alarm	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=AlarmID Par1=Os_CurrentApplicationId

Description
此服务函数输出报警<AlarmID>当前时刻距离下次发生时刻的时间间隔，以计数器Tick值的形式输出。 The system service GetAlarm returns the relative value in ticks before the alarm <AlarmID> expires.
Call Opportunity
None
Remark
None

4.1.23 SetRelAlarm

Syntax	
StatusType SetRelAlarm (AlarmType AlarmID, TickType Increment, TickType Cycle)	
Parameters	
AlarmID(in)	报警名称 Reference to the alarm element.
Increment(in)	计数器 Tick 相对值 Relative value in ticks.
Cycle(in)	周期性报警的周期值（对于单次报警，周期值应设为零） Cycle value in case of cyclic alarm. In case of single alarms, cycle shall be zero.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
当前报警所属的 Application 已经被终结 The application to which the current alarm belongs has been killed	E_OS_ACCESS	OS_IE_OBJECT_KILLED	Par=AlarmID
若报警<AlarmID>已经激活使用	E_OS_STATE	OS_IE_ALAR	Par=AlarmID

Alarm <AlarmID> is already in use		M_ALREADY_ALIVE	
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若<AlarmID>为无效 ID <AlarmID> is invalid	E_OS_ID	OS_IE_ALARM_ID	Par=AlarmID
若所设置的<increment>的值超出有效范围（等于零或是大于 maxallowedvalue） Value of <increment> outside of the admissible limits (equal to zero or greater than maxallowedvalue)	E_OS_VALUE	OS_IE_ALARM_INCREAMENT	Par=AlarmID Par1=increment Par2=cycle
若设置的<cycle>的值不为 0 并且超出有效范围（小于 mincycle 或者是大于 maxallowedvalue） Value of <cycle> unequal to 0 and outside of the admissible counter limits (less than mincycle or greater than maxallowedvalue)	E_OS_VALUE	OS_IE_ALARM_CYCLE	Par=AlarmID Par1=increment Par2=cycle
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Alarm 没有访问权限 The current application does not have access to the alarm	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=Os_CurrentApplicationId Par1=AlarmID

Description

此服务函数将激活报警<AlarmID>，当tick设定值<Increment>到达后，与该报警关联的报警动作将被执行。

The system service occupies the alarm <AlarmID> element. After <increment> ticks have elapsed, the task assigned to the alarm <AlarmID> is activated or the assigned event (only for extended tasks) is set or the alarm-callback routine is called.

Call Opportunity

None

Remark

1. 若<Increment>为零时报警将会在计数器下一个 Tick 到来时发生；

If <Increment> is zero, the alarm will occur when the next tick of the counter arrives;
2. 若<Cycle>不为零，表示报警为周期性的；
If <Cycle> is not zero, it means the alarm is periodic;
3. 如果需要改变已经激活的报警的属性参数，需要首先取消该报警。
To change values of alarms already in use the alarm shall be cancelled first。

4.1.24 SetAbsAlarm

Syntax	
StatusTypeSetAbsAlarm (AlarmType AlarmID, TickType Start, TickType Cycle)	
Parameters	
AlarmID(in)	报警名称 Reference to the alarm element.
Start(in)	计数器 Tick 绝对值 Absolute value in ticks.
Cycle(in)	周期性报警的周期值（对于单次报警，周期值应设为零） Cycle value in case of cyclic alarm. In case of single alarms, cycle shall be zero.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
当前报警所属的 Application 已经被终结 The application to which the current alarm belongs has been killed	E_OS_ACCESS	OS_IE_OBJECT_KILLED	Par=AlarmID
若报警<AlarmID>已经激活使用 Alarm <AlarmID> is already in use	E_OS_STATE	OS_IE_ALARM_ALREADY_ALIVE	Par=AlarmID
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若<AlarmID>为无效 ID <AlarmID> is invalid	E_OS_ID	OS_IE_ALARM_ID	Par=AlarmID
若所设置的<start>的值超出有效范	E_OS_VALUE	OS_IE_ALAR	Par=AlarmID

围（大于 maxallowedvalue） Value of <start> outside of the admissible limits (greater than maxallowedvalue)		M_START	Par1=start Par2=cycle
若设置的<cycle>的值不为 0 并且超出有效范围（小于 mincycle 或者是大于 maxallowedvalue） Value of <cycle> unequal to 0 and outside of the admissible counter limits (less than mincycle or greater than maxallowedvalue)	E_OS_VALUE	OS_IE_ALARM_CYCLE	Par=AlarmID Par1=start Par2=cycle
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Alarm 没有访问权限 The current application does not have access to the alarm	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=Os_CurrentApplicationId Par1=AlarmID

Description

此服务函数将激活报警<AlarmID>，使该报警的状态变为使用中；当计数器达到<Start>所标识的tick值时，与该报警相关联的报警动作将被执行。

The system service occupies the alarm <AlarmID> element. When <start> ticks are reached, the task assigned to the alarm <AlarmID> is activated or the assigned event(only for extended tasks) is set or the alarm-callback routine is called.

Call Opportunity

None

Remark

1. 若计数器在此服务函数被调用之前就达到 tick 的绝对值<Start>，那么该报警会在计数器加到最大值返回零重新开始累加到<Start>时才会发生；
   If the absolute value <start> already was reached before the system call, the alarm shall only expire when the absolute value <start> is reached again, i.e. after the next overrun of the counter;
2. 若<Cycle>不为零，表示报警为周期性的；
   If <Cycle> is not zero, it means the alarm is periodic;
3. 如果需要改变已经激活的报警的属性参数，需要首先取消该报警。
   To change values of alarms already in use the alarm shall be cancelled first.

4.1.25 CancelAlarm

Syntax

StatusType CancelAlarm(AlarmType AlarmID)

Parameters

AlarmID(in)	报警名称

	Reference to an alarm.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
当前报警所属的 Application 已经被终结 The application to which the current alarm belongs has been killed	E_OS_ACCESS	OS_IE_OBJECT_KILLED	Par=AlarmID
若报警<AlarmID>没有被激活 Alarm <AlarmID> is not activated	E_OS_STATE	OS_IE_NOMORE_INFO	Par=AlarmID
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若<AlarmID>为无效 ID <AlarmID> is invalid	E_OS_ID	OS_IE_ALARM_ID	Par=AlarmID
若所设置的<increment>的值超出有效范围（恒等于零或是大于 maxallowedvalue） Value of <increment> outside of the admissible limits (equal to zero or greater than maxallowedvalue)	E_OS_VALUE	OS_IE_ALARM_INCREAMENT	Par=AlarmID Par1=start Par2=cycle
若设置的<cycle>的值不为 0 并且超出有效范围（小于 mincycle 或者是大于 maxallowedvalue） Value of <cycle> unequal to 0 and outside of the admissible counter limits (less than mincycle or greater than maxallowedvalue)	E_OS_VALUE	OS_IE_ALARM_CYCLE	Par=AlarmID Par1=start Par2=cycle
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Alarm 没有访问权限 The current application does not have	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=Os_CurrentApplicationId

access to the alarm			Par1=AlarmID

Description
<AlarmID>所标识的报警将被取消。 The system service cancels the alarm <AlarmID>.
Call Opportunity
None
Remark
None

4.1.26 IncrementCounter

Syntax	
StatusType IncrementCounter(CounterType CounterID)	
Parameters	
CounterID(in)	自定义计数器名称 The Counter to be incremented.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
系在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若<CounterID>为无效 ID <CounterID> is invalid	E_OS_ID	OS_IE_COUNTER_ID	Par=CounterID
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par=CounterID
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Counter 没有访问权限	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGH	Par=Os_CurrentApplicatio

The current application does not have access to the counter		T	nId Par1=CounterID

Description
该服务函数会将<CounterID>所标识的自定义计数器加一。 This service increments a software counter.
Call Opportunity
None
Remark
该函数只对用户自定义的 Counter 有效。对系统 Counter 无效。 This function is only valid for user-defined Counter. It has no effect on the system Counter.

4.1.27 GetCounterValue

Syntax	
StatusType GetCounterValue(CounterType CounterID, TickRefType Value)	
Parameters	
CounterID(in)	计数器名称 The Counter which tick value should be read.
Value(out)	用于存储计数器<CounterID>的计数值 Contains the current tick value of the counter.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若<Value>为空指针 Task< Value > is null	E_OS_PARAM_POINTER	OS_IE_NOMORE_INFO	None
若<CounterID>为无效 ID <CounterID> is invalid	E_OS_ID	OS_IE_COUNTER_ID	Par=CounterID
API 不支持跨核调用。 The API does not support cross-core	E_OS_SYS_SERVICE_NOTSUPP	OS_IE_NOMORE_INFO	Par=CounterID

calls.	ORT_CROSSCORE		
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Counter 没有访问权限 The current application does not have access to the counter	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=Os_CurrentApplicationId Par1=CounterID

Description
此服务返回指定<CounterID>当前的计数值。 This service reads the current count value of a counter.
Call Opportunity
None
Remark
None

4.1.28 GetElapsedValue

Syntax	
StatusType GetElapsedValue(CounterType CounterID, TickRefType Value, TickRefType ElapsedValue)	
Parameters	
CounterID(in)	计数器名称 The Counter to be read.
Value(in/out)	输入时，计数器<CounterID>上次的计数值 In: the previously read tick value of the counter. 输出时，用于存储计数器<CounterID>当前的计数值 Out: the current tick value of the counter.
ElapsedValue(out)	用于存储计数器<CounterID>从上次到当前流逝的时间差 The difference to the previous read value.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。	E_OS_CALLEV	OS_IE_NOM	None

Call this API in wrong OS Context.	EL	ORE_INFO	
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若< Value >或< ElapsedValue >为空指针 < Value > or < ElapsedValue > is null	E_OS_PARAM_POINTER	OS_IE_NOMORE_INFO	None
若<CounterID>为无效 ID <CounterID> is invalid	E_OS_ID	OS_IE_COUNTER_ID	Par=CounterID
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par=CounterID
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Counter 没有访问权限 The current application does not have access to the counter	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=Os_CurrentApplicationId Par1=CounterID

Description

此服务返回指定<CounterID>两次计数值之差。
This service gets the number of ticks between the current tick value and a previously read tick value.

Call Opportunity

None

Remark

1. 如果自<Value>至今时间流逝超过一次或多次计数器最大时间，则返回结果<ElapsedValue>是错误的，因为本服务无法检测到这样的溢出。
   If the time since <Value> has elapsed more than once or more than the counter maximum time, the return result <ElapsedValue> is erroneous because this service cannot detect such an overflow.

4.1.29 StartScheduleTableRel

Syntax

StatusType StartScheduleTableRel(ScheduleTableType ScheduleTableID, TickType Offset)

Parameters

ScheduleTableID(in)	调度表名称 Schedule table to be started.
Offset(in)	调度表启动时间距当前时刻的偏移值 Number of ticks on the counter before the schedule table processing is started.

Return Value

StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
当前调度表所属的 Application 已经被终结 The application to which the current schedule table belongs has been killed	E_OS_ACCESS	OS_IE_OBJECT_KILLED	Par=ScheduleTableID
若调度表已启动 Schedule table was already started	E_OS_STATE	OS_IE_SCHTBL_ALREADY_ALIVE	Par=ScheduleTableID
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若< ScheduleTableID >为无效 ID < ScheduleTableID > is invalid	E_OS_ID	OS_IE_SCHTBL_ID	Par=ScheduleTableID
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par=ScheduleTableID
若所设置的<offset>的值超出有效范围（恒等于零或是大于 maxallowedvalue） Value of <offset> outside of the admissible limits (equal to zero or greater than maxallowedvalue)	E_OS_VALUE	OS_IE_SCHTBL_OFFSET	Par=Offset
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Schedule Table 没有访问权限 The current application does not have access to the schedule table	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=ScheduleTableID Par1=Os_CurrentApplicationId

Description

此服务以相对时间启动调度表<ScheduleTableID>。

This service starts the processing of a schedule table at "Offset" relative to the "Now" value on the underlying counter.
Call Opportunity
None
Remark
None

4.1.30 StartScheduleTableAbs

Syntax	
StatusType StartScheduleTableAbs(ScheduleTableType ScheduleTableID, TickType Start)	
Parameters	
ScheduleTableID(in)	调度表名称 Schedule table to be started.
Start(in)	调度表启动的绝对时间 Absolute counter tick value at which the schedule table is started.
Return Value	
StatusType	若成功调用，返回 E_OS_OK; No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
当前调度表所属的 Application 已经被终结 The application to which the current schedule table belongs has been killed	E_OS_ACCESS	OS_IE_OBJECT_KILLED	Par=ScheduleTableID
若调度表已启动 Schedule table was already started	E_OS_STATE	OS_IE_SCHTBL_ALREADY_ALIVE	Par=ScheduleTableID
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若< ScheduleTableID >为无效 ID < ScheduleTableID > is invalid	E_OS_ID	OS_IE_SCHTBL_ID	Par=ScheduleTableID
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par=ScheduleTableID

若所设置的<start>的值超出有效范围（是大于 maxallowedvalue） Value of <start> outside of the admissible limits (greater than maxallowedvalue)	E_OS_VALUE	OS_IE_SCHTBL_START	Par=Start
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Schedule Table 没有访问权限 The current application does not have access to the schedule table	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=Os_CurrentApplicationId Par1=ScheduleTableID

Description
此服务以绝对时间启动调度表<ScheduleTableID>。 This service starts the processing of a schedule table at an absolute value "Start" on the underlying counter.
Call Opportunity
None
Remark
None

4.1.31 StopScheduleTable

Syntax	
StatusType StopScheduleTable(ScheduleTableType ScheduleTableID)	
Parameters	
ScheduleTableID(in)	调度表名称 Schedule table to be stopped.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
当前调度表所属的 Application 已经被终结 The application to which the current schedule table belongs has been killed	E_OS_ACCESS	OS_IE_OBJECT_KILLED	Par=ScheduleTableID
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。	E_OS_CALLEV	OS_IE_NOM	None

Call this API in wrong OS Context.	EL	ORE_INFO	
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若< ScheduleTableID >为无效 ID < ScheduleTableID > is invalid	E_OS_ID	OS_IE_SCHTBL_ID	Par=ScheduleTableID
若调度表已经停止 Schedule table was already stopped	E_OS_NOFUNC	OS_IE_NOMORE_INFO	Par=ScheduleTableID
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par=ScheduleTableID
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Schedule Table 没有访问权限 The current application does not have access to the schedule table	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=Os_CurrentApplicationId Par1=ScheduleTableID

Description
此服务停止指定<ScheduleTableID>的处理。 This service stops the processing of a schedule table immediately at any point while the schedule table is running.
Call Opportunity
None
Remark
当指定的表有 Next 的 Table 时，Next 的 Table 也会被停止。 When the specified table has a Next Table, the Next Table will also be stopped.

4.1.32 NextScheduleTable

Syntax	
StatusType NextScheduleTable(ScheduleTableType ScheduleTable_From, ScheduleTableType ScheduleTable_To)	
Parameters	
ScheduleTableID_From(in)	当前处理的调度表名称 Currently processed schedule table
ScheduleTableID_To(in)	要切换到的调度表名称 Schedule table that provides its series of expiry points
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
当前调度表所属的 Application 已经被终结 The application to which the current schedule table belongs has been killed	E_OS_ACCESS	OS_IE_OBJECT_KILLED	Par=ScheduleTableID
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None
若调度表<ScheduleTable_From>或<ScheduleTable_To>为无效 ID ScheduleTableID_From or ScheduleTableID_To is invalid	E_OS_ID	OS_IE_SCHTBL_ID	Par=ScheduleTableID_From Par1=ScheduleTableID_To
若调度表<ScheduleTable_From>与<ScheduleTable_To>的 refCounter 不相等 [ScheduleTableID_From]. refCounter is not equal to [ScheduleTableID_To]. refCounter	E_OS_ID	OS_IE_SCHTBL_ID	Par=ScheduleTableID_From Par1=ScheduleTableID_To
若调度表<ScheduleTable_To>已开始或为 Next 状态 ScheduleTableID_To is started or next	E_OS_STATE	OS_IE_NOMORE_INFO	Par=ScheduleTableID_To
若调度表<ScheduleTable_From>还未开启 ScheduleTableID_From not started	E_OS_NOFUNC	OS_IE_NOMORE_INFO	Par=ScheduleTableID_From
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par=ScheduleTableID_From Par1=ScheduleTableID_To
当前的 OS-Application 无效。 Current OS-Application is invalid.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Schedule Table 没有访问权限 The current application does not have access to the schedule table	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=Os_CurrentApplicationId Par1=Schedu

			leTableID_From Par2=ScheduleTableID_To

Description
此服务将一个调度表切换为另一个调度表。 This service switches the processing from one schedule table to another schedule table.
Call Opportunity
None
Remark
1.必须等当前的调度表完整的运行完毕后，才会切换至下一个调度表。 You must wait for the current schedule to run completely before switching to the next schedule. 2.在 Next 运行之前，再次调用该函数更新 Next 表，以最后调用的一次为准。 Before Next table runs, call this function again to update the Next table, the last call will be valid.

4.1.33 GetScheduleTableStatus

Syntax	
StatusType GetScheduleTableStatus(ScheduleTableType ScheduleTableID, ScheduleTableStatusRefType ScheduleTableStatus)	
Parameters	
ScheduleTableID(in)	请求状态的调度表名称 Schedule table for which status is requested
ScheduleStatus(out)	用于存储调度表<ScheduleTableID>的状态 Reference to ScheduleTableStatusType
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK; 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
当前调度表所属的 Application 已经被终结 The application to which the current schedule table belongs has been killed	E_OS_ACCESS	OS_IE_OBJECT_KILLED	Par=ScheduleTableID
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	None

若< ScheduleTableID >为无效 ID < ScheduleTableID > is invalid	E_OS_ID	OS_IE_SCHTBL_ID	Par=ScheduleTableID
若<ScheduleStatus>为空 Schedule table< ScheduleStatus > is null	E_OS_PARAM_POINTER	OS_IE_NOMORE_INFO	None
API 从不同的内核调用 API is called from different core	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par=ScheduleTableID
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId
当前的 Application 对该 Schedule Table 没有访问权限 The current application does not have access to the schedule table	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=Os_CurrentApplicationId Par1=ScheduleTableID

Description
此服务返回指定<ScheduleTableID>当前的状态。 This service switches the processing from one schedule table to another schedule table.
Call Opportunity
None
Remark
None

4.1.34 GetISRID

Syntax	
ISRType GetISRID(void)	
Parameters	
void	none
Return Value	
StatusType	若成功调用，返回正在执行的 ISR 的 ID； No error, ID of the ISR currently executing; 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	OS_INVALID_ISR	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	OS_INVALID_ISR	OS_IE_NOMORE_INFO	None

API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	OS_INVALID_ISR	OS_IE_NOMORE_INFO	None

Description
此服务返回当前正在执行的ISR的ID。 This service returns the identifier of the currently executing ISR.
Call Opportunity
None
Remark
该函数只能返回当前运行的二类中断的 ID，不支持一类中断。 This function can only return the ID of the C2 interrupt currently running, and does not support the C1 interrupt.

4.1.35 GetCoreID

Syntax	
CoreIdType GetCoreID(void)	
Parameters	
void	none
Return Value	
CoreIdType	The return value is the unique ID of the core.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前< CoreID >为无效 ID Current < CoreID > is invalid	OS_INVALID_COREID	None	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	OS_INVALID_ISR	OS_IE_NOMORE_INFO	None

Description
获取当前的CoreId。 Get the current CoreId.
Call Opportunity
None
Remark
None

4.1.36 GetTaskStackUsage

Syntax
StatusType GetTaskStackUsage(TaskType TaskId, CONSTP2VAR(Os_uint16, AUTOMATIC, OS_APPL_DATA) Value)
Parameters

TaskId (in)	Identifier of a task
Value (out)	return the stack usage in byte
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
若<Value>为空指针 Task< Value > is null	E_OS_PARAM_POINTER	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	OS_INVALID_ISR	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
若< TaskId >为无效 ID Task < TaskId > is invalid	E_OS_ID	OS_IE_TASK_ID	Par=TaskId
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par=TaskId

Description
Get the maximum usage of the task stack since the OS is started.
Call Opportunity
None
Remark
None

4.1.37 GetISRStackUsage

Syntax
StatusType GetISRStackUsage(ISRType ISRId, CONSTP2VAR(Os_uint16, AUTOMATIC, OS_APPL_DATA) Value)
Parameters

ISRId (in)	Identifier of an ISR
Value (out)	return the stack usage in byte
Return Value	

StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
若<Value>为空指针 Task< Value > is null	E_OS_PARAM_POINTER	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	OS_INVALID_ISR	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
若< ISRId >为无效 ID Task < ISRId > is invalid	E_OS_ID	OS_IE_ISR_ID	Par=ISRId
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par=ISRId

Description

Get the maximum usage of the ISR stack since the OS is started.

Call Opportunity

None

Remark

None

4.1.38 EnableInterruptSource

Syntax

StatusType EnableInterruptSource (ISRType ISRID, Os_boolean ClearPending)

Parameters

ISRID (in)	Identifier of an ISR
ClearPending (in)	Whether to clear the interrupt flag

Return Value

StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	Par=ISRID, Os_IntAPIUsedFlag[currentCoreID]
中断所属的 CoreID 非当前 CoreID The CoreID to which the interrupt belongs is not the current CoreID	E_OS_ID	OS_IE_ISR_ID	Par=ISRID, Os_ISRCfg[ISRID].objCoreId
中断为二类中断且当前的 Application 非该中断所属的 Application	E_OS_ACCESS	OS_IE_ISR_ID	Par=ISRID, Os_ISRCfg[ISRID].ApplID
Extended			
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	Par=ISRID, Os_CurrentContext[currentCoreID]
若< ISRID >为无效 ID ISR < ISRID > is invalid	E_OS_ID	OS_IE_ISR_ID	Par=ISRID
若中断已经使能 ISR is already enabled	E_OS_NOFUNC	OS_IE_NOMORE_INFO	Par=ISRID, OS_TRUE

Description
Enables the interrupt source by modifying the interrupt controller registers. Additionally it may clear the interrupt pending flag.
Call Opportunity
None
Remark
None

4.1.39 DisableInterruptSource

Syntax	
StatusType DisableInterruptSource (ISRType ISRID)	
Parameters	
ISRID (in)	Identifier of an ISR
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK. 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	Par=ISRID, Os_IntAPIUsedFlag[currentCoreID]
中断所属的 CoreID 非当前 CoreID The CoreID to which the interrupt belongs is not the current CoreID	E_OS_ID	OS_IE_ISR_ID	Par=ISRID, Os_ISRCfg[ISRID].objCoreId
中断为二类中断且当前的 Application 非该中断所属的 Application	E_OS_ACCESS	OS_IE_ISR_ID	Par=ISRID, Os_ISRCfg[ISRID].ApplID
Extended			
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	Par=ISRID, Os_CurrentContext[currentCoreID]
若< ISRID >为无效 ID ISR < ISRID > is invalid	E_OS_ID	OS_IE_ISR_ID	Par=ISRID
若中断已经使能 ISR is already enabled	E_OS_NOFUNC	OS_IE_NOMORE_INFO	Par=ISRID, OS_TRUE

Description
Disables the interrupt source by modifying the interrupt controller registers.
Call Opportunity
None
Remark
None

4.1.40 ClearPendingInterrupt

Syntax	
StatusType ClearPendingInterrupt (ISRType ISRID)	
Parameters	
ISRID (in)	Identifier of an ISR
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK. 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	Par=ISRID, Os_IntAPIUsedFlag[currentCoreID]
中断所属的 CoreID 非当前 CoreID The CoreID to which the interrupt belongs is not the current CoreID	E_OS_ID	OS_IE_ISR_ID	Par=ISRID, Os_ISRCfg[ISRID].objCoreId
中断为二类中断且当前的 Application 非该中断所属的 Application	E_OS_ACCESS	OS_IE_ISR_ID	Par=ISRID, Os_ISRCfg[ISRID].ApplID
Extended			
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	Par=ISRID, Os_CurrentContext[currentCoreID]
若< ISRID >为无效 ID ISR < ISRID > is invalid	E_OS_ID	OS_IE_ISR_ID	Par=ISRID

Description
Clears the interrupt pending flag by modifying the interrupt controller registers.
Call Opportunity
None
Remark
None

4.2 SC3

4.2.1 GetCurrentApplicationID

Syntax	
ApplicationType GetCurrentApplicationID(void)	
Parameters	
void	none
Return Value	
ApplicationType	<identifier of running OS-Application> or INVALID_OSAPPLICATION
Error Report	

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	OS_INVALID_ISR	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR

Description
This service determines the OS-Application where the caller of the service is currently executing.
Call Opportunity
None
Remark
只有在使用可信函数时，GetCurrentApplicationID 的返回值和 GetApplicationID 是不同的。Only when a trusted function is used, the return value of GetCurrentApplicationID and GetApplicationID are different. 可信函数运行时，GetCurrentApplicationID 返回的是可信函数所属的 OS-Application。When the trusted function is running, GetCurrentApplicationID returns the OS-Application to which the trusted function belongs.

4.2.2 GetApplicationID

Syntax	
ApplicationType GetApplicationID(void)	
Parameters	
void	none
Return Value	
ApplicationType	<identifier of running OS-Application> or INVALID_OSAPPLICATION

Situation	Return Value	Report ErrInfo	Report ErrPar
Error Report			
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR

API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	OS_INVALID_ISR	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR

Description
This service determines the OS-Application (a unique identifier has to be allocated to each application) where the caller originally belongs to (was configured to).
Call Opportunity
None
Remark
None

4.2.3 AllowAccess

Syntax	
StatusType AllowAccess(void)	
Parameters	
void	none
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK. 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	OS_INVALID_ISR	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
Current application is OS_INVALID_OSAPPLICATION	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId[CoreID]

Description
This service sets the own state of an OS-Application from APPLICATION_RESTARTING to APPLICATION_ACCESSIBLE.
Call Opportunity
None
Remark

None

4.2.4 GetApplicationState

Syntax	
StatusType GetApplicationState(ApplicationType Application, ApplicationStateRefType Value)	
Parameters	
Application (in)	The OS-Application from which the state is requested
Value (out)	The current state of the application
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	OS_INVALID_ISR	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
< Application > is invalid	E_OS_ID	OS_IE_OSAPP_ID	Par= Application
< Value > is null	E_OS_PARAM_POINTER	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 不支持跨核调用。 The API does not support cross-core calls.	E_OS_SYS_SERVICE_NOTSUPPORT_CROSSCORE	OS_IE_NOMORE_INFO	Par = Application

Description
This service returns the current state of an OS-Application.
Call Opportunity
None
Remark
None

4.2.5 TerminateApplication

Syntax	
StatusType TerminateApplication(ApplicationType Application, RestartType RestartOption)	
Parameters	
Application (in)	The identifier of the OS-Application to be terminated. If the caller belongs to <Application> the call results in a self-termination.
RestartOption (in)	Either RESTART for doing a restart of the OS-Application or NO_RESTART if OS-Application shall not be restarted.
Return Value	
StatusType	若成功调用，返回 E_OS_OK; No error, E_OS_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	OS_INVALID_ISR	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
< Application > is invalid	E_OS_ID	OS_IE_OSAPP_ID	Par = Application
Current application invalid	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par = Os_CurrentApplicationId[CoreID]
Application state is APPLICATION_TERMINATED	E_OS_STATE	OS_IE_APP_ALREADY_TERMINATE	OS_IE_NO_ERRPAR

Description
This service terminates or restart the OS-Application. If OS-Application is terminated, all OS objects belong to this OS-Application will be terminated.
Call Opportunity
None
Remark

None

4.2.6 CheckObjectAccess

Syntax	
ObjectAccessType CheckObjectAccess (ApplicationType ApplID, ObjectTypeType ObjectType, Os_ObjectIdType ObjectId)	
Parameters	
ApplID (in)	OS-Application identifier
ObjectType (in)	Type of the following parameter
ObjectId (in)	The object to be examined
Return Value	
ObjectAccessType	ACCESS if the ApplID has access to the object NO_ACCESS otherwise

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	NO_ACCESS	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	NO_ACCESS	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	NO_ACCESS	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
< Application > is invalid	NO_ACCESS	OS_IE_OSAPP_ID	Par = Application
Object invalid	NO_ACCESS	OS_IE_OBJECT_ID	Par = ObjectType

Description
This service determines if the OS-Applications, given by ApplID, is allowed to use the IDs of a Task, Resource, Counter, Alarm or Schedule Table in API calls.
Call Opportunity
None
Remark
None

4.2.7 CheckObjectOwnership

Syntax	
ApplicationType CheckObjectOwnership (ObjectTypeType ObjectType, Os_ObjectIdType ObjectId)	
Parameters	
ObjectType (in)	Type of the following parameter
ObjectId (in)	The object to be examined
Return Value	
ApplicationType	The OS-Application to which the object ObjectType belongs or INVALID_OSAPPLICATION if the object does not exists

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	INVALID_OSAPPLICATION	None	None
在错误的 OS Context 调用该 API。Call this API in wrong OS Context.	INVALID_OSAPPLICATION	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	INVALID_OSAPPLICATION	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
ObjectType is OBJECT_RESOURCE	INVALID_OSAPPLICATION	OS_IE_OBJECT_ID	Par = ObjectType
Object invalid	INVALID_OSAPPLICATION	OS_IE_OBJECT_ID	Par = ObjectType

Description
This service determines to which OS-Application a given Task, ISR, Counter, Alarm or Schedule Table belongs.
Call Opportunity
None
Remark
None

4.2.8 CallTrustedFunction

Syntax
StatusType CallTrustedFunction (

TrustedFunctionIndexType FunctionIndex, TrustedFunctionParameterRefType FunctionParams)

Parameters	
FunctionIndex (in)	Index of the function to be called.
FunctionParams (in)	Pointer to the parameters for the function - specified by the FunctionIndex - to be called. If no parameters are provided, a NULL pointer has to be passed.

Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
FunctionIndex invalid	E_OS_ID	OS_IE_TRUSTFUN_ID	Par = FunctionIndex

Description
A (trusted or non-trusted) OS-Application uses this service to call a trusted function.
Call Opportunity
None
Remark
None

4.2.9 IocWrite

Syntax
Std_ReturnType IocWrite_IocIdX_SenderY (DataType DataName)
Parameters

DataName (in)	User-defined data name
Return Value	
Std_ReturnType	若成功调用，返回 IOC_E_OK； No error, IOC_E_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	IOC_E_NOK	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
当前 OS-Application ID 与 IOC 配置的发送方 OS-Application ID 不同 Os_CurrentApplicationId[CoreID] is not the Configured ID	IOC_E_NOK	OS_IE_NO_ACCESS_RIGHT	(Os_uint32)Os_CurrentApplicationId[CoreID]
IOC ID 大于等于 IOC 个数 IocId >=OS_IOC_NUMBER	IOC_E_NOK	OS_IE_IOC_ID	(Os_uint32)IocId

Description

该函数用于发送数据，适用于单个数据，且不使用队列的情况。
This function is used to send data and is suitable for a single data without using a queue.

Call Opportunity

None

Remark

None

4.2.10 IocRead

Syntax

```
Std_ReturnType IocRead_IocIdX_ReceiverY
(
        DataType* DataName
)
```

Parameters

DataName (in)	User-defined data name
Return Value	
Std_ReturnType	若成功调用，返回 IOC_E_OK；

	No error, IOC_E_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	IOC_E_NOK	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
当前 OS-Application ID 与 IOC 配置的发送方 OS-Application ID 不同 Os_CurrentApplicationId[CoreID] is not the Configured ID	IOC_E_NOK	OS_IE_NO_ACCESS_RIGHT	(Os_uint32)Os_CurrentApplicationId[CoreID]
IOC ID 大于等于 IOC 个数 IocId >=OS_IOC_NUMBER	IOC_E_NOK	OS_IE_IOC_ID	(Os_uint32)IocId

Description

该函数用于读取数据，适用于单个数据，且不使用队列的情况。
This function is used to read data and is suitable for a single data without using a queue.

Call Opportunity

None

Remark

None

4.2.11 IocWriteGroup

Syntax

```
Std_ReturnType IocWriteGroup_IocIdX_SenderY
(
    DataType0 DataName0,
    DataType1 DataName1,
    …
)
```

Parameters

DataName0,1…(in)	User-defined data name

Return Value

Std_ReturnType	若成功调用，返回 IOC_E_OK； No error, IOC_E_OK.

	其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	IOC_E_NOK	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
当前 OS-Application ID 与 IOC 配置的发送方 OS-Application ID 不同 Os_CurrentApplicationId[CoreID] is not the Configured ID	IOC_E_NOK	OS_IE_NO_ACCESS_RIGHT	(Os_uint32)Os_CurrentApplicationId[CoreID]
IOC ID 大于等于 IOC 个数 IocId >=OS_IOC_NUMBER	IOC_E_NOK	OS_IE_IOC_ID	(Os_uint32)IocId

Description

该函数用于发送数据，适用于多个数据，且不使用队列的情况。
This function is used to send data and is suitable for multiple data and no queue is used.

Call Opportunity

None

Remark

None

4.2.12 IocReadGroup

Syntax

```
Std_ReturnType IocReadGroup_IocIdX_ReceiverY
(
    DataType0* DataName0,
    DataType1* DataName1,
    …
)
```

Parameters

DataName0,1…(in)	User-defined data name

Return Value

Std_ReturnType	若成功调用，返回 IOC_E_OK； No error, IOC_E_OK. 其他情况参考 error report。

	For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	IOC_E_NOK	None	None
在错误的 OS Context 调用该 API。Call this API in wrong OS Context.	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
当前 OS-Application ID 与 IOC 配置的发送方 OS-Application ID 不同 Os_CurrentApplicationId[CoreID] is not the Configured ID	IOC_E_NOK	OS_IE_NO_ACCESS_RIGHT	(Os_uint32)Os_CurrentApplicationId[CoreID]
IOC ID 大于等于 IOC 个数 IocId >=OS_IOC_NUMBER	IOC_E_NOK	OS_IE_IOC_ID	(Os_uint32)IocId

Description

该函数用于读取数据，适用于多个数据，且不使用队列的情况。
This function is used to read data. It is suitable for multiple data and does not use queues.

Call Opportunity

None

Remark

None

4.2.13 IocSend

Syntax

```
Std_ReturnType IocSend_IocIdX_SenderY
(
        DataType DataName
)
```

Parameters

DataName (in)	User-defined data name

Return Value

Std_ReturnType	若成功调用，返回 IOC_E_OK； No error, IOC_E_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar

Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	IOC_E_NOK	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
当前 OS-Application ID 与 IOC 配置的发送方 OS-Application ID 不同 Os_CurrentApplicationId[CoreID] is not the Configured ID	IOC_E_NOK	OS_IE_NO_ACCESS_RIGHT	(Os_uint32)Os_CurrentApplicationId[CoreID]
IOC ID 大于等于 IOC 个数 IocId >=OS_IOC_NUMBER	IOC_E_NOK	OS_IE_IOC_ID	(Os_uint32)IocId

Description
该函数用于发送数据，适用于单个数据，且使用队列的情况。 This function is used to send data and is suitable for single data and queues are used.
Call Opportunity
None
Remark
None

4.2.14 IocReceive

Syntax
Std_ReturnType IocReceive_IocIdX_Receiver (DataType* DataName)

Parameters	
DataName (in)	User-defined data name
Return Value	
Std_ReturnType	若成功调用，返回 IOC_E_OK； No error, IOC_E_OK. 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			

Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	IOC_E_NOK	None	None
在错误的 OS Context 调用该 API。Call this API in wrong OS Context.	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
当前 OS-Application ID 与 IOC 配置的发送方 OS-Application ID 不同 Os_CurrentApplicationId[CoreID] is not the Configured ID	IOC_E_NOK	OS_IE_NO_ACCESS_RIGHT	(Os_uint32)Os_CurrentApplicationId[CoreID]
IOC ID 大于等于 IOC 个数 IocId >=OS_IOC_NUMBER	IOC_E_NOK	OS_IE_IOC_ID	(Os_uint32)IocId

Description

该函数用于读取数据，适用于单个数据，且使用队列的情况。
This function is used to read data. It is suitable for single data and queues are used.

Call Opportunity

None

Remark

None

4.2.15 IocSendGroup

Syntax

```
Std_ReturnType IocSendGroup_IocIdX_SenderY
(
    DataType0 DataName0,
    DataType1 DataName1,
    …
)
```

Parameters

DataName0,1…(in)	User-defined data name

Return Value

Std_ReturnType	若成功调用，返回 IOC_E_OK； No error, IOC_E_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or	IOC_E_NOK	None	None

Current CoreState is not OS_CORE_START			
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
当前 OS-Application ID 与 IOC 配置的发送方 OS-Application ID 不同 Os_CurrentApplicationId[CoreID] is not the Configured ID	IOC_E_NOK	OS_IE_NO_ACCESS_RIGHT	(Os_uint32)Os_CurrentApplicationId[CoreID]
IOC ID 大于等于 IOC 个数 IocId >=OS_IOC_NUMBER	IOC_E_NOK	OS_IE_IOC_ID	(Os_uint32)IocId

Description

该函数用于发送数据，适用于多个数据，且使用队列的情况。
This function is used to send data and is suitable for multiple data and queues are used.

Call Opportunity

None

Remark

None

4.2.16 IocReceiveGroup

Syntax

```
Std_ReturnType IocReceiveGroup_IocIdX_Receiver
(
    DataType0* DataName0,
    DataType1* DataName1,
    …
)
```

Parameters

DataName0,1…(in)	User-defined data name

Return Value

Std_ReturnType	若成功调用，返回 IOC_E_OK； No error, IOC_E_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not	IOC_E_NOK	None	None

OS_CORE_START			
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	IOC_E_NOK	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
当前 OS-Application ID 与 IOC 配置的发送方 OS-Application ID 不同 Os_CurrentApplicationId[CoreID] is not the Configured ID	IOC_E_NOK	OS_IE_NO_ACCESS_RIGHT	(Os_uint32)Os_CurrentApplicationId[CoreID]
IOC ID 大于等于 IOC 个数 IocId >=OS_IOC_NUMBER	IOC_E_NOK	OS_IE_IOC_ID	(Os_uint32)IocId

Description

该函数用于读取数据，适用于多个数据，且使用队列的情况。
This function is used to read data. It is suitable for multiple data and queues are used.

Call Opportunity

None

Remark

None

4.2.17 IocEmptyQueue

Syntax

```
Std_ReturnType IocEmptyQueue_x
(
    void
)
```

Parameters

None	

Return Value

Std_ReturnType	若成功调用，返回 IOC_E_OK； No error, IOC_E_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
当前 OS-Application ID 与 IOC 配置的发送方和接收方 OS-Application ID 都不同 Os_CurrentApplicationId[CoreID] is not the Configured ID	IOC_E_NOK	OS_IE_NO_ACCESS_RIGHT	(Os_uint32)Os_CurrentApplicationId[CoreID]

Description
Used to clear IOC queue.
Call Opportunity
None
Remark
None

4.3 MultiCore

4.3.1 ControlIdle

Syntax
StatusType ControlIdle (CoreIdType CoreID, IdleModeType IdleMode)

Parameters	
CoreID (in)	Selects the core which idle mode is set.
IdleMode (in)	The mode which shall be performed during idle time.

Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	OS_INVALID_COREID	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	E_OS_DISABLEDINT	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
CoreID >= OS_PL_MAX_COER_NUMBER	E_OS_ID	OS_IE_COREID	(Os_uint32)CoreID

Description
This API allows the caller to select the idle mode action which is performed during idle time of

the OS (e.g. if no Task/ISR is active). It can be used to implement energy savings. The real idle modes are hardware dependent and not standardized. The default idle mode on each core is IDLE_NO_HALT.
Call Opportunity
None
Remark
None

4.3.2 ShutdownAllCores

Syntax	
void ShutdownAllCores (StatusType Error)	
Parameters	
Error (in)	<Error> needs to be a valid error code supported by the AUTOSAR OS.
Return Value	
None	

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid	None	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	None	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
API 在中断处理 APIs 之间调用 API is called between interrupt processing APIs	None	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
Os_CurrentApplicationId invalid	None	OS_IE_INVALID_OSAPPICATION	Par = Os_CurrentApplicationId[CurrentCoreId]
CurrentApplication is untrusted	None	OS_IE_SHUTDOWN_BY_UNTRUST_APP	Par = Os_CurrentApplicationId[CurrentCoreId]

Description
After this service the OS on all AUTOSAR cores is shut down. Allowed at TASK level and ISR level and also internally by the OS. The function will never return. The function will force other cores into a shutdown.

Call Opportunity
None
Remark
None

4.3.3 StartCore

Syntax	
void StartCore (CoreIdType CoreID, P2VAR(StatusType, AUTOMATIC, OS_APPL_DATA) Status)	
Parameters	
CoreID (in)	Core identifier.
Status (out)	Return value of the function in extended status: E_OS_OK: No Error E_OS_ID: Core ID is invalid. E_OS_ACCESS: The function was called after starting the OS. E_OS_STATE: The Core is already activated. Return value of the function in standard status: E_OS_OK: No Error
Return Value	
None	
Description	
It is not supported to call this function after StartOS(). The function starts the core specified by the parameter CoreID. The OUT parameter allows the caller to check whether the operation was successful or not. If a core is started by means of this function StartOS shall be called on the core.	
Call Opportunity	
None	
Remark	
None	

4.3.4 StartNonAutosarCore

Syntax
void StartNonAutosarCore (CoreIdType CoreID, P2VAR(StatusType, AUTOMATIC, OS_APPL_DATA) Status)
Parameters

CoreID (in)	Physical Core identifier.
Status (out)	Return value of the function in extended status: E_OS_OK: No Error E_OS_ID: Core ID is invalid. E_OS_STATE: The Core is already activated. Return value of the function in standard status: E_OS_OK: No Error
Return Value	
None	
Description	
The function starts the core specified by the parameter CoreID. It is allowed to call this function after StartOS(). The OUT parameter allows the caller to check whether the operation was successful or not. It is not allowed to call StartOS on cores activated by StartNonAutosarCore. Otherwise the behavior is unspecified.	
Call Opportunity	
None	
Remark	
None	

4.3.5 GetNumberOfActivatedCores

Syntax			
Os_uint32 GetNumberOfActivatedCores(void)			
Parameters			
None			
Return Value			
Os_uint32	Number of cores activated by the StartCore function		
Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is not OS_CORE_START	OS_INVALID_COREID	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	OS_INVALID_COREID	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
Description			
The function returns the number of cores activated by the StartCore function.			
Call Opportunity			
None			
Remark			
None			

4.3.6 GetSpinlock

Syntax	
StatusType GetSpinlock (SpinlockIdType SpinlockID)	
Parameters	
SpinlockID (in)	操作的自旋锁 ID The SpinLock ID of the operation.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK. 其他情况参考 error report。 For more information, see error report.

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
SpinlockID >= OS_SPINLOCK_NUMBER	E_OS_ID	OS_IE_SPINLOCK_ID	(Os_uint32)SpinlockID
Current application is OS_INVALID_OSAPPLICATION	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId[CoreID]
当前 OS-Application 没有权限操作该 Spinlock。 The current OS-Application does not have permission to operate on the spinlock.	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=Os_CurrentApplicationId[CoreID]
OsSpinlock_GetEnvCheck() return OS_FALSE	E_OS_SYS_SPINLOCKMETHOD	OS_IE_NOMORE_INFO	(Os_uint32)Os_IntAPIUsedFlag[CoreID]
The Spinlock is already occupied on the same core	E_OS_INTERFERENCE_DEADLOCK	OS_IE_NOMORE_INFO	(Os_uint32)SpinlockID
If the sequence by which multiple spinlocks are occupied at the same	E_OS_NESTING_DEADLOCK	OS_IE_NOMORE_INFO	(Os_uint32)SpinlockID

time on one core do not comply with the configured order.			
Os_SpinlockDepth[CoreID] > OS_SPINLOCK_INVALD_NEST_LEVEL	None	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR

Description
GetSpinlock tries to occupy a spin-lock variable. If the function returns, either the lock is successfully taken or an error has occurred. The spinlock mechanism is an active polling mechanism. The function does not cause a de-scheduling.
Call Opportunity
None
Remark
None

4.3.7 TryToGetSpinlock

Syntax
StatusType TryToGetSpinlock (SpinlockIdType SpinlockID, P2VAR(TryToGetSpinlockType, AUTOMATIC, OS_APPL_DATA) Success)

Parameters	
SpinlockID (in)	操作的自旋锁 ID The SpinLock ID of the operation.
Success (out)	Returns if the lock has been occupied or not.
Return Value	
StatusType	若成功调用，返回 E_OS_OK； No error, E_OS_OK. 其他情况参考 error report。 For more information, see error report.

Error Report			
Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。 Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
SpinlockID >= OS_SPINLOCK_NUMBER	E_OS_ID	OS_IE_SPINLOCK_ID	(Os_uint32)SpinlockID
Success 为空指针。	E_OS_PARAM_	OS_IE_NOM	OS_IE_NO_

Success is a null pointer.	POINTER	ORE_INFO	ERRPAR
Current application is OS_INVALID_OSAPPLICATION	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId[CoreID]
当前 OS-Application 没有权限操作该 Spinlock。 The current OS-Application does not have permission to operate on the spinlock.	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=Os_CurrentApplicationId[CoreID]
OsSpinlock_GetEnvCheck() return OS_FALSE	E_OS_SYS_SPINLOCKMETHOD	OS_IE_NOMORE_INFO	(Os_uint32)Os_IntAPIUsedFlag[CoreID]
The Spinlock is already occupied on the same core	E_OS_INTERFERENCE_DEADLOCK	OS_IE_NOMORE_INFO	(Os_uint32)SpinlockID
If the sequence by which multiple spinlocks are occupied at the same time on one core do not comply with the configured order.	E_OS_NESTING_DEADLOCK	OS_IE_NOMORE_INFO	(Os_uint32)SpinlockID
Os_SpinlockDepth[CoreID] > OS_SPINLOCK_INVALD_NEST_LEVEL	None	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR

Description

TryToGetSpinlock has the same functionality as GetSpinlock with the difference that if the spinlock is already occupied by a TASK on a different core the function sets the OUT parameter "Success" and returns with E_OK.

Call Opportunity

None

Remark

None

4.3.8 ReleaseSpinlock

Syntax

```
StatusType ReleaseSpinlock
(
    SpinlockIdType SpinlockID
)
```

Parameters

SpinlockID (in)	操作的自旋锁 ID The SpinLock ID of the operation.

Return Value

Error Report

Situation	Return Value	Report ErrInfo	Report ErrPar
Standard			
None			
Extended			
Current < CoreID > is invalid or Current CoreState is not OS_CORE_START	E_OS_CORE	None	None
在错误的 OS Context 调用该 API。Call this API in wrong OS Context.	E_OS_CALLEVEL	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR
SpinlockID >= OS_SPINLOCK_NUMBER	E_OS_ID	OS_IE_SPINLOCK_ID	(Os_uint32)SpinlockID
Current application is OS_INVALID_OSAPPLICATION	E_OS_ACCESS	OS_IE_INVALID_OSAPPICATION	Par=Os_CurrentApplicationId[CoreID]
当前 OS-Application 没有权限操作该 Spinlock。The current OS-Application does not have permission to operate on the spinlock.	E_OS_ACCESS	OS_IE_NO_ACCESS_RIGHT	Par=Os_CurrentApplicationId[CoreID]
SpinLock is not released LIFO	E_OS_ACCESS	OS_IE_SPINLOCK_LIFO_ERROR	(Os_uint32)SpinlockID

Description
ReleaseSpinlock releases a spinlock variable that was occupied before. Before terminating a TASK all spinlock variables that have been occupied with GetSpinlock() shall be released.
Call Opportunity
None
Remark
None

4.4 Callback

None.

4.5 Callout

4.5.1 StartupHook

Syntax	
void StartupHook (void)	
Parameters	
void	none
Return Value	
void	none

Description
系统启动过程中StartOS，会调用该Hook函数。 StartOS will call the Hook function during system startup.
Call Opportunity
None
Remark
See 3.1.1 for more details.

4.5.2 ShutdownHook

Syntax	
void ShutdownHook (StatusType Error)	
Parameters	
Error	发生错误的错误标识 error occurred
Return Value	
void	none
Description	
系统关闭过程中ShutsownOS，会调用该Hook函数。 ShutsownOS will call the Hook function during system shutdown.	
Call Opportunity	
None	
Remark	
See 3.1.2 for more details.	

4.5.3 PreTaskHook

Syntax	
void PreTaskHook (void)	
Parameters	
void	none
Return Value	
void	none
Description	
EAS.OS中，每一次有任务要进入到运行状态，PreTaskHook函数都会被调用。 In EAS.OS, PreTaskHook function is called every time a task enter the running state.	
Call Opportunity	
None	
Remark	
1.PreTaskHook/ PostTaskHook通常用于测量任务的执行时间。 PreTaskHook/ PostTaskHook is usually used to measure the execution time of a task. 2.用户也可以在PreTaskHook中通过调用系统服务函数“GetTaskID()”获取将要开始运行的任务ID信息。 The user can also obtain the ID information of the task to be started by calling the system service function "GetTaskID()" in the PreTaskHook.	

4.5.4 PostTaskHook

Syntax	
void PostTaskHook (void)	
Parameters	
void	none
Return Value	
void	none
Description	
EAS.OS中，每一次有任务要离开运行状态，PostTaskHook函数都会被调用。 In EAS.OS, each time a task has to leave the running state, the PostTaskHook function is called.	
Call Opportunity	
None	
Remark	
用户可以在 PostTaskHook 中通过调用系统服务函数“GetTaskID()”获取即将退出运行的任务 ID 信息。 The user can obtain the ID information of the task that about to exit the running state by calling the system service function "GetTaskID()" in the PostTaskHook.	

4.5.5 PreIsrHook

Syntax	
void PreIsrHook (void)	
Parameters	
void	none
Return Value	
void	none
Description	
EAS.OS中，每一次有中断要进入到运行状态，PreIsrHook函数都会被调用。 In EAS.OS, PreIsrHook function is called every time a ISR has to enter the running state.	
Call Opportunity	
None	
Remark	
1. PreIsrHook/ PostIsrHook 通常用于测量 ISR 的执行时间； PreIsrHook/ PostIsrHook is usually used to measure the execution time of a ISR; 2. 用户也可以在 PreIsrHook 中通过调用系统服务函数“GetISRID()”获取将要开始运行的 ISRID 信息。 The user can also obtain the ID information of the ISR to be started by calling the system service function "GetISRID()" in the PreIsrHook.	

4.5.6 PostIsrHook

Syntax	
void PostIsrHook (void)	
Parameters	
void	none
Return Value	
void	none

Description
EAS.OS中，每一次有中断要离开运行状态，PostIsrHook函数都会被调用。 In EAS.OS, each time a ISR has to leave the running state, the PostIsrHookfunction is called.
Call Opportunity
BCC1, BCC2, ECC1, ECC2
Remark
1. 用户可以在 PostIsrHook 中通过调用系统服务函数“GetISRID()”获取即将退出运行的中断 ID 信息。 The user can obtain the ID information of the ISR that about to exit the running state by calling the system service function " GetISRID()" in the PostIsrHook.

4.5.7 ErrorHook

Syntax	
void ErrorHook (StatusType Error)	
Parameters	
StatusType	发生错误的错误标识 error occurred
Return Value	
void	none
Description	
EAS.OS 中，每当系统服务函数在运行过程中发生错误时，ErrorHook 函数都将被调用。 In EAS.OS, the ErrorHook function will be called whenever an error occurs in the system service function.	
Call Opportunity	
None	
Remark	
None	

4.5.8 AppErrorHook

Syntax	
void AppErrorHook(void)	
Parameters	
void	none
Return Value	
void	none
Description	
EAS.OS 中，每当 OS-Application 在运行过程中发生错误时，AppErrorHook 函数都将被调用。具体函数名由用户通过配置工具生成。 In EAS.OS, the AppErrorHook function will be called whenever an error occurs in the OS-Application. The specific function name is generated by the user through the configuration tool.	
Call Opportunity	
None	
Remark	

None

4.5.9 AppShutdownHook

Syntax	
void AppShutdownHook(void)	
Parameters	
void	none
Return Value	
void	none
Description	
EAS.OS 中，每当 OS-Application 被终止时，AppShutdownHook 函数都将被调用。具体函数名由用户通过配置工具生成。 In EAS.OS, the AppShutdownHook function will be called whenever the OS-Application is terminated. The specific function name is generated by the user through the configuration tool.	
Call Opportunity	
None	
Remark	
None	

4.5.10 AppStartupHook

Syntax	
void AppStartupHook(void)	
Parameters	
void	none
Return Value	
void	none
Description	
EAS.OS 中，每当 OS-Application 启动时，AppStartupHook 函数都将被调用。具体函数名由用户通过配置工具生成。 In EAS.OS, the AppStartupHook function will be called whenever Startup the OS-Application. The specific function name is generated by the user through the configuration tool.	
Call Opportunity	
None	
Remark	
None	

4.5.11 ProtectionHook

Syntax	
ProtectionReturnType ProtectionHook(StatusType FatalError)	
Parameters	
FatalError	E_OS_PROTECTION_MEMORY: A memory access violation

	occurred E_OS_PROTECTION_TIME: A Task/Category 2 ISR exceeds its execution time budget E_OS_PROTECTION_ARRIVAL: A Task/Category 2 ISR arrives before its timeframe has expired E_OS_PROTECTION_LOCKED: A Task/Category 2 ISR blocks for too long
Return Value	
ProtectionReturnType	The return value defines the action the OS shall take after the protection hook. PRO_IGNORE: Ignore this error. PRO_TERMINATETASKISR: terminate current running task or ISR PRO_TERMINATEAPPL: terminate current OS-Application PRO_TERMINATEAPPL_RESTART: restart current OS-Application PRO_SHUTDOWN: shutdown OS
Description	
时间保护或内存保护发生错误时，调用该 Hook 函数。 When an error occurs in the time protection or memory protection, the Hook function is called.	
Call Opportunity	
None	
Remark	
1.只有错误是 E_OS_PROTECTION_ARRIVAL 时，才允许返回 PRO_IGNORE。其他情况下，用户返回 PRO_IGNORE，OS 会执行 Shutdown。 2.如果用户没有配置 OS-Application，或者当前无有效的 OS-Application，该函数返回 PRO_TERMINATEAPPL 或 PRO_TERMINATEAPPL_RESTART 时，OS 会执行 Shutdown。 3.如果返回了 PRO_TERMINATETASKISR，但当前没有任务或中断在运行，OS 会执行 Shutdown。 1. Only when the error is E_OS_PROTECTION_ARRIVAL, is it allowed to return to PRO_IGNORE. In other cases, if the user returns to PRO_IGNORE, the OS will execute Shutdown. 2. If the user has not configured OS-Application, or there is currently no valid OS-Application, when the function returns PRO_TERMINATEAPPL or PRO_TERMINATEAPPL_RESTART, the OS will execute Shutdown. 3. If PRO_TERMINATETASKISR is returned, but no task or interrupt is currently running, the OS will execute Shutdown.	

4.5.12 TASK

Syntax	
TASK(TaskName)	
Parameters	
void	none

Return Value	
void	none
Description	
用户自定义的任务。用户在任务函数中实现相关的功能。 User-defined tasks. The user implements related functions in the task function.	
Call Opportunity	
None	
Remark	
任务名称由用户定义。The task name is defined by the user.	

4.5.13 ISR

Syntax	
ISR(IsrName)	
Parameters	
void	none
Return Value	
void	none
Description	
用户自定义的中断。用户在ISR函数中实现相关的功能。 User-defined ISR. The user implements related functions in the ISR function.	
Call Opportunity	
None	
Remark	
ISR 名称由用户定义。The ISR name is defined by the user.	

4.5.14 AlarmCallback

Syntax	
ALARMCALLBACK(CallBackName)	
Parameters	
void	none
Return Value	
void	none
Description	
用户自定义的Alarm Callback。 User-defined Alarm Callback.	
Call Opportunity	
None	
Remark	
Callback 名称由用户定义。The callback name is defined by the user. 仅在 SC1 时支持该 Callback。This Callback is only supported in SC1.	

4.5.15 IocInit Callout

Syntax

void IocInit_IocIdx_DataInitz(UserDataType* UserData)	
Parameters	
UserData	User define data
Return Value	
void	none
Description	
Init User define data.	
Call Opportunity	
None	
Remark	
None	

4.6 Scheduled API

None.

5 Configuration

本章节描述了通过工具配置 Operating System SC1 的方法，关于 EAS.Configurator 配置工具总体的使用说明参见之前的相关文档和说明。

This section describes the method of configuring OS SC1 module through EAS.Configurator tools. The general usage of the EAS.Configurator configuration tool see previous related documents and instructions.

5.1 General configuration interface

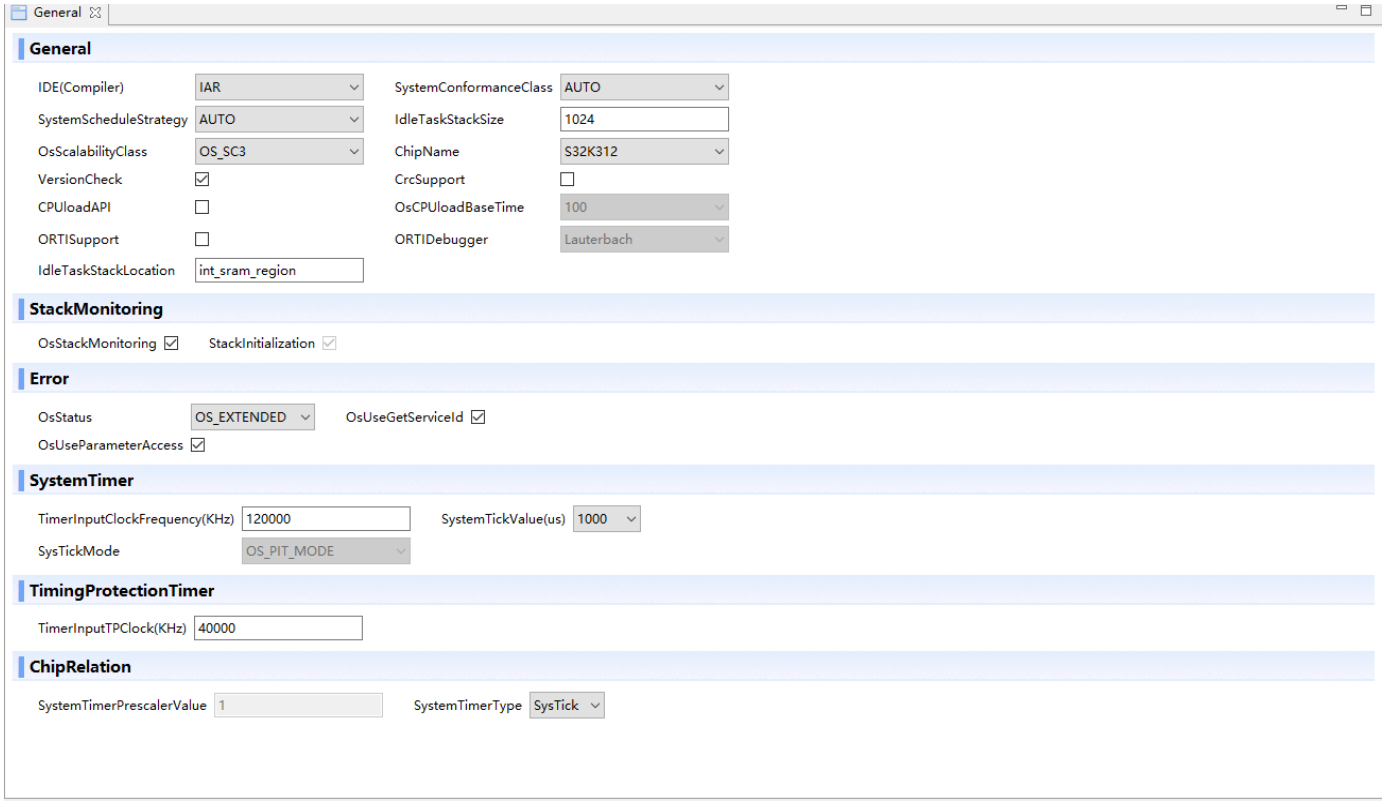


Figure5-1 General configuration interface

在对操作系统进行配置时，用户首先应当在上图所示的树形菜单中鼠标左键单击 “Os -> General”选项进入到 General 配置界面对操作系

统的全局属性进行配置。上图显示了操作系统配置工具的 General 配置界面。界面左侧树形结构菜单，右侧分为两部分区域—“配置窗口”和“备注/错误提示窗口”。用户配置操作系统时在配置窗口中对各个配置项进行操作，备注窗口内会显示当前鼠标所在配置项的功能描述和参数取值范围。在配置代码生成过程中，工具会对用户的配置信息进行错误检测。如果用户的配置内容有误，工具会在备注/错误提示窗口内显示相关错误描述。

In configuration, the user should click "Os - > General" option with the left key of mouse firstly and switch into the general configuration interface. Figure above shows the general configuration interface in Configurator. The right side is divided into two parts: "configuration window" and "note/error window". When the user configures the operating system, each item is operated in the configuration window, the function description and parameter value range of the item the current mouse located at are displayed in the note window. During configuration code generation, Configurator will detect the correctness of user's configuration information. If the information is incorrect, related error description will display in the note/error window by Configurator.

5.1.1 General configuration items

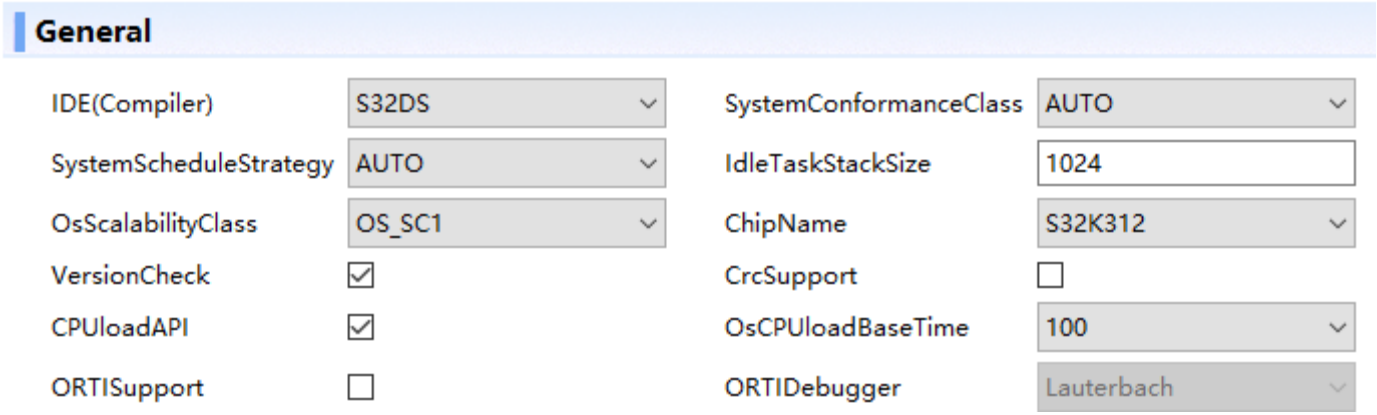


Figure5-2 General configuration parameters

Figure above shows the configuration parameter for the general options, each configuration item is shown in table below.

Table5-1 General configuration parameter information

Name	Type	Range	Default Value	Description
IDE(Compiler)	Enum			选择使用的编译器环境。根据芯片而定。 Select the compiler environment to use. The choice depends on the chip.
SystemConformanceClass	Enum	OS_BCC1/ OS_BCC2/ OS_ECC1/ OS_ECC2/ AUTO	AUTO	选择 OS 符合类。可选取： Select the OS conformance class. Can be selected: BCC1：系统仅支持 BasicTask，不支持任务的多次激活，每个优先级仅能有一个任务； BCC1: only basic tasks, limited to one activation request per task and one task per priority, while all tasks have different priorities; BCC2：系统仅支持 BasicTask，支持任务的多次激活，每个优先级可以有多个任务； BCC2: like BCC1, plus more than one task per priority possible and multiple requesting of task activation allowed; ECC1：类似于 BCC1，增加了 ExtendedTask。 ECC1: like BCC1, plus extended tasks; ECC2：类似于 ECC1，支持 BasicTask 的多次激活，每个优先级可以有多个任务； ECC2: like ECC1, plus more than one task per priority possible and multiple requesting of task activation allowed for basic tasks AUTO：根据系统的配置情况，自动选定系统符合类。

				AUTO: according to the configuration of the system, Configurator will automatic selects conformance class. 如若用户对 OS 符合类不熟悉，建议使用默认 AUTO 选项。 If the user is not familiar with conformance class, the default AUTO option is recommended.
SystemScheduleStrategy	Enum	OS_NON_PREEMPT/ OS_MIX_PREEMPT/ OS_FULL_PREEMPT/ AUTO	AUTO	选择系统调度模式。可选取： Select the OS scheduling policy. Can be selected: 系统不支持任务抢占，即所有任务都要为“不可抢占”的情况； OS_NON_PREEMPT：The system does not support task preemption, that is, all tasks should be "non preemptive "; 系统支持完全任务抢占，即所有任务都要为“可抢占”的情况； OS_FULL_PREEMPT：The system supports complete task preemption, that is, all tasks must be "full preemptive "; 系统支持某些任务为“非抢占”、某些为“可抢占”情况。 OS_MIX_PREEMPT: The system supports certain tasks as "non-preemptive" and certain "preemptible" situations; AUTO：根据系统中任务的配置情况，自动选定系统的调度模式。 AUTO：according to the configuration of the system,

				Configurator will automatic selects scheduling policy.
IdleTaskStackSize	Int	256--4096	1024	Select the stack capacity used by the StatisticsTask.
OsScalabilityClass	Enum	OS_SC1/ OS_SC2/ OS_SC3/ OS_SC4/	OS_SC1	选择 OS 裁剪类。可选取： Select the OS Scalability Class. Can be selected. OS_SC1：OSEK OS + Counter Interface+ SWFRT Interface+Schedule Tables+ Stack Monitoring; OS_SC2：OS_SC1+ ProtectionHook+TimingProtection; OS_SC3：OS_SC1 + Memory Protection+ OS-Applications+ServiceProtection+ CallTrustedFunction; OS_SC4：OS_SC1 + OS_SC2+OS_SC3. 具体支持类型随芯片而定。 The support type depends on the chip.
ChipName	Enum			选择芯片型号。可选择。 Select the Chip Name. Can be selected.
CrcSupport	Boolean	False/True	False	选择是否开启 CRC 检测功能。可选择。 Select whether to open the CRC detection function. Can be selected.
CPUloadAPI	Boolean	False/True	False	选择是否开启 CPU 负载检测功能。可选择。 Select whether to open the CPU load detection function. Can be selected.
OsCPUloadBaseTime	Long	10/100/500/1000/	100	选择 CPU 负载检测的基本时间

				Select the basic time for CPU load detection.

5.1.2 SystemTimer configuration items

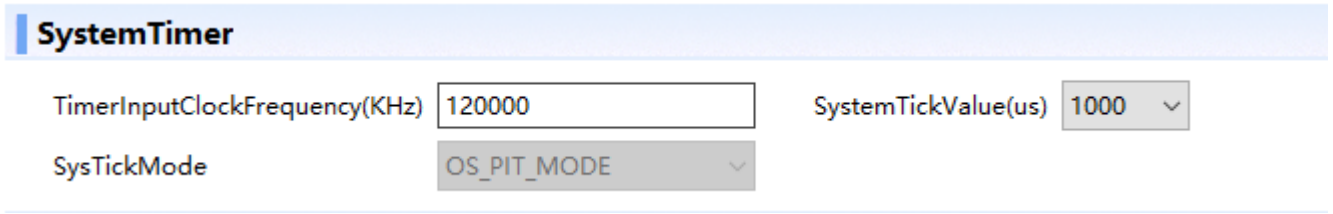


Figure5-3 SystemTimer configuration items

Figure above shows the configuration parameters for the system timer, each configuration item is shown in table below.

Table5-2 SystemTimer configuration parameter information

Name	Type	Range	Default Value	Description
TimerInputClockFrequency(KHz)	Int	参考芯片手册 Refer to chip manual	-	填写定时器模块的输入频率。 Select the input frequency of the timer module.

5.1.3 TimingProtectionTimer configuration items

Figure5-4 TimingProtectionTimer configuration items

Name	Type	Range	Default Value	Description
TimerInputTPClock(KHz)	Int	参考芯片手册 Refer to chip manual	-	Time Protection Timer 所使用的时钟频率。 Clock used by Time Protection Timer.

Table5-3 SystemTimer configuration parameter information

5.1.4 Stack monitoring configuration items

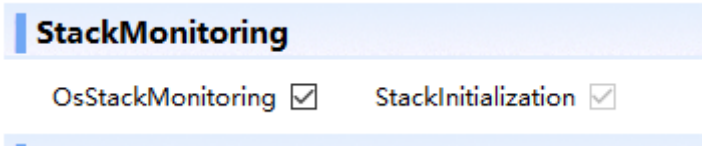


Figure5-5 Stack monitoring configuration parameters

Figure above shows the configuration parameters for the operating system stack detection, each configuration item is shown in table below.

Table5-4 StackMonitoring configuration parameter information

Name	Type	Range	Default Value	Description
OsStackMonitoring	Boolean	False/True	True	选择是否开启堆栈检测功能。可选择： Select whether to open the stack detection function. Can be selected: Disable：关闭堆栈检测； Disable: Disable stack detection; Enable：开启堆栈检测。 Enable: Enable stack detection. 开启堆栈检测功能能够检测堆栈溢出错误。开启堆栈检测功能后，堆栈初始化配置项（StackInitialization）将被强制使能。开发调试阶段建议用户选择 Enable，产品发布阶段建议用户选择 Disable。 The stack detection function can detect stack overflow errors.

				After it is enabled, the StackInitialization configuration item(StackInitialization) will be coerced. During the development and debugging phase, the user is recommended to select Enable, and it is suggested that the user choose Disable to release product.
StackInitialization	Boolean	False/True	True	选择是否在系统启动阶段进行堆栈初始化操作。可取值： Select whether to open the StackInitialization .Can be selected: Disable：关闭堆栈初始化； Disable: Disable stack initialization; Enable：开启堆栈初始化。 Enable: Enable stack initialization. 当启用堆栈检测(StackMonitoring)功能的时候，该配置项将被强制使能。开发调试阶段建议用户选择 Enable，产品发布阶段建议用户选择 Disable。 When the stack detection function is enabled, this configuration item will be forcibly enabled. It is recommended that the user select the “Enable”during the development and debugging phase, and select the “Disable” during the product release phase.

5.1.5 Error configuration items

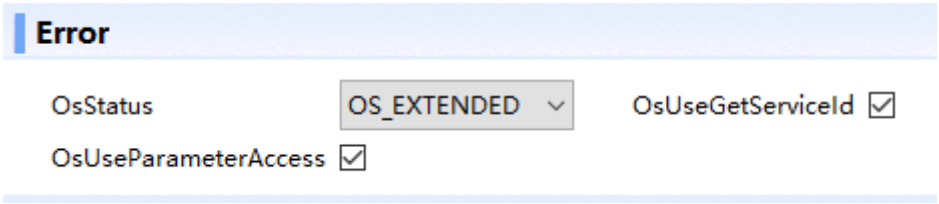


Figure5-6 Error configuration parameters

Figure above shows the configuration options for four operating system error handling parameters in the General configuration interface, as shown in table below for each configuration item.

Table5-5 Error configuration parameter information

Name	Type	Range	Default Value	Description
OsStatus	Enum	OS_STANDARD/ OS_EXTENDED	OS_EXTENDED	选择操作系统的错误检查等级。可取值： Select the error check level for the operating system. Can be selected: OS_STANDARD：操作系统只能对少数几类错误进行检测； OS_STANDARD: the operating system can only detect a few types of errors; OS_EXTENDED：操作系统能够进行更为全面的错误检测。 OS_EXTENDED: the operating system can perform more comprehensive error detection. 建议用户在开发调试阶段选择 OS_EXTENDED 模式，在产品发布阶段选择 OS_STANDARD 模式。

				It is recommended that the user select the “OS_EXTENDED” during the development and debugging phase, and select the “OS_STANDARD” during the product release phase.
OsUseGetServiceID	Boolean	False/True	True	选择当系统中有错误发生时，系统是否为用户提供发生错误的 API 编号。可取值： When an error occurs in the system, the system provides the user with the wrong API number. Can be selected: Disable：不启用宏函数 OSErrorGetServiceId(CoreId)； Disable: Disable function OSErrorGetServiceId(CoreId); Enable：启用宏函数 OSErrorGetServiceId(CoreId)。 Enable: Enable function OSErrorGetServiceId(CoreId). 用户可以在 ErrorHook 中通过上述函数获取发生错误的系统 API 的 ID。系统 API 的 ID 格式为：OSServiceId_xxx，其中 xxx 为系统 API 的函数名。 The user can obtain the ID of the system API where the error occurred through the above function in ErrorHook. The ID format of the system API is: OSServiceId_xxx, where xxx is the function name of the system API. 建议用户在开发调试阶段选择 Enable 选项，在产品发布阶段选择 Disable 选项。 It is recommended that the user select the “Enable”during the development and debugging phase, and select the “Disable” during the product release phase.

OsUseParameterAccess	Boolean	False/True	True	选择当系统中有错误发生时，系统是否为用户提供出错 API 所使用的参数。可取值： Select the parameters used by the system to provide the user with the error API when errors occur in the system. Can be selected: Disable：不启用该项功能； Disable: Disable this feature; Enable：启用该项功能。 Enable: Enable this feature. 建议用户在开发调试阶段选择 Enable 选项，在产品发布阶段选择 Disable 选项。 It is recommended that the user select the “Enable” during the development and debugging phase, and select the “Disable” during the product release phase.

5.1.6 ChipRelation

Figure5-7 ChipRelation configuration items

Figure above shows the configuration options for the ChipRelation parameter, and the configuration information is shown in table below.

Table5-6 ChipRelation configuration parameter information

Name	Type	Range	Default Value	Description
SystemTimerPresaler Value	Int	1--256	1	当 STM 被选择时，输入系统定时器的预分频值。 Input the prescaler value of the system timer when STM is selected. 可能值：当使用 STM 时：1-256 Possible value: When STM is used: 1-256.
SystemTimerType	Enum	--	--	选择系统计数器使用的硬件时钟源类型。 Choose the hardware timer source type used for the system counter.

5.1.7 Hook configuration items

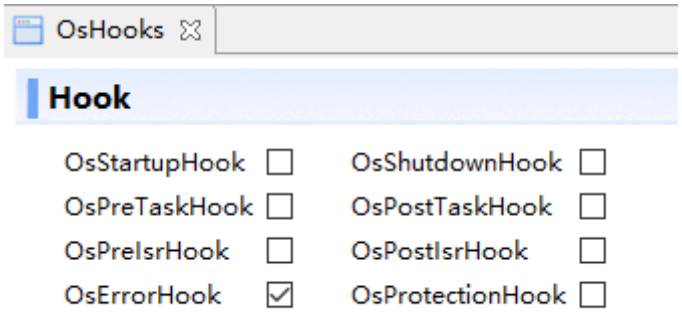


Figure5-8 Hook configuration parameters

Figure above shows the configuration options for the Hook parameter, and the configuration information is shown in table below.

Table5-7 Hook configuration parameter information

Name	Type	Range	Default Value	Description
OsStartupHook	Boolean	False/True	False	选择系统是否使用 StartupHook 功能。可勾选： Select whether the system use the StartupHook function. Can be selected.
OsShutdownHook	Boolean	False/True	False	选择系统是否使用 ShutdownHook 功能。可勾选： Select whether the system use the ShutdownHook function. Can be selected.
OsPreTaskHook	Boolean	False/True	False	选择系统是否使用 PreTaskHook 功能。可勾选： Select whether the system use the PreTaskHook function. Can be selected.
OsPostTaskHook	Boolean	False/True	False	选择系统是否使用 PostTaskHook 功能。可勾选： Select whether the system use the PostTaskHookfunction.Can be selected.
OsPreIsrHook	Boolean	False/True	False	选择系统是否使用 PreIsrHook 功能。可勾选： Select whether the system use the PreIsrHook function. Can be selected.
OsPostIsrHook	Boolean	False/True	False	选择系统是否使用 PostIsrHook 功能。可勾选： Select whether the system use the PostIsrHookfunction. Can be selected.
OsErrorHook	Boolean	False/True	True	选择系统是否使用 ErrorHook 功能。可勾选： Select whether the system use the ErrorHook function. Can be selected.
OsProtectionHook	Boolean	False/True	False	选择系统是否使用 ProtectionHook 功能。可勾选： Select whether the system use the ProtectionHook function. Can be selected.

5.2 Task configuration interface

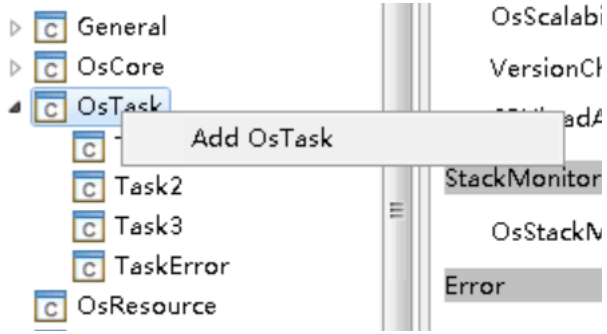


Figure5-9 Add tasks

在对操作系统各项全局参数完成配置以后，用户可以通过鼠标右键单击树形菜单“Os -> Task”，然后鼠标左键单击弹出的“Add Task”按钮在操作系统中建立任务，如图 7 所示。建立任务时，需要在弹出的命名窗口中输入任务名称，如图 8 所示。之后，新建立的任务“Task1”将作为树形结构的第三级菜单出现在第二级菜单“Task”下面，用户可以通过鼠标右键单击“Os -> Task -> 任务名称”，然后在弹出的选项中通过鼠标左键单击“Del Task1”、“Modify name of Task1”或者“Add OsTaskAutostart”对任务进行删除、重命名或者设置为自启动。

After general configuration, users can choose the menu "Os - > Task", and then click the right mouse button, then the "Add Task" button will be popped up, click the bottom shown in figure 7 will add a task to system. When setting up a task, you need to enter a task name in a pop-up named window, as shown in figure 8. After that, the newly created Task "Task" will appear as the third level menu of the tree structure under the second menu "Task". User can click the right mouse button Os - > Task - > Task name ", and then in the pop-up window, user can click "Del Task", "Modify the name of Task" or "Add OsTaskAutostart" to delete task, rename task or set task to auto start.

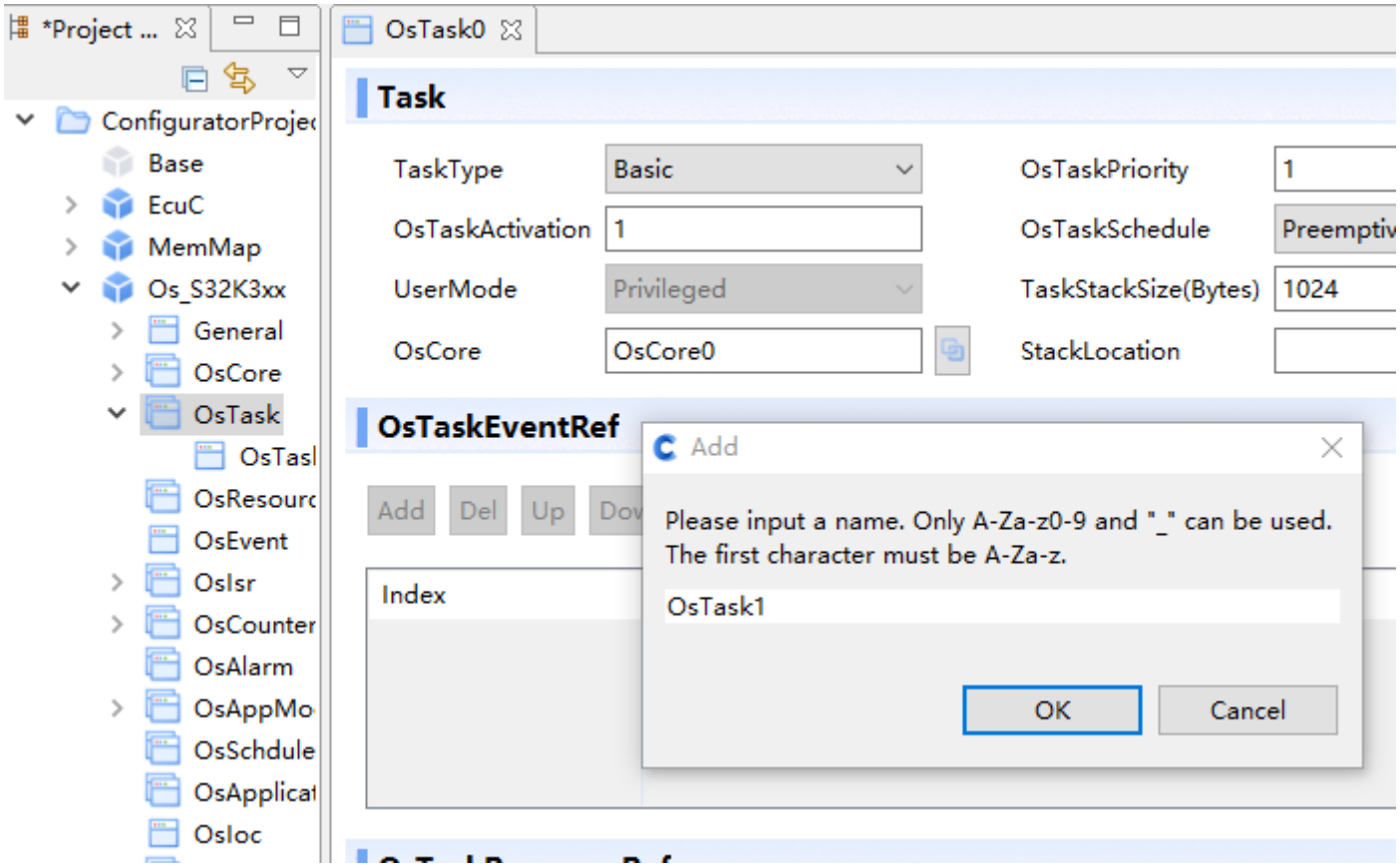


Figure5-10 Task naming window

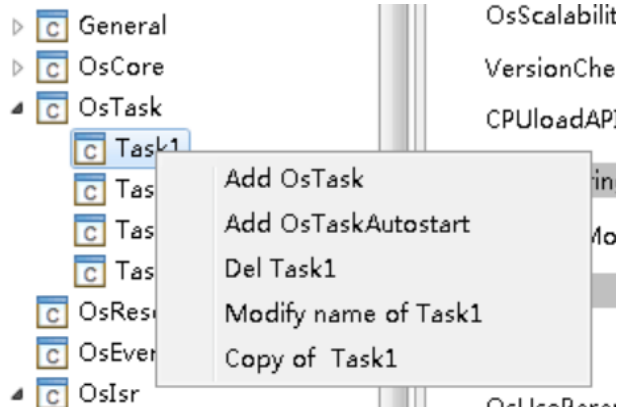


Figure5-11 Delete/rename task window

注 1：用户需对 Task 的命名注意，不要与 OS 内部函数重名，比如 OsTask_Init，工具不会进行检查，只能在编译的过程中发现。

Note 1: Users should pay attention to the naming of task, and do not duplicate the name of OS internal functions, such as OsTask_Init, the tool will not check this and can only find it during compilation.

对于操作系统中其它对象的删除、重命名和设置自启动操作，与上述任务的操作过程相同，在下文中不再重复描述。

Since the operation above of other objects of operating system is the same as operation of task, so the repeat process above of other objects is ignored below.

下图显示了操作系统配置工具的 Task 配置界面。Task 配置界面中包含四类配置项，以下对这四类配置选项进行详细介绍。

It is shown the task configuration interface for the operating system with Configurator in Figure 11. The Task configuration interface contains four categories of configuration items, and the following are detailed descriptions of these four configuration options.

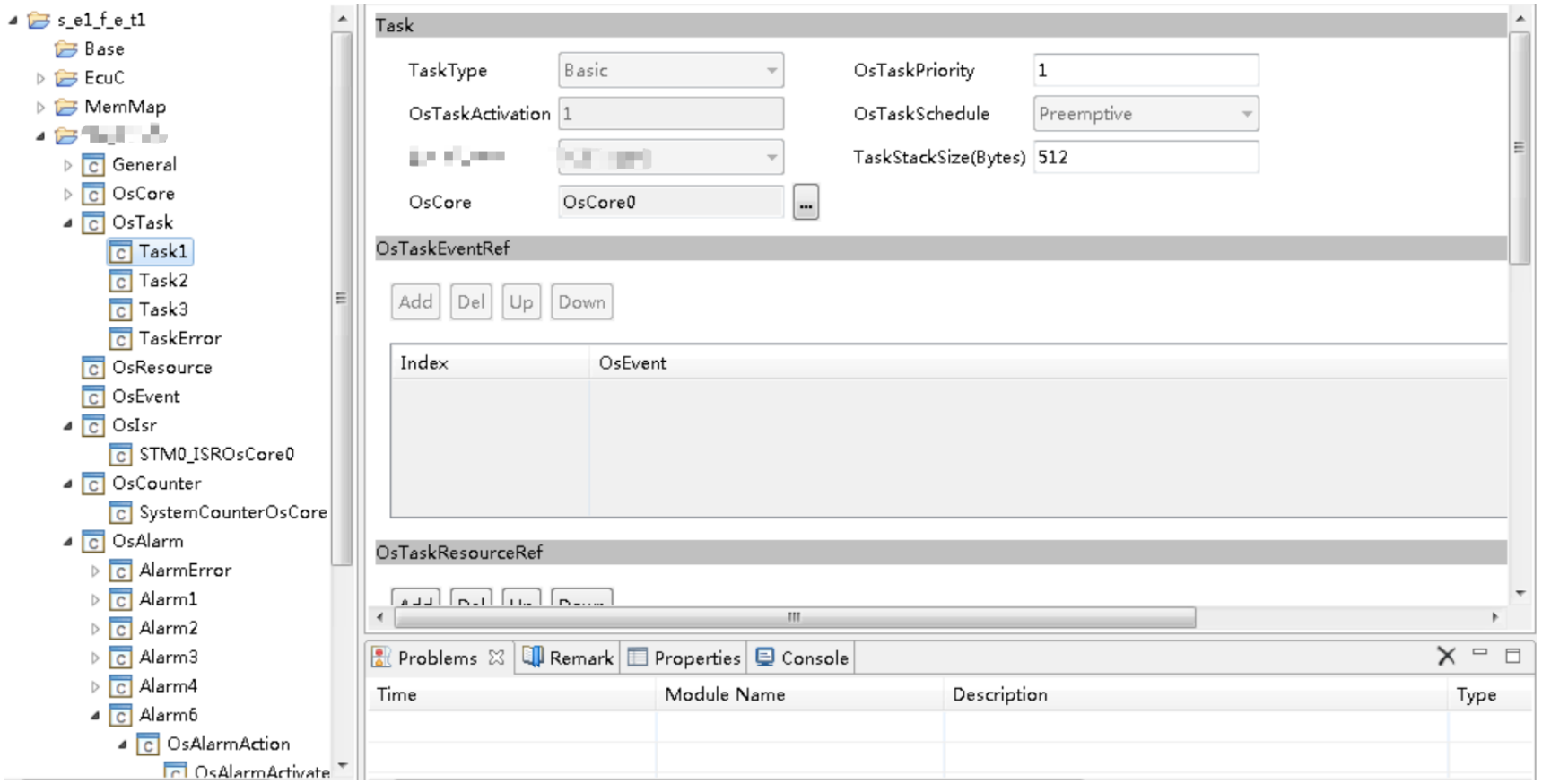


Figure5-12 Task configuration interface

5.2.1 Basic property configuration

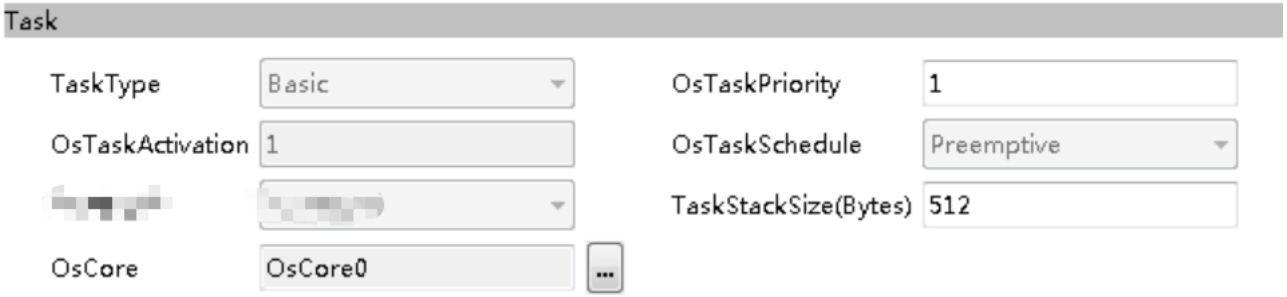


Figure5-13 Basic configuration about property and parameters of task

Figure above shows the configurated options about basic property of task, each options is shown in table below.

Table5-8 basic attribute parameter information of task

Name	Type	Range	Default Value	Description
TaskType	Enum	Basic/ Extended	Basic	选择任务类型。可取值： Select the task type. Can be selected: Basic：可多次激活，不能使用事件同步功能； Basic: can be activated multiple times and cannot use event synchronization; Extended：不可多次激活，可以使用事件同步功能。 Extended: cannot be activated multiple times, can use event synchronization function. 用户应根据应用程序的需要对任务类型进行选择。 The user should select the task type according to the needs of the application.
OsTaskPriority	Int	0--1021	1	填写任务优先级。

				Fill in task priority. 可取值：0--1021 Value: 0--1021. 优先级数字越大，代表的优先级越高。 The larger the priority number, the higher the priority.
OsTaskActivation	Int	1--255	1	填写任务的最大激活次数。 Fill in the maximum number of activation times. 由于只有 BasicTask 可以被多次激活，所以只有当任务类型(TaskType)被配置为 Basic 时，用户才可以进行对该配置项进行配置。 Since only basic task can be activated multiple times, the user can configure the item only if the “TaskType” is configured as “BasicTask”.
OsTaskSchedule	Enum	Preemptive/ Non- Preemptive	Preemptive	选择任务的抢占属性。可取值： Select the preemptive attribute of the task. Can be selected: Preemptive：任务为可抢占类型； Preemptive: the task is the Preemptive type; Non-preemptive：任务为不可抢占类型。 Non-preemptive: the task is non-preemptive type.
TaskStackSize	Int	256--32768	512	填写任务的堆栈大小。 Fill in the stack size of the task.
OsCore	Ref			

Right click on a Task to set it as Auto Start. Figure above shows the configuration options when Task is configured to auto start, and selects the

AppMode when the Task start.

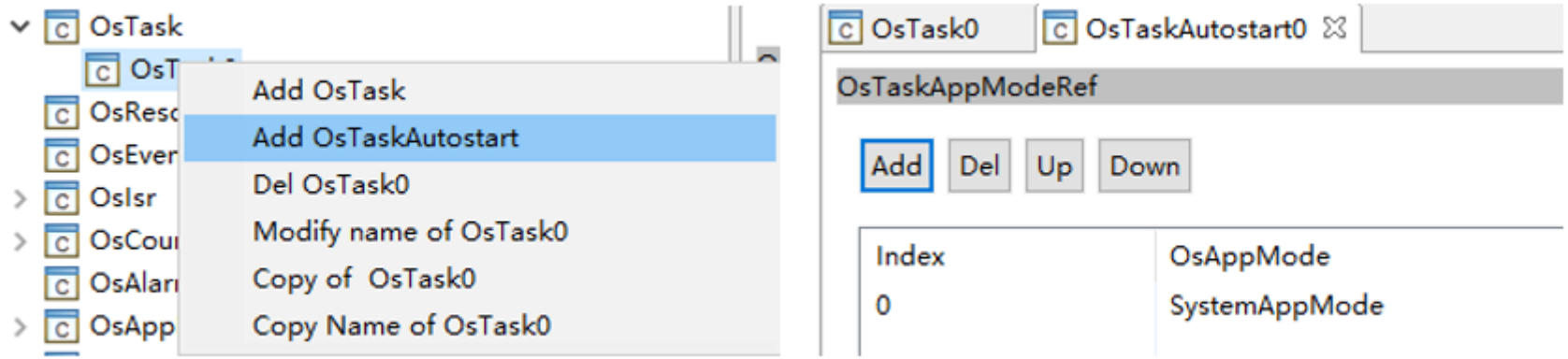


Figure5-14 Basic property configuration of auto start task

5.2.2 Event allocation configuration for task

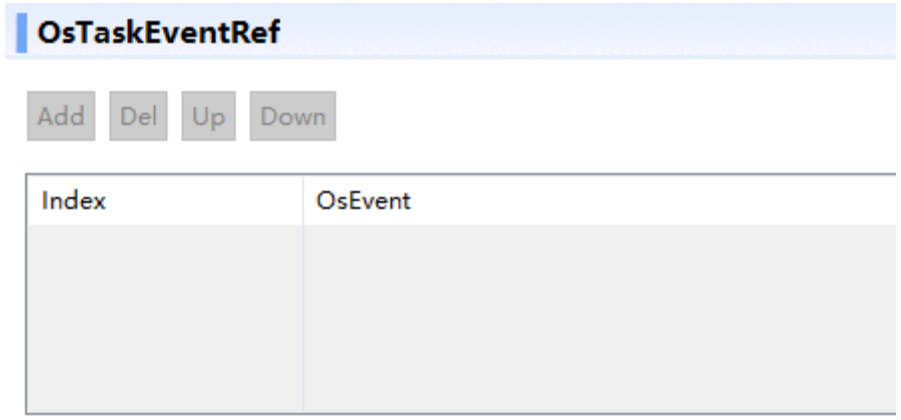


Figure5-15 the event allocation option for Task

上图显示了 Task 配置界面中的事件分配选项。如果用户在 Task 基本属性配置项中将任务类型(TaskType)配置为 Extended，上图所示的事

件分配选项将被开启。用户可以鼠标左键单击“Add”按钮，将事件列表中的事件分配给当前任务。如果用户需要将已经分配给当前任务的事件删除，则需要在图中下方的事件列表中通过鼠标左键单击的形式选中需要被删除的事件，然后鼠标左键单击“Del”按钮将其删除。每个任务需要分配多少事件由用户根据应用程序的需求而定。

Figure above shows the event allocation options in the Task configuration interface. If the user configured the “Task type” of a task as “Extended”, the event allocation option shown in figure above will be opened. The user can click the "Add" button to assign events in the event list to the current task. If the user needs to delete event which has been assigned to the task, click on the "Del" button is needed. How many events each task needs to be allocated with depends on the user's requirements for the application.

5.2.3 Resource allocation configuration for task

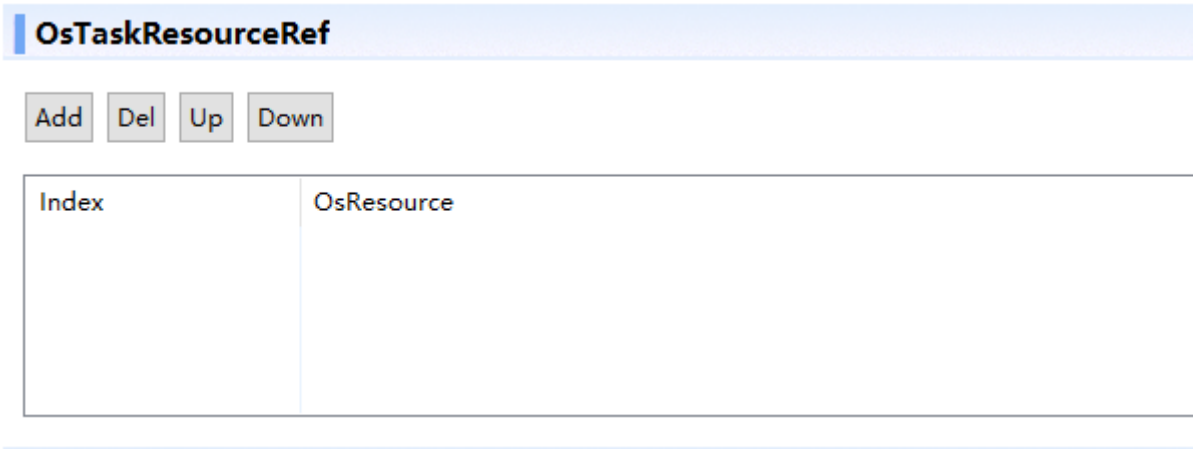


Figure5-16 the resource allocation option for Task

上图显示了 Task 配置界面中的资源分配选项。用户可以鼠标左键单击“Add”按钮，将资源分配给当前任务。如果用户需要将已经分配给当前任务的资源删除，则需要在图下方的资源列表中通过鼠标左键单击的形式选中需要被删除的资源，然后鼠标左键单击“Del”按钮将其删除。每个任务需要分配多少资源由用户根据应用程序的需求而定。同时用户需要注意，每个任务至多可以分配一个内部资源。

Figure above shows the resource allocation options in the Task configuration interface. The user can click the "Add" button to assign resource in the resource list to the current task. If the user needs to delete resource which has been assigned to the task, click on the "Del" button is needed. How many

resources each task needs to be allocated with depends on the user's requirements for the application. Also, it is aware that each task can allocate at most one internal resource.

5.2.4 TaskAccessingApplication configuration

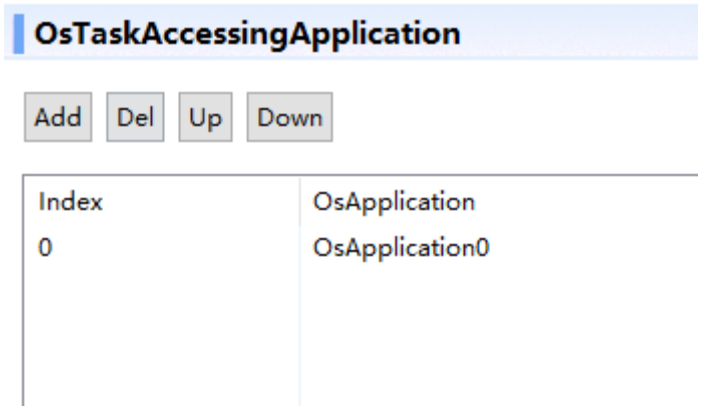


Figure5-17 TaskAccessingApplication configuration

上图显示了 Task 配置界面中的 Application 访问选项。用户可以鼠标单击“Add”按钮，允许当前任务访问所选 Application。如果用户需要将 Task 的 application 访问权限删除，则需要在图下方的列表中通过鼠标左键单击的形式选中此配置，然后鼠标左键单击“Del”按钮将其删除。每个 Task 需要访问多少 Applications 由用户跟据 Task 的需求而定。

The figure above shows the Application access option in the Task configuration interface. The user can click the "Add" button with the mouse to allow the current task to access the selected Application. If the user needs to delete the application access permission of the Task, you need to select this configuration in the list below the figure by left-clicking with the left mouse button, click on the "Del" button is needed. How many Applications each Task needs to access depends on the needs of the user and the Task.

5.2.5 Comment window of task

下图显示了 Task 配置界面中的注释填写窗口。用户可以在其中填写关于当前任务的描述信息。这部分信息不会在操作系统的配置文件中生成，只作为提示信息在配置过程中提醒用户。是否需要填写注释信息由用户根据实际需要而定。

Figure below shows the comment window in the Task configuration interface. Users can fill in the description information about the current task into the window. This information is not generated in the configuration file, but only as a reminder to the user in the configuration process. Whether the user need to fill the comment information is determined by the user according to the actual needs.

对于操作系统中其他对象的配置界面，存在同样的注释填写窗口，作用与任务的注释填写窗口相同，在操作系统的其他章节不再重复描述。

For other objects in the operating system configuration interface, there is the same comment to fill in the window, the comment window of other objects in the other chapters of the operating system will been no longer repeated describe.

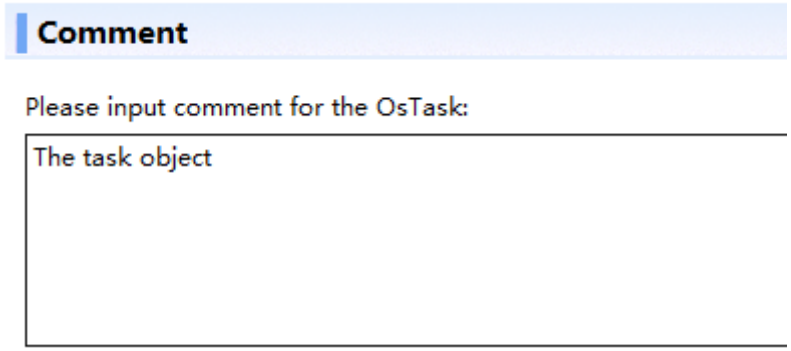


Figure5-18 comment window of task

5.3 Resource configuration interface

用户可通过鼠标右键单击树形菜单“Os -> Resource”，然后鼠标左键单击弹出的“Add Resource”按钮在操作系统中建立资源。建立资源时，需要在弹出的命名窗口中输入资源名称。之后，新建立的资源将作为树形结构的第三级菜单出现在第二级菜单“Resource”下面。用户可以通过鼠标右键单击“Os -> Resource -> 资源名称”，然后在弹出的选项中通过鼠标左键对资源进行删除或者重命名。以上内容的图示信息请参考章节 5.2 中图 9—图 11。

The user can right-click the tree menu "Os-> Resource" and then click the "Add Resource" button popped up to set up the resource in the operating system. When building a resource, you need to enter a resource name in the named window popped up. After that, the newly established resources will appear as the third level menu of the tree structure under the second level menu "Resource". The user can right click "Os-> resource-> Resource name" and then delete or rename the resource with the pop up option. Please refer to section 5.2 for detailed information above in Figure 9 - Figure 11.

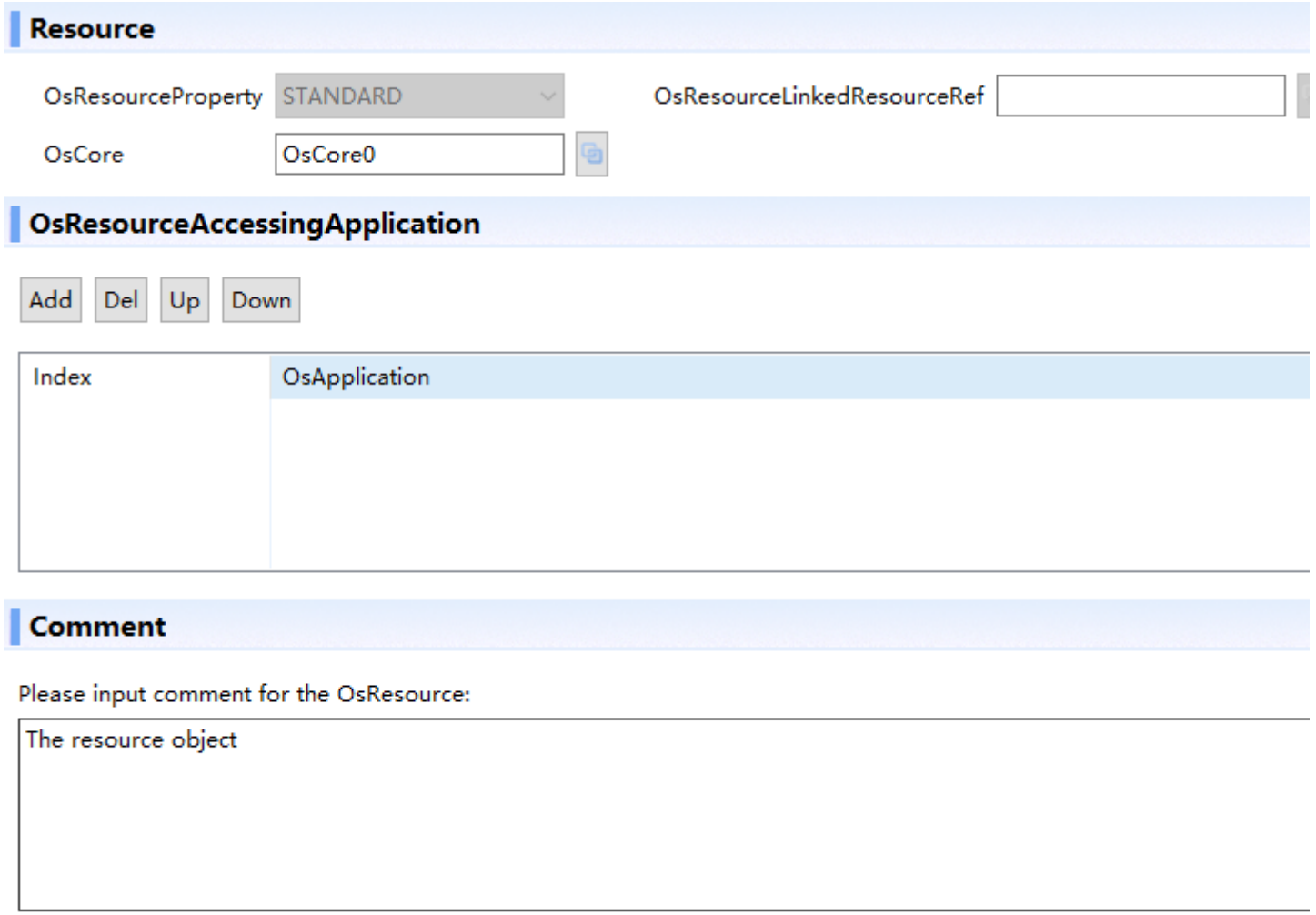


Figure5-19 Resource basic property configuration

Figure above shows the two resource base property configuration options in the resource configuration interface, each option is shown in table below.

Table5-9 basic attribute parameter information of resource

Name	Type	Range	Default Value	Description
OsResourceProperty	Enum	STANDARD/ LINKED/ INTERNAL	STANDARD	选择资源的类型。 Select the type of resource. STANDARD：该资源为标准资源； STANDARD: this resource is STANDARD resource; LINKED：该资源为链接资源； INTERNAL：该资源为内部资源。 如果当前资源被配置为链接资源，配置项(LinkedResource)将开启，需要用户配置当前资源所链接的对象。 If the current resource is configured as a link resource, the configuration item (LinkedResource) will be turned on, requiring the user to configure the object linked to the current resource. 当前版本仅支持标准资源。用户无需更改。 The current version only supports STANDARD resource. The user does not need to modify.

5.4 Event configuration interface

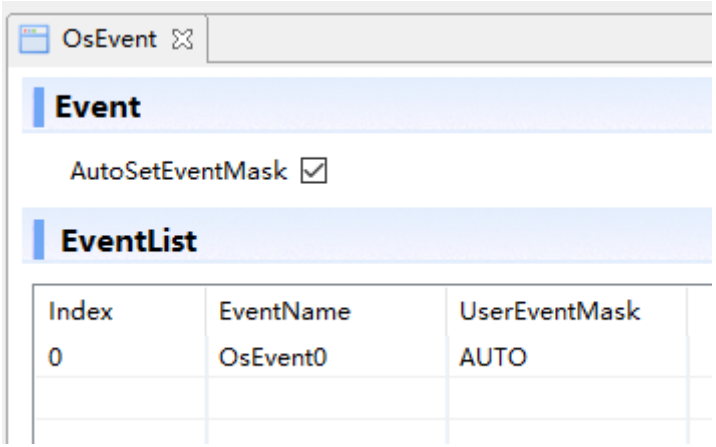


Figure5-20 Event mask property configuration

The user can click the "Os-> Event" option in the tree menu shown in Figure above to enter the event list interface, which contains the general configuration item of an event module as shown in figure above. The description for this configuration item are shown in table below.

Table5-10 information of event mask

Name	Type	Range	Default Value	Description
AutoSetEventMask	Boolean	False/True	True	选择是否自动对事件分配掩码。可选取： Select whether the event mask is automatically allocated. Can be selected: Disable：用户需手动对系统中建立的事件分配掩码； Disable: the user needs to manually assign mask to the event set up in the system; Enable：操作系统自动对系统中建立的事件分配掩码。 Enable: the operating system automatically assigns a mask to the events set up in the system.

				如用户无特殊需求，建议选择默认 Enable 选项。 If the user has no special requirement, the default “Enable” option is recommended.

用户可通过鼠标右键单击树形菜单“Os -> Event”，然后鼠标左键单击弹出的“Add Event”按钮在操作系统中建立事件。建立事件时，需要在弹出的命名窗口中输入事件名称。之后，新建立的事件将作为树形结构的第三级菜单出现在第二级菜单“Event”下面。用户可以通过鼠标右键单击“Os -> Event -> 事件名称”，然后在弹出的选项中通过鼠标左键单击“Del Event”或者是“Modify name of Event ”对事件进行删除或者重命名。以上内容的图示信息请参考章节 5.2 中图 9—图 11。

The user can right-click the tree menu "Os-> Event" and then click the "Add Event" button popped up to set up the event in the operating system. When user creates an event, user needs to enter the event name in the named window popped up. Later, the newly created event will appear as the third level menu of the tree structure under the second menu "Event". Users can click on the "Os - > Event - > Event name", and then in the pop-up option to delete or rename the event. Please refer to section 5.2 for detailed information above in Figure 9 - Figure 11.

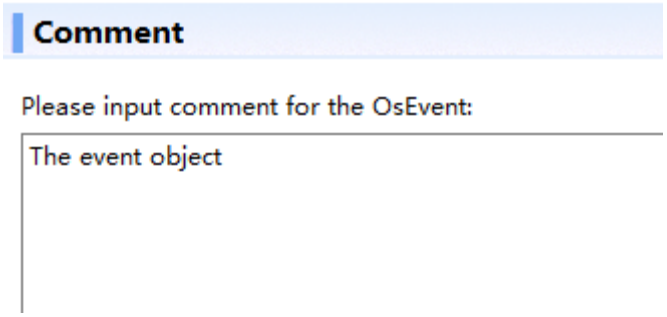


Figure5-21 Event basic property configuration

Figure above shows the basic property configuration parameter for event, each configuration item is shown in table below.

Table5-11 basic attribute parameter information of event

Name	Type	Range	Default Value	Description
OsEventMask	Enum	AUTO	AUTO	填写事件掩码。

				Fill in the event mask. 用户只有在 Event 配置界面中将配置项(AutoSetEventMask)设置为“Disable”，才会开启事件配置项 EventMask，否则系统中各个事件的掩码将由系统自动分配，用户无法对其进行修改。 only if the configuration of “AutoSetEventMask” of the event is set to "Disable", this item will open , otherwise the mask of each event in the system will be automatically assigned , and users can't modify it.

5.5 ISR configuration interface

用户可通过鼠标右键单击树形菜单“Os -> ISR”，然后鼠标左键单击弹出的“Add ISR”按钮在操作系统中建立中断。建立中断时，需要在弹出的命名窗口中输入中断名称。之后，新建立的中断将作为树形结构的第三级菜单出现在第二级菜单“ISR”下面。用户可以通过鼠标右键单击“Os -> ISR -> 中断名称”，然后在弹出的选项中通过鼠标左键单击“Del ISR”或者是“Modify name of ISR ”对中断进行删除或者重命名。以上内容的图示信息请参考章节 5.2 中图 9—图 11。

The user can right-click the tree menu "Os-> ISR" and then click the "Add ISR" button popped up to add an interrupt to operating system. When an interrupt is established, the interrupt name needs to be entered in a pop-up named window. After that, the new interrupt will appear as the third level menu of the tree structure under the second menu "ISR". Users can click "Del ISR" or "Modify the name of the ISR" to delete or rename an interrupt. Please refer to section 5.2 for detailed information above in Figure 9 - Figure 11.

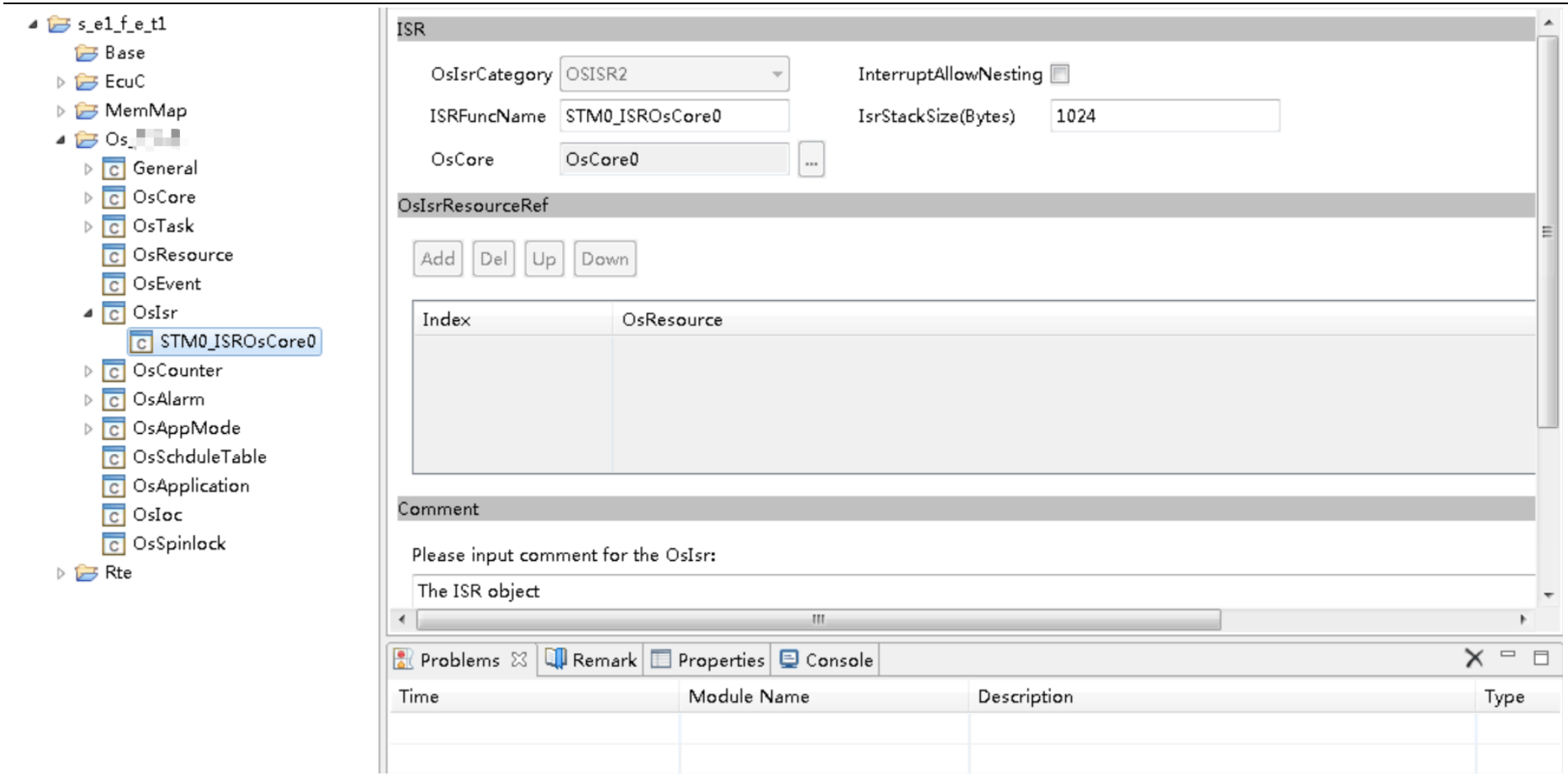


Figure5-22　ISR property configuration

上图显示的操作系统配置工具的 ISR 配置界面。除注释填写窗口外，ISR 配置界面中包含三类配置项，以下对 ISR 和 OsIsrResourceRef 这两类配置选项进行详细介绍。

Figure above shows the ISR configuration interface. The ISR configuration interface contains three types of configuration items in addition to the comment window, and the context following will descripted ISR and OsIsrResourceRef these two configuration options detailed.

注：工具默认生成系统定时器中断，因此，若芯片支持 MCAL，则用户无需在 MCAL 中配置相关中断名称，中断优先级等，由操作系统完成系统定时器中断的初始化。

Note: The tool will default to generating System Timer interrupt. Therefore, if the chip supports MCAL, users do not need to configure relevant interrupts name, interrupt priorities, etc. in MCAL. The operating system will complete the initialization of System Timer interrupt.

5.5.1 Configuration option of InterruptStackMerge

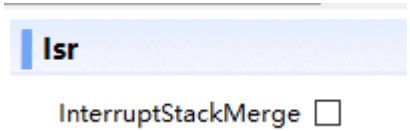


Figure5-23 Configuration option of InterruptStackMerge

Figure above shows the Configuration option of InterruptStackMerge is shown in table below.

Table5-12 information of event mask

Name	Type	Range	Default Value	Description
InterruptStackMerge	Boolean	False/True	False	当所有中断都没有嵌套时，可以选择此选项合并所有中断堆栈以节省 Ram 空间。合并的中断堆栈大小是所有已配置中断堆栈的最大值。 When all interrupts cannot be nested, this option can be selected to merge all interrupt stacks to save Ram space. The merged interrupt stack size is the maximum value of all configured interrupt stacks.

5.5.2 ISR basic property configuration

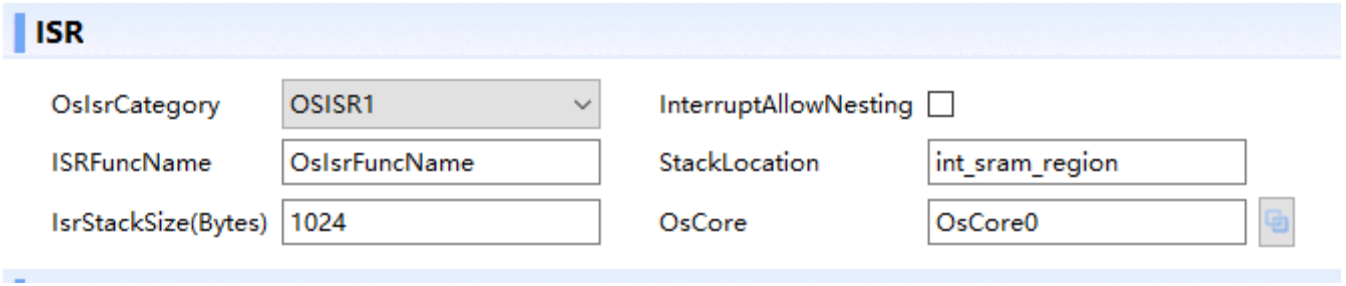


Figure5-24 ISR basic property configuration

Figure above shows the six ISR basic property options in the ISR configuration interface, and each configuration item information is shown in table below.

Table5-13 basic attribute parameter information of ISR

Name	Type	Range	Default Value	Description
OsIsrCategory	Enum	OSISR1/ OSISR2	OSISR1	选择中断类别，可取值： Select interrupt category, which can be evaluated: 1：该中断为第一类中断； OSISR1: the interruption is the ISR category 1; 2：该中断为第二类中断。 OSISR2: the interruption is the ISR category 2; 第一类中断的中断服务函数中不允许用户使用操作系统的服务函数，第二类中断可以使用。用户应根据应用程序的实际需要选择中断类别。 The first type of interrupt service function does not allow the user to use the operating system's service function, and the

				second type interrupts can be used. Users should select interrupt categories according to the actual needs of the application.
InterruptAllowNesting	Boolean	False/True	False	选择中断是否允许嵌套。 Select whether the interrupt is allowed nesting.
ISRFuncName	String	user-defined	--	填写用户自定义的中断服务函数名称。 Fill in the user-defined interrupt service function name.
ISRStackSize	Int	256--32768	1024	配置中断栈大小。 Configure the size of interrupt stack .

5.5.3 Chip configuration for ISR

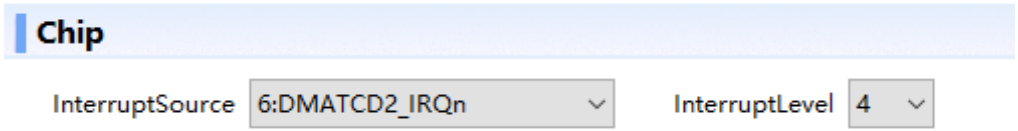


Figure5-25 Chip configuration items for ISR

Figure above shows the parameter of Chip configuration, for ISR and the configuration information is shown in table below.

Table5-14 ChipRelation configuration parameter information

Name	Type	Range	Default Value	Description
InterruptSource	Enum	--		Choose the hardware interrupt source of this ISR. 选择当前 ISR 的硬件中断源

				If a core interrupt resource is selected, the properties ISRCategory, Interrupt Level and InterruptAllowNesting are no longer necessary to configure. 如果一个核中断源被选择，ISR 类别，中断优先级和中断嵌套不再需要配置。
InterruptLevel	Int	1--255	1	Choose the priority of this ISR. 选择当前 ISR 的中断优先级 Note: Any ISR of category 1 must have higher priorities than all ISRs of category 2.

5.5.4 Resource allocation options of ISR

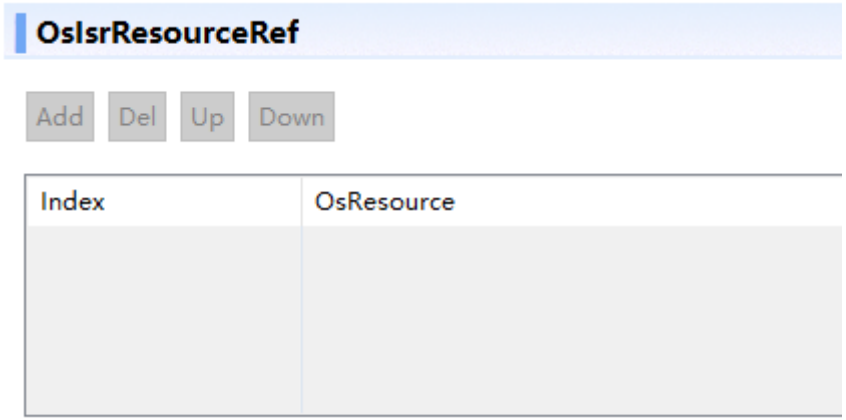


Figure5-26 Resource allocation options of ISR

上图显示了 ISR 配置界面中的资源分配选项。用户可以在“Resource”下拉框中选择需要分配给当前中断服务函数的资源，然后鼠标左键

单击“Add”按钮，将资源分配给当前 ISR。如果用户需要将已经分配给当前中断服务函数的资源删除，则需要在图下方的资源列表中通过鼠标左键单击的形式选中需要被删除的资源，然后鼠标左键单击“Del”按钮将其删除。每个中断服务函数需要分配多少资源由用户根据应用程序的需求而定。

Figure above shows the resource allocation options in the ISR configuration interface. The user can select the resource that needs to be allocated to the current interrupt service function in the "Resource" drop-down box, then click the "Add" button to assign the resource to the current ISR If users need to delete resources which have been assigned to the current interrupt service function, users need to choose the resource in the list below in figure 24 and then click the “Del” button. How much resources each interrupt service function needs to allocate to are determined by users' requirements for the application.

5.6 Alarm configuration interface

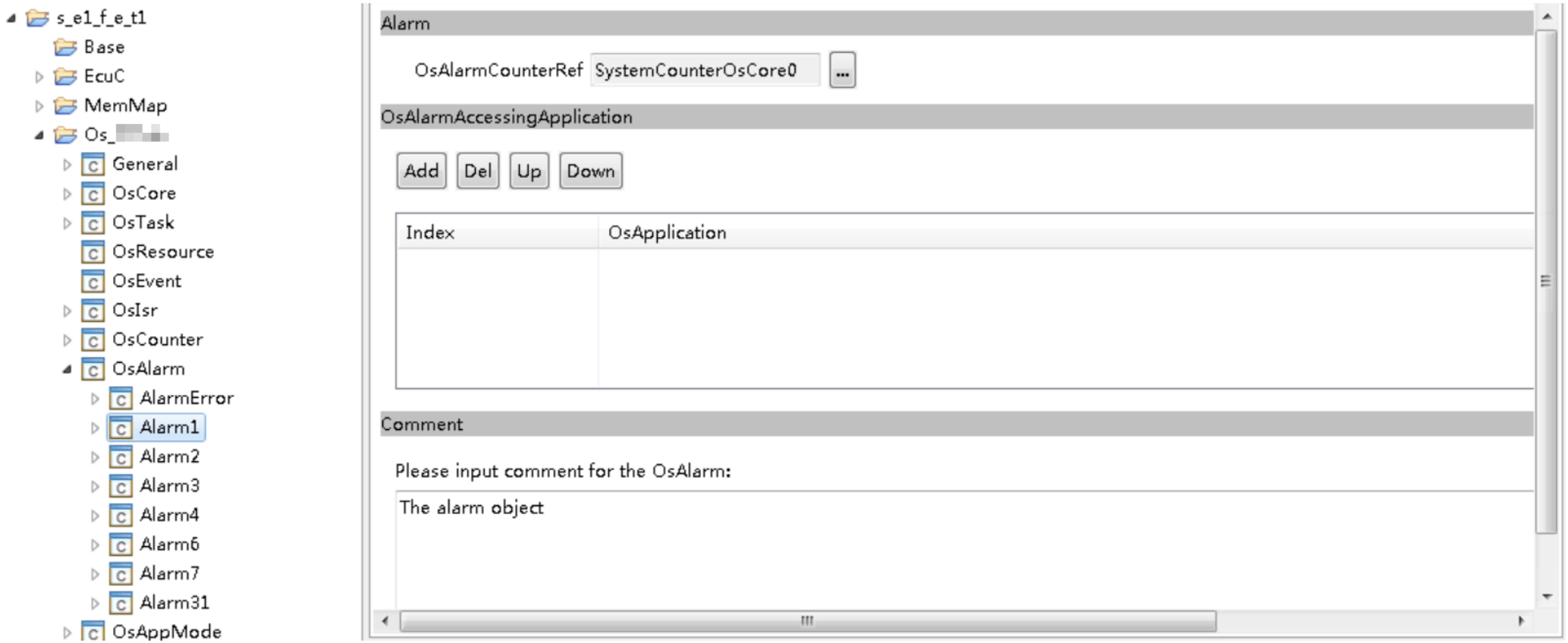


Figure5-27 alarm general configuration interface

用户可通过鼠标右键单击树形菜单“Os -> Alarm”，然后鼠标左键单击弹出的“Add Alarm”按钮在操作系统中建立报警。建立报警时，需要在弹出的命名窗口中输入报警名称。之后，新建立的报警将作为树形结构的第三级菜单出现在第二级菜单“Alarm”下面。用户可以通过鼠标右键单击“Os -> Alarm -> 报警名称”，然后在弹出的选项中通过鼠标左键单击“Del Alarm”或者是“Modify name of Alarm”对报警进行删除或者重命名。以上内容的图示信息请参考章节 5.2 中图 9—图 11。

The user can right click the tree menu "Os-> Alarm", and then click the "Add Alarm" button to set up the Alarm in the operating system. When setting up the alarm, you need to enter the alarm name in the named window popped up. After that, the newly established alarm will appear as the third level menu

of the tree structure under the second menu "Alarm". After clicking the button of Os - > Alarm - > Alarm name ", users can click "Del Alarm", "Modify the name of Alarm" or "Add Os AlarmAutostart" to delete alarm, rename alarm or set alarm to auto start. Please refer to section 5.2 for detailed information above in Figure 9 - Figure 11.

5.6.1 Basic property configuration of alarm

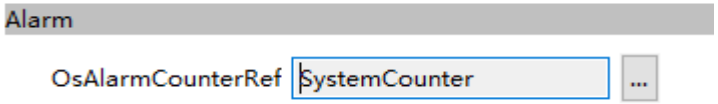


Figure5-28 Basic property configuration of alarm

Figure above shows the three alarm basic property configuration options in the alarm configuration interface, as shown in table below.

Table5-15 basic attribute parameter information of alarm

Name	Type	Range	Default Value	Description
OsAlarmCounterRef	Ref	--	SystemCounter	选择报警所使用的计数器。 Select the counter used by the alarm.

5.6.2 Configuration item of alarm action

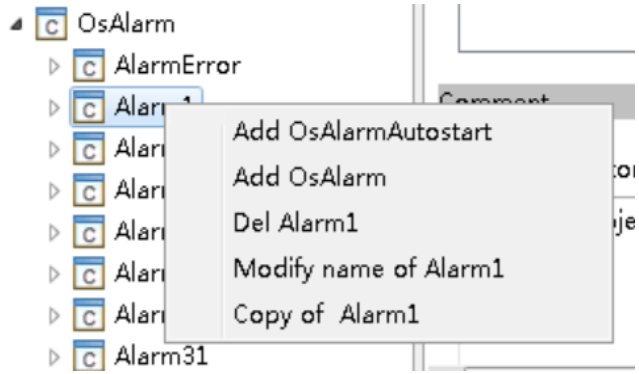


Figure5-29 configuration parameters of alarm action

Figure above shows the configuration options for alarm actions, Users can choose the corresponding alarm action according to their own design.

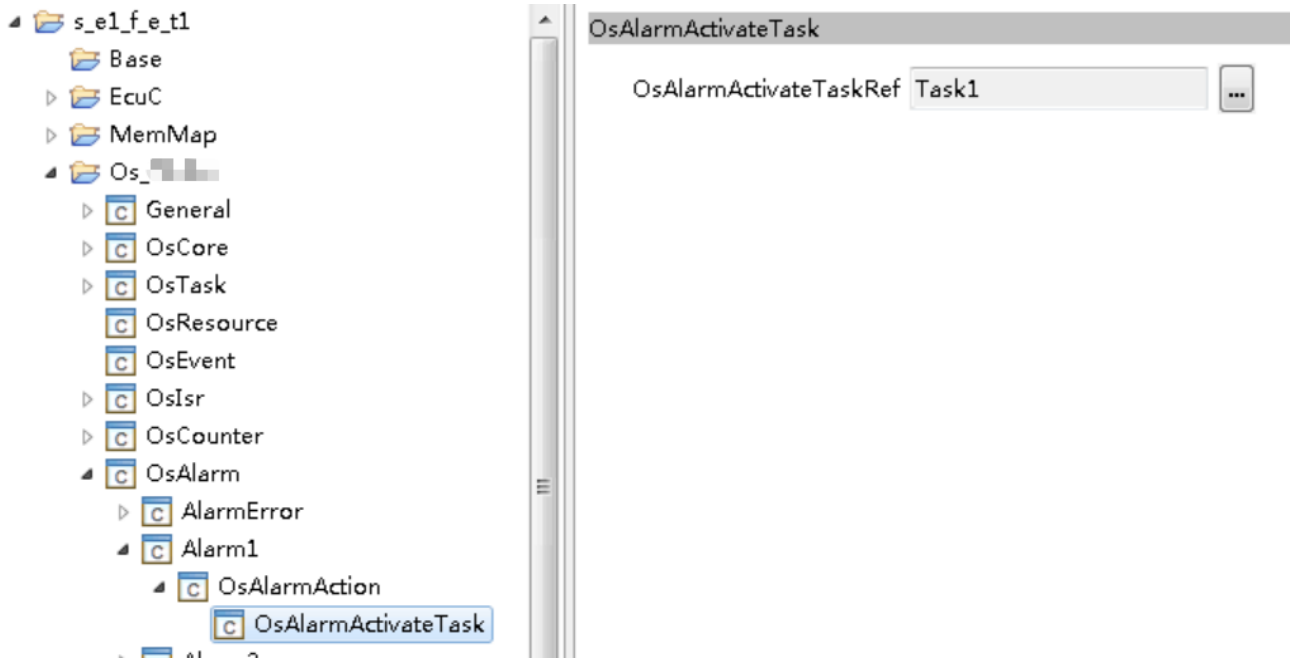


Figure5-30 configures the alarm action is to activate a task

上图显示了配置报警动作为激活任务时对应的配置界面，用户可以在 OsAlarmActivateTask 中选择对应的任务。

Figure above shows the configuration interface in which users can choose the task that has been configurated to activate a task. The user can select the corresponding task in “OsAlarmActivateTask”.

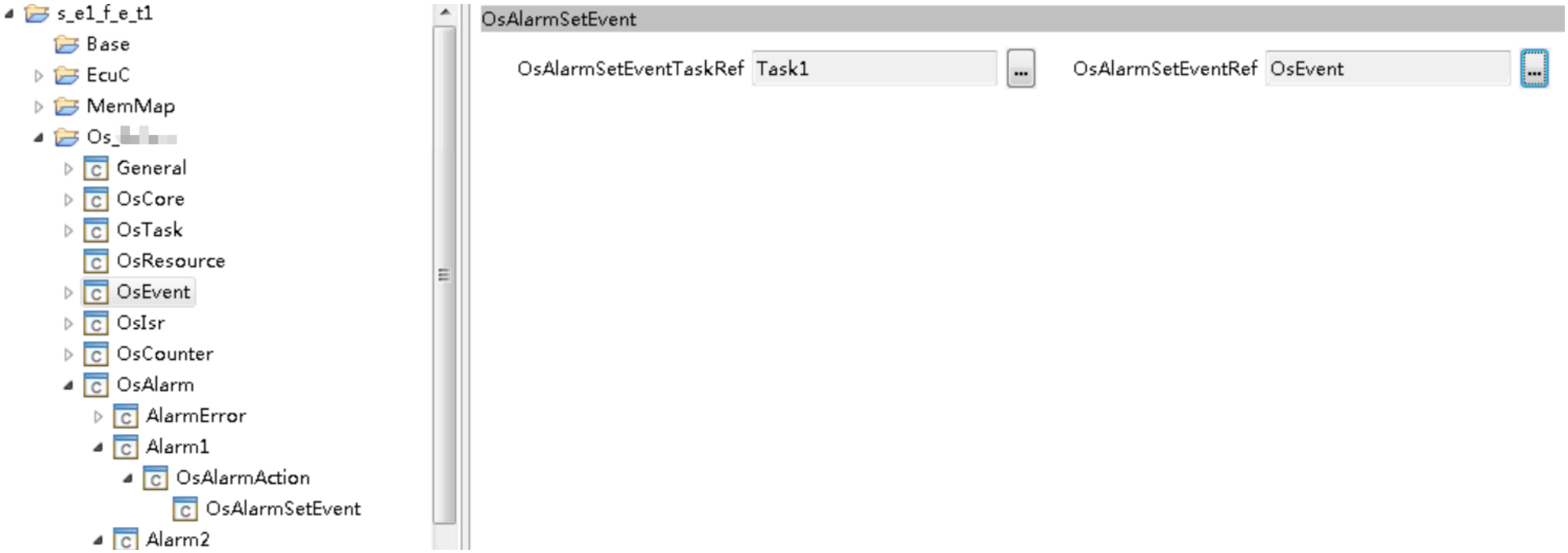


Figure5-31 configures the alarm action is to set event

上图显示了配置报警动作为设置事件时对应的配置界面，用户可以在 OsAlarmSetEvent 中选择对应的被设置的事件及接收该事件的任务。

Figure above shows the configuration interface for configuring the alarm action to set up the event, and the user can select the corresponding set of events and the task to receive the event in the “OsAlarmSetEvent”.

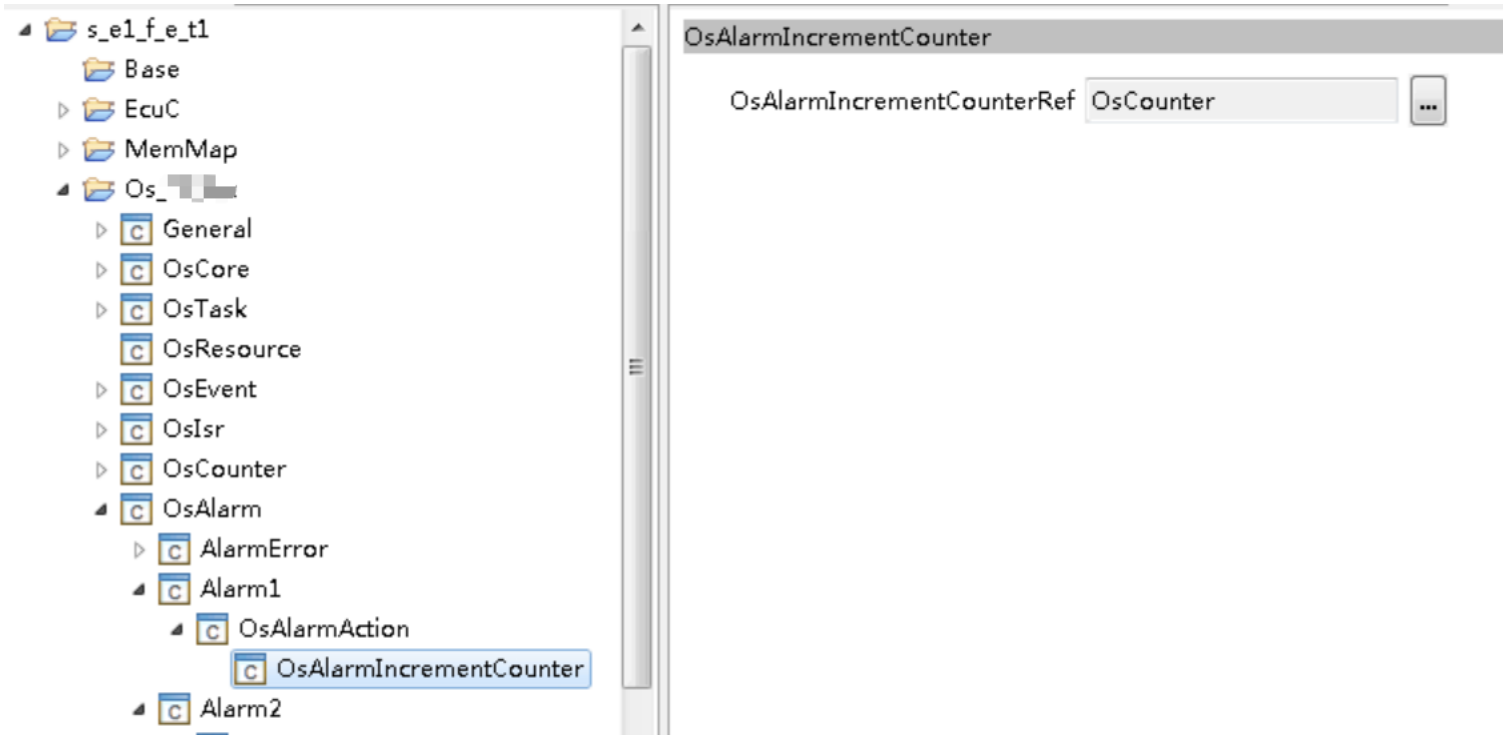


Figure5-32 configures the alarm action is to IncrementCounter

上图显示了配置报警动作为增加软件计数器值时对应的配置界面，用户可以在 OsAlarmIncrementCounterRef 中选择对应的软件计数器。

Figure above shows the configuration interface for configuring the alarm action to increment counter, the user can choose the corresponding software counter in “OsAlarmIncrementCounterRef”.

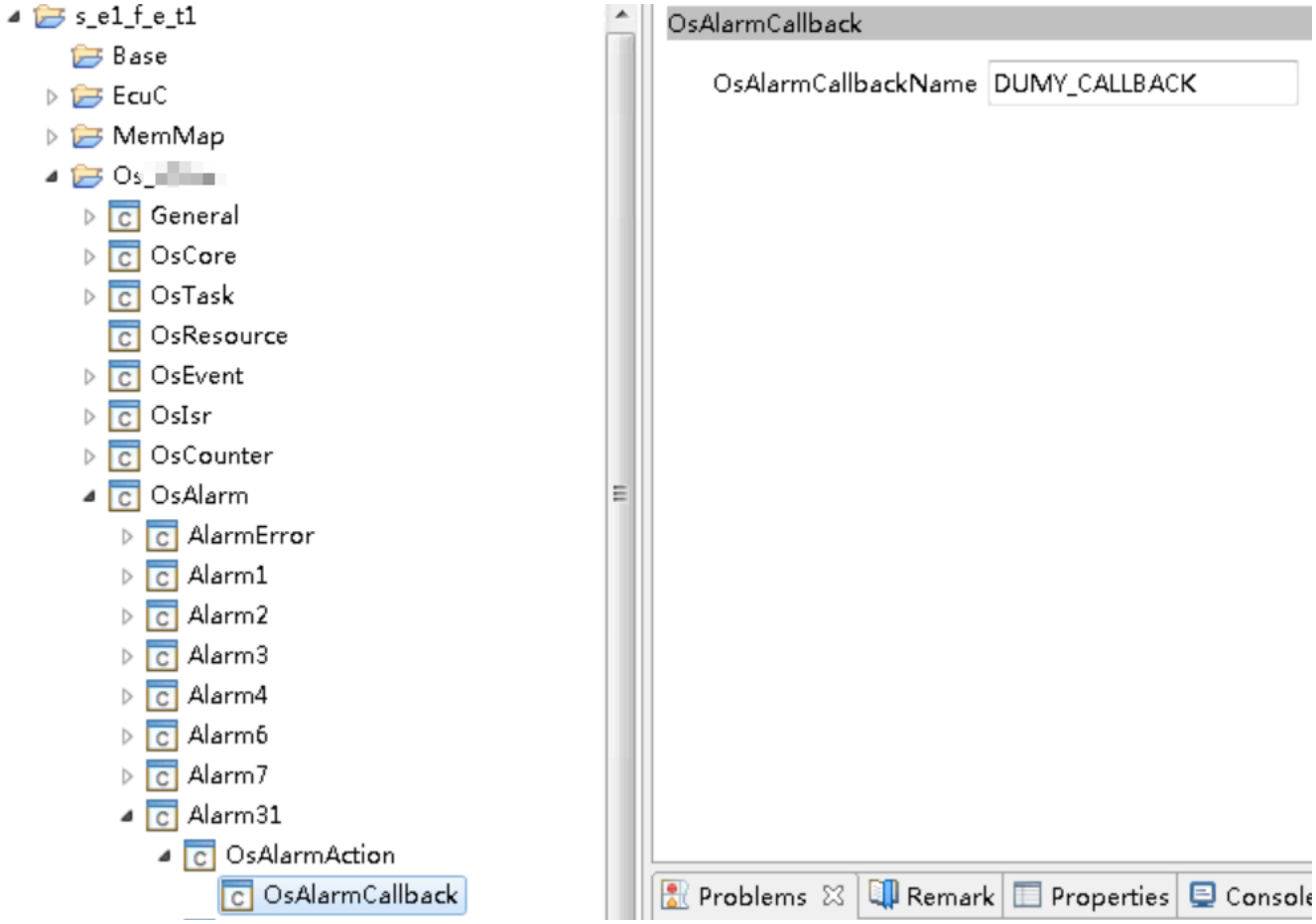


Figure5-33 configures the alarm action is to call back

上图显示了配置报警动作为调用回调函数时对应的配置界面，用户可以在 OsAlarmCallbackName 中选择对应的回调函数函数名。

Figure above shows the configuration interface for configuring the alarm action to call back, and the user can select the corresponding call back in the “OsAlarmCallbackName”.

5.6.3 Configuration option of auto start alarm

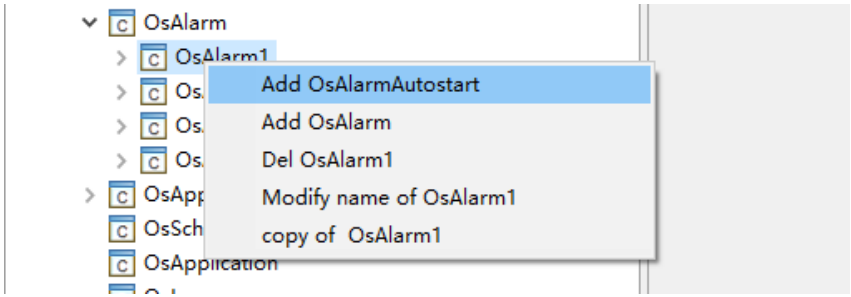


Figure5-34 configuration option of auto start alarm

User can configure the alarm to auto start by clicking "Add Os AlarmAutostart" in figure above.

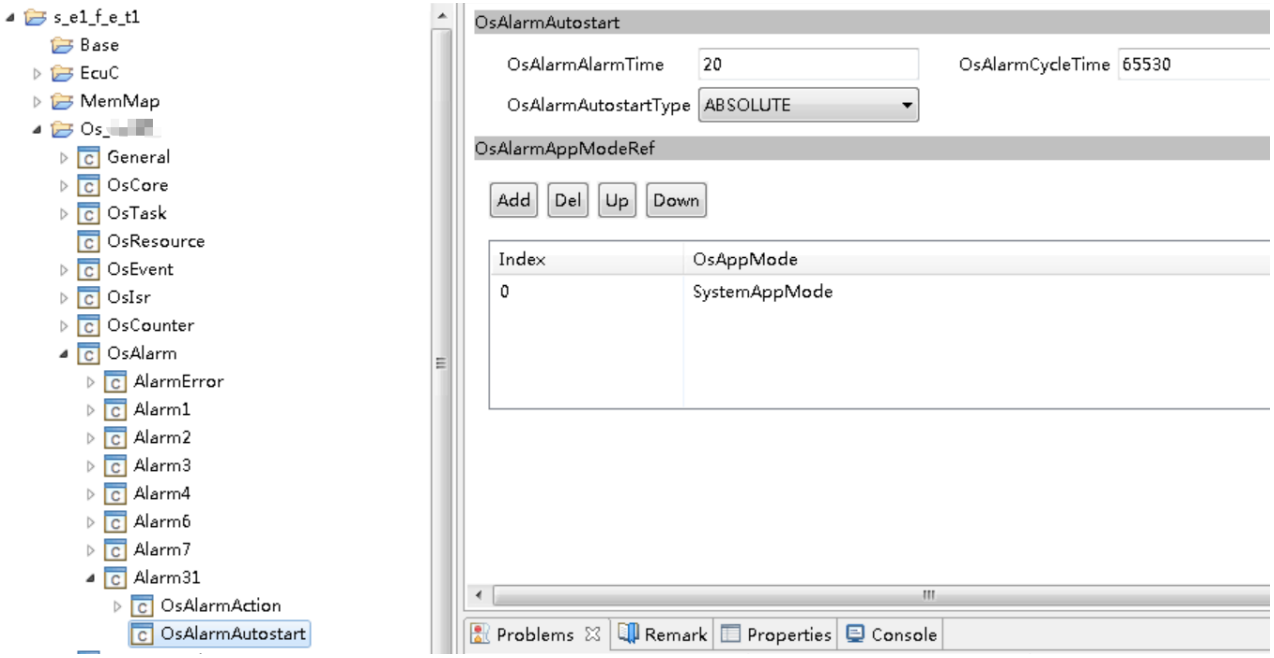


Figure5-35 configuration parameters of auto start alarm

Figure above shows the configuration interface when the alarm is configured as auto-start, and each configuration item information is shown in table below.

Table5-16 basic attribute parameter information of auto-start alarm

Name	Type	Range	Default Value	Description
OsAlarmAlarmTime	Int	1--65535	1	填写报警第一次发生的时间，以计数器的 tick 值表示。可取值。 Fill in the tick value of the counter that first occurrence of the alarm.
OsAlarmCycleTime	Int	1--65535	1	选择报警周期性发生的周期值，以计数器的 tick 值表示。 Select the periodic value of the alarm, which is represented in tick value of the counter. 该配置项为 0 表示当前报警为非周期性报警。 This configuration item is 0 indicating that the current alarm is not periodic alarm.
OsAlarmAutoStartType	Enum	ABSOLUTE/ RELATIVE	ABSOLUTE	选择自启动类型，可取值： Select the auto-start type, can be selected: ABSOLUTE：以绝对时间自启动 ABSOLUTE: start at absolute time. RELATIVE：以相对时间自启动 RELATIVE: start at relative time.

5.7 Counter configuration interface

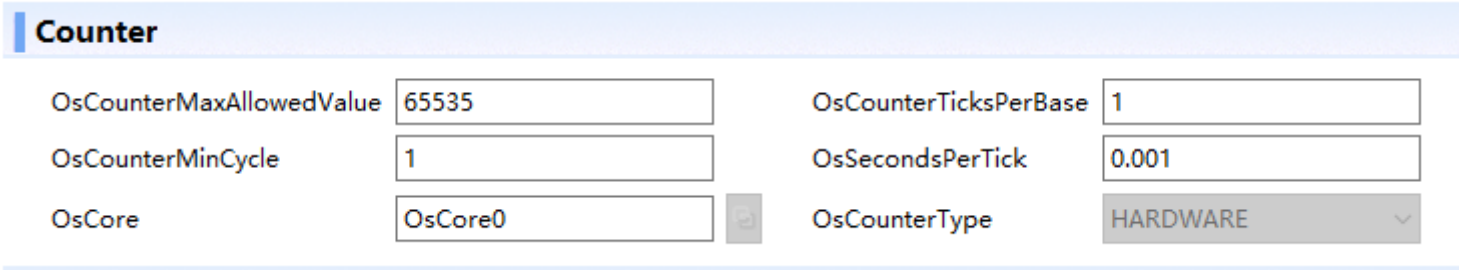


Figure5-36 Basic configuration parameters of counter

Figure above shows the five basic property configuration options of counter in the counter configuration interface, each option is shown in table below.

Table5-17 basic attribute parameter information of counter

Name	Type	Range	Default Value	Description
OsCounterMaxAllowedValue	Long	1--4294967295	65535	填写计数器的最大计数值。 Fill in the maximum value of the counter.
OsCounterTicksPerBase	Int	1--65535	1	填写计数器每计数一次，硬件定时器需要累加的 tick 数量。 The hardware timer needs a cumulative tick value for the count.
OsCounterMinCycle	Int	1--65535	1	填写使用当前计数器的周期性报警的周期最小值。 Fill in the minimum cycle value of the periodic alarm using the current counter.
OsSecondsPerTick	Int	1--65535	1	用户根据实际应用配置 Configured according to the actual application by users.
OsCounterType	Enum	SOFTWARE/ HARDWARE	SOFTWARE	选择计数器类型. Select the counter type. 当前版本仅支持软件计数器。用户无需更改。

				The current version only supports SOFTWARE counter. The user does not need to modify.
OsCore	Ref			

用户在配置时需要注意，根据 OSEK 操作系统标准的规定，配置工具默认建立了一个系统计数器 SystemCounter。用户可以对 SystemCounter 的参数值进行修改，但是不能将其删除。

The user needs to be aware that Configurator has set up a SystemCounter by default according to the OSEK operating system standard, the user can modify the parameters of SystemCounter, but cannot delete them.

5.8 AppMode configuration interface

OS 支持最多 256 种应用模式。用户可以在系统配置阶段自定义应用模式。OS 的运行必须是在某一种应用模式下。如果用户没有在系统中设置任何自定义的应用模式，OS 提供一种默认的应用模式。此时系统将工作在这种默认模式下。更多关于系统应用模式的信息请参阅章节 3.7。

OS supports up to 256 application modes. Users can customize the application modes in the system configuration phase. The operation of OS must be in one application mode. If the user does not set any custom application modes in the system, OS provides a default application mode. In this case, the system will work in this default mode. For more information on system application modes, see chapter 10 of this article.

用户可通过鼠标右键单击树形菜单“Os -> AppMode”，然后鼠标左键单击弹出的“Add AppMode”按钮在操作系统中建立应用模式。建立应用模式时，需要在弹出的命名窗口中输入应用模式名称。之后，新建立的应用模式将作为树形结构的第三级菜单出现在第二级菜单“AppMode”下面。用户可以通过鼠标右键单击“Os -> AppMode -> 应用模式名称”，然后在弹出选项中通过鼠标左键单击“Del AppMode”或者是“Modify name of AppMode”对应用模式进行删除或者重命名。以上内容的图示信息请参考章节 5.2 中图 9—图 11。

The user can right-click the tree menu "Os-> AppMode", and then click the "Add AppMode" button popped up to set up the application mode in the operating system. When building an application mode, the user need to enter the name of application mode in the pop-up name window. After that, the new application pattern will appear as the third level menu of the tree structure under the second menu "AppMode". Users can through the right mouse-click on the button of "Os - > AppMode - > application model name" to choose application mode, and then left-click "Del AppMode" or "Modify the name of AppMode" in the pop-up option to delete or rename application mode. Please refer to section 5.2 for detailed information above in Figure 9 - Figure 11.

用户在操作系统中建立了应用模式之后，无需配置任何与应用模式相关的参数，应用模式配置界面上只有当前应用模式的 ID 信息，由配置工具自动分配，用户无需对其进行配置。

After setting up an application mode by users in the operating system, there is only the current ID information of the application mode in configuration interface and without any relevant parameters to be set up by users, the ID information is automatically assigned by Configurator, users don't need to configure it.

5.9 ScheduleTable configuration interface

用户可以通过鼠标右键单击树形菜单“Os -> ScheduleTable”，然后鼠标左键单击弹出的“Add ScheduleTable”按钮在操作系统中建立调度表。建立调度表时，需要在弹出的命名窗口中输入调度表名称。之后，新建立的调度表将作为树形结构的第三级菜单出现在第二级菜单“ScheduleTable”下面。用户可以通过鼠标右键单击“Os -> ScheduleTable -> 调度表名称”，然后在弹出的选项中通过鼠标左键单击“Del ScheduleTable”或者是“Modify name of ScheduleTable”对调度表进行删除或者重命名。

The user can right-click the tree menu "Os-> ScheduleTable" and then click the "Add ScheduleTable" button popped up to set up the ScheduleTable in the operating system. When the schedule table is set up, the schedule table name needs to be entered in the named window popped up. After that, the newly created schedule will appear as the third level menu of the tree structure under the second menu "ScheduleTable”. Users can through the right-click on the button of "Os - > ScheduleTable - > scheduling table name “to choose the schedule table, and then through the left-click of "Del ScheduleTable" or "Modify the name of ScheduleTable" in the pop-up option to delete or rename the scheduling table.

用户在创建调度表的同时，配置工具还会在该调度表下面自动创建一个第四级菜单“EPx”，作为该调度表的一个执行点。用户可以通过鼠标右键单击“Os -> ScheduleTable -> 调度表名称 -> 执行点名称”，然后在弹出的选项中通过鼠标左键单击“Add EP”按钮在该调度表中新建执行点。建立执行点时，需要在弹出的命名窗口中输入执行点名称。用户可以通过鼠标右键单击“Os -> ScheduleTable -> 调度表名称 -> 执行点名称”，然后在弹出的选项中通过鼠标左键单击“Del EP”执行点进行删除等。以上内容的图示信息请参考章节 5.2 中图 9—图 11。

After the user creates the schedule table, Configurator will automatically create a fourth-level menu "EPx" below the schedule table as an execution point for the schedule table. Users can choose the schedule table through the right-click on the button of "Os - > ScheduleTable - > scheduling table name- > expiry point name", and left-click the "Add EP" button in the pop-up option, then new expiry point is added. When establishing the expiry point, enter the expiry point name in the pop-up name window. The user can right-click "Os-> ScheduleTable -> schedule table name -> expiry point name", and then click

"Del EP" to delete expiry point. Please refer to section 5.2 for detailed information above in Figure 9 - Figure 11.

5.9.1 Basic property configuration of schedule table

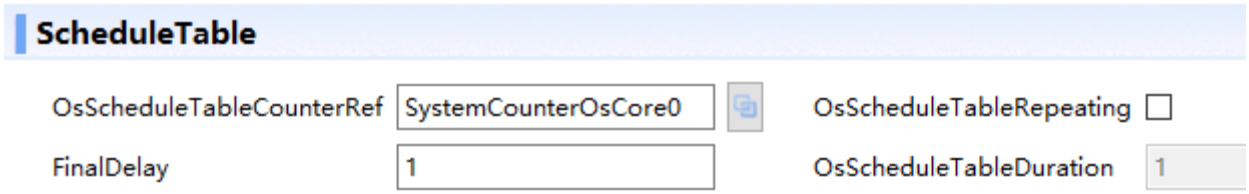


Figure5-37 basic property configuration of schedule table

Figure above shows the four alarm basic property configuration options in the schedule table configuration interface, and each option is shown in table below.

Table5-18 basic parameter information of schedule table

Name	Type	Range	Default Value	Description
OsScheduleTableCounterRef	Ref			选择调度表所使用的计数器。 This parameter contains a reference to the counter which drives the schedule table.
OsScheduleTableRepeating	Boolean	False/True	False	选择调度表是否为重复执行调度表，可勾选为： Select whether setting the schedule table to repeating mode, can be selected: 选中：调度表执行完毕后自动从头开始执行。 Selected: first expiry point on the schedule table shall be processed at final expiry point delay ticks after the final expiry point is processed. 不选中：调度表执行完毕后结束执行。

				Not selected: the schedule table processing stops when the final expiry point is processed.
FinalDelay	Long	0--65535	1	调度表执行完最后一个执行点后过多少个 tick 结束执行。 The number of tick value between the last execution point and the finishing of the schedule table.
OsScheduleTableDuration	Long	0--65535	1	用户根据实际应用配置。 Duration of the schedule table. 该配置项仅在 OsScheduleTableRepeating 选中时有效。 The configuration item is valid only when “OsScheduleTableRepeating” is selected.

5.9.2 Configuration property of auto start schedule table

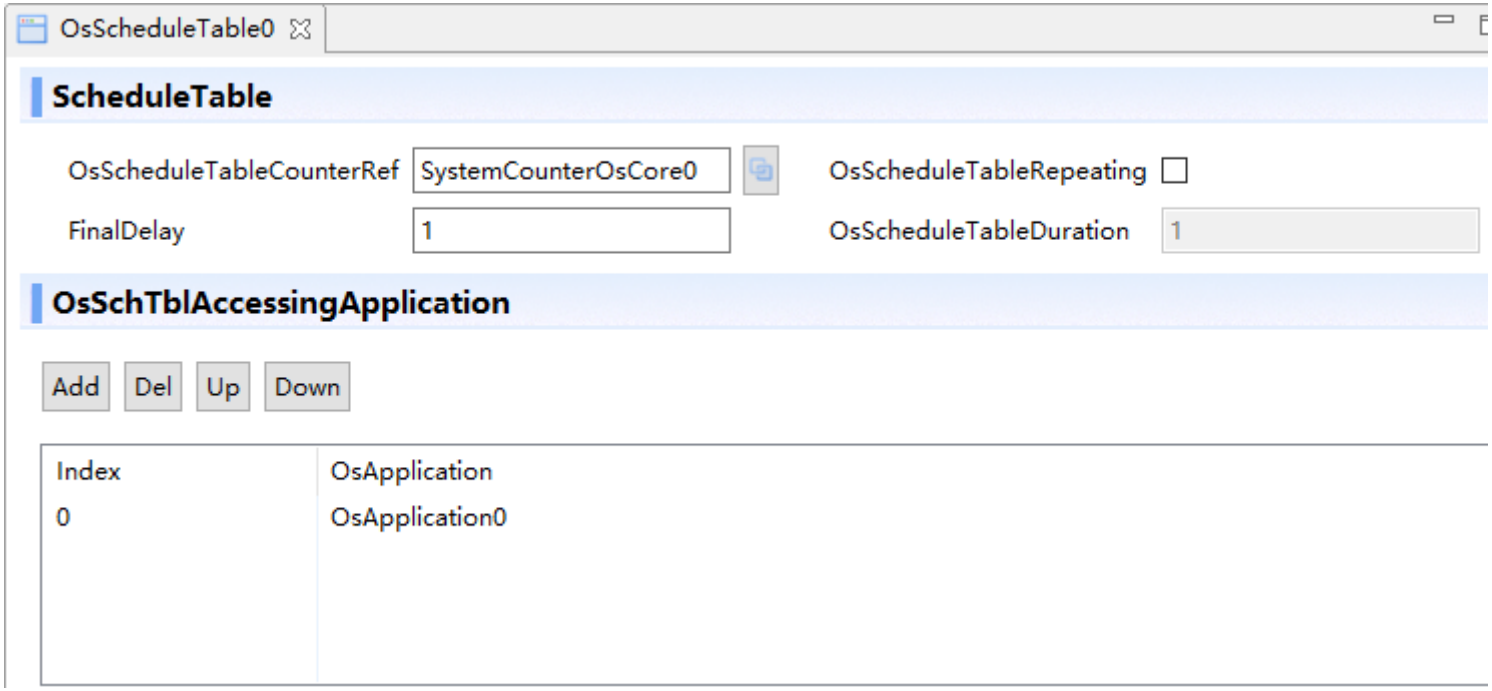


Figure5-38 configuration property of auto starts schedule table

Figure above shows that the configuration options of auto start schedule table, and each option is shown in table below.

Table5-19 parameter information of auto starts schedule table

Name	Type	Range	Default Value	Description
OsScheduleTableAutostartType	Enum	ABSOLUTE\ RELATIVE\ SYNCHRON(暂 不 支 持)	ABSOLUTE	选择自启动类型，可取值： This specifies the type of the autostart for the schedule table, can be selected： ABSOLUTE：以绝对时间自启动 ABSOLUTE：The schedule table is started during startup with

				the StartScheduleTableAbs() service. RELATIVE：以相对时间自启动 RELATIVE：The schedule table is started during startup with the StartScheduleTableRel() service.
OsScheduleTableStartValue	Int	1--65535	1	填写调度表第一次发生的时间，以计数器的 tick 值表示。 Absolute autostart tick value when the schedule table starts. Only used if the OsScheduleTableAutostartType is ABSOLUTE. Relative offset in ticks when the schedule table starts. Only used if the OsScheduleTableAutostartType is RELATIVE.
OsScheduleTableAppModeRef	Ref		SystemAppMode	选择调度表自启动所处的应用模式。 Reference in which application modes the schedule table should be started during startup

5.9.3 Property of schedule table expiry point

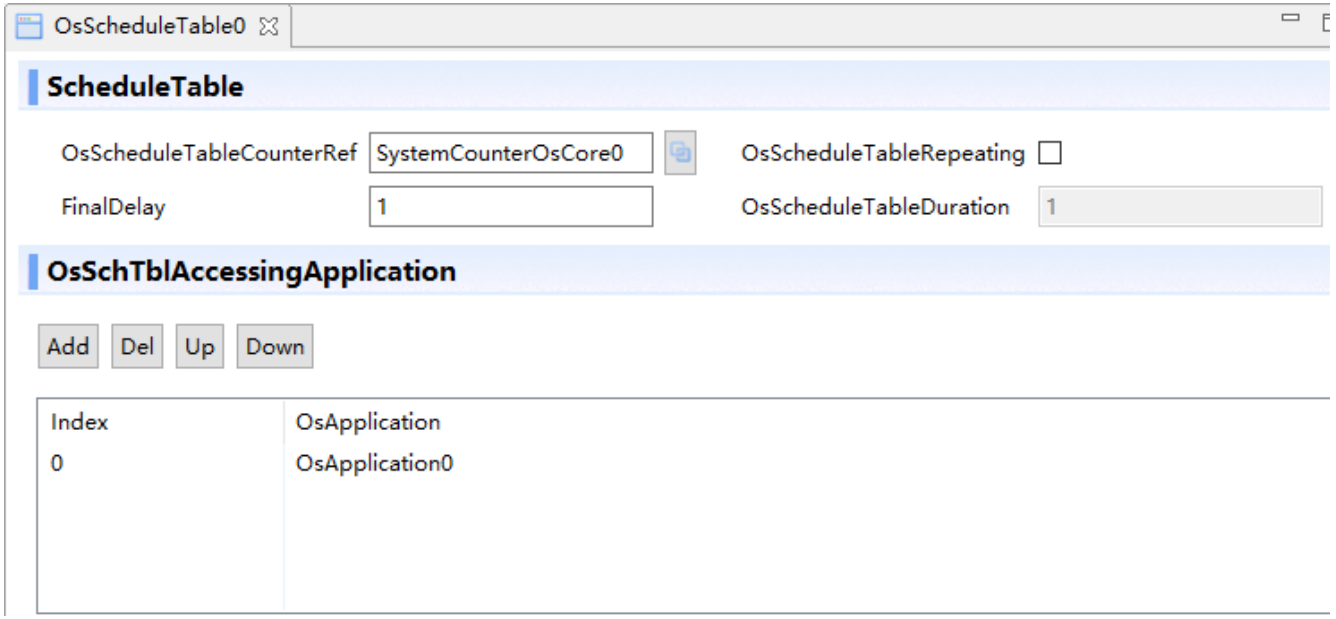


Figure5-39 property of schedule table expiry point

上图显示了 ScheduleTable 配置界面的执行点列表。该图展示了该调度表包含的所有执行点及其属性。该图不可直接修改。用户可在 EP 的配置界面修改对应属性。

Figure above shows the list of expiry points in the schedule table configuration interface. This table shows all the expiry points and their attributes. The table cannot be modified directly. The user can modify the corresponding attributes in the EP configuration interface.

5.9.4 Basic property configuration of EP

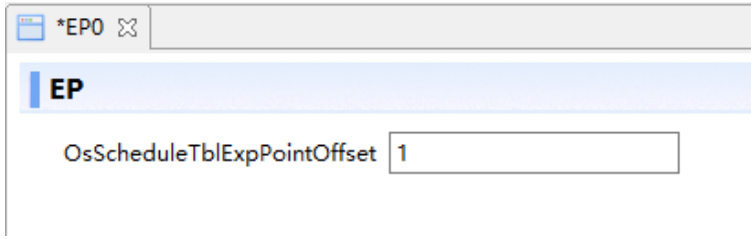


Figure5-40 property of expiry point

Figure above shows a basic property configuration option for the EP in configuration interface, as shown in table below.

Table5-20 parameter information of expiry point

Name	Type	Range	Default Value	Description
OsScheduleTblExpPointOffset	Long	1-- 4294967295	1	设置 EP 在所属 ScheduleTable 中距离起始位置的偏移值，以计数器的 tick 值表示。 Set the offset value of EP at the starting position in the ScheduleTable, in tick value of the counter. 第一个 EP：0--计数器的最大计数值。 The first EP: 0-- the maximum value of the counter. 其他 EP：与前一个 EP 差值为 1--计数器的最大计数值。 Other EP: the previous EP reduce 1--the maximum value of a counter .

5.9.5 Configuration of EP actions

Figure below shows the property configuration option for EP actions, configuration principle please reference chapter 5.6.2 for detailed information.

Figure5-41 configuration of EP actions

5.10 Application configuration interface

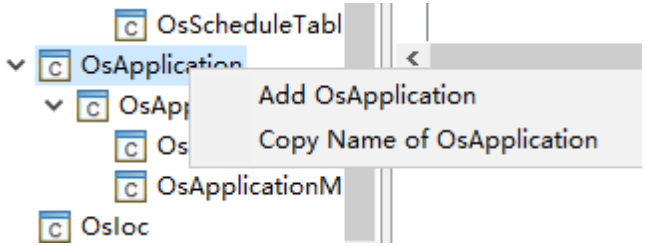


Figure5-42 Add Application

用户可以通过鼠标右键单击树形菜单“Os -> Application”，然后鼠标左键单击弹出的“Add OsApplicaton”按钮在操作系统中建立 Application。建立 Application 时，需要在弹出的命名窗口中输入 Application 名称。之后，新建立的 Application 将作为树形结构的第三级菜单出现在第二级菜单“Application”下面。用户可以通过鼠标右键单击“Os -> Application ->Application 名称”，然后在弹出的选项中通过鼠标左键单击“Del Application”或者是“Modify name of Application”对应用程序进行删除或者重命名。以上内容的图示信息请参考章节 5.2 中图 9—图 11。

The user can right-click the tree menu "Os-> Application " and then click the "Add Application " button popped up to set up the Application in the operating system. When the Application is set up, the Application name needs to be entered in the named window popped up.After that, the newly created Application will appear as the third level menu of the tree structure under the second menu " Application ".Users can through the right-click on the button of "Os - > Application - > Application name “to choose the Application, and then through the left-click of" Del Application”" or "Modify the name of Application " in the pop-up option to delete or rename the Application. Please refer to section 5.2 for detailed information above in Figure 9 - Figure 11.

5.10.1 Basic property configuration of Application

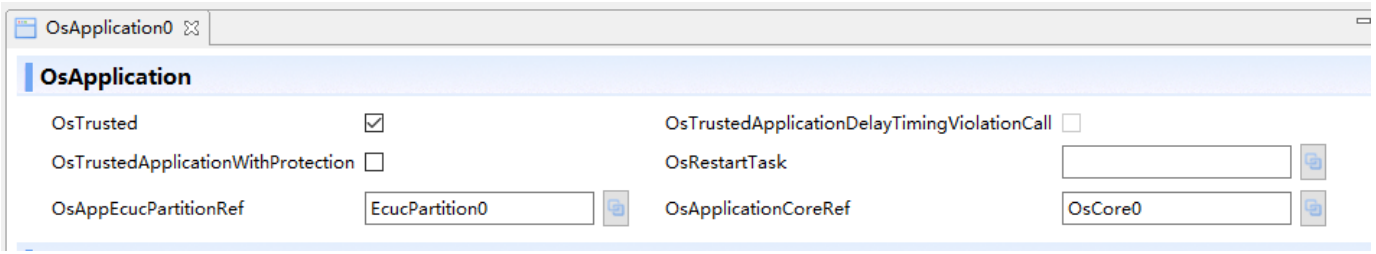


Figure5-43 Basic property configuration of Application

Figure above shows six basic Application basic property configuration options in the Application configuration interface, as is shown in table below.

Table5-21 Basic attribute parameter information of Application

Name	Type	Range	Default Value	Description
OsTrusted	Boolean	False/True	True	选择是否在 template.c 中生成触发代码。 Choose whether generate the trigger code in the template.c. 使能：计数器的触发功能在 ISR 的相关任务中生成。 Enable: trigger function for this counter will be generated in corresponding task or ISR. 关闭使能：触发功能将不会在 template.c 中生成。 Disable: trigger function will not be generated in template.c.
OsTrustedApplication DelayTimingViolaton Call	Boolean	False/True	False	用于指定是否立即引发受信任操作系统应用程序中发生的定时冲突是否会立即发生，或者是否延迟到当前任务返回才调用操作系统的应用程序（返回 CallTrustedFunction） Parameter to specify if a timing violation which occurs within a trusted OS-Application is raised immediately of if it is delayed until the current task returns to the calling OS-Application (return of CallTrustedFunction) 是：冲突、调用 ProtectionHook() 会延迟 否：定时冲突引发对 ProtectionHook()立即调用。 true: violation / call to ProtectionHook() is delayed false: timing violation cause an immediate call to the ProtectionHook().
OsTrustedApplication	Boolean	False/True	False	用于指定一个受信任的系统应用程序的执行是需要内存保

With Protection				护。 Parameter to specify if a trusted OS-Application is executed with memory protection or not. 是：系统应用程序运行在受保护的环境，这意味着写权限是受限制的。 否：应用程序拥有全部写权限（默认） true: OS-Application runs within a protected environment. This means that write access is limited. false: OS-Application has full write access (default)
OsRestartTask	Enum	Name of OsTask	--	（可选）可以将操作系统应用程序的一个任务定义为重新启动任务 Optionally one task of an OS-Application may be defined as Restart Task. Multiplicity = 1： 如果 protection hook 请求发生，重启任务会被操作系统激活。 Multiplicity = 0: 在保护错误发生之后，没有任务自动启动。 Multiplicity = 1: Restart Task is activated by the Operating System if the protection hook requests it. Multiplicity = 0: No task is automatically started after a protection error happened.
OsAppEcuPartitonRef	Ref	--	--	表示此“应用程序”所实现的 Ecuc 分区。 Denotes which "EcucPartition" is implemented by this "OSApplication".
OsCore	Enum	--	--	这个条目属所属于的核。 Which Core this Item belongs to

5.10.2 Task allocation configuration for Application

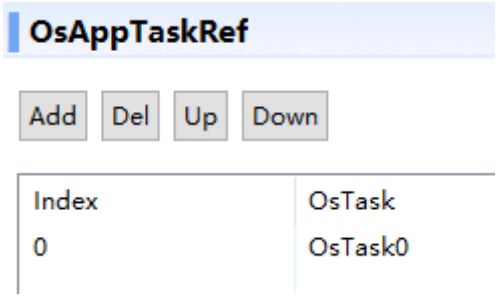


Figure5-44 The task allocation option for Application

上图显示了 Application 配置界面中的任务分配选项。上图所示的任务分配选项被开启。用户可以鼠标点击“Add”按钮，将任务列表中的任务分配给当前应用程序。如果用户需要将已经分配给当前应用程序的任务删除，则需要在图中下方的任务列表通过鼠标左键单击的形式选中需要被删除的任务，然后鼠标左键单击“Del”按钮将其删除。每个任务需要分配多少任务由用户根据应用程序的需求而定。

The figure above shows the task assignment options in the Application configuration interface. The task assignment option shown in the image above is turned on. The user can click the "Add" button to assign the task in the task list to the current application. If the user needs to delete the task that has been assigned to the current application, the task to be deleted needs to be selected in the task list at the bottom of the figure by left-clicking, and then clicking the "Del" button with the left mouse button to delete it. How many tasks need to be assigned to each task is determined by the user based on the needs of the application.

5.10.3 Alarm allocation configuration for Application

Figure5-45 The Alarm allocation option for Application

上图显示了 Application 配置界面中的报警分配选项。上图所示的报警分配选项被开启。用户可以鼠标点击“Add”按钮，将报警列表中的报警分配给当前应用程序。如果用户需要将已经分配给当前应用程序的报警删除，则需要在图中下方的报警列表通过鼠标左键单击的形式选中需要被删除的报警，然后鼠标左键单击“Del”按钮将其删除。每个报警需要分配多少报警由用户根据应用程序的需求而定。

The figure above shows the alarm assignment options in the Application configuration interface. The alarm assignment option shown in the image above is enabled. The user can click the "Add" button to assign the alarm in the alarm list to the current application. If the user needs to delete the alarm that has been assigned to the current application, you need to select the alarm to be deleted in the alarm list at the bottom of the figure by left-clicking it, and then click the Del button with the left mouse button to delete it. How many alarms need to be assigned to each alarm is determined by the user based on the needs of the application

5.10.4 AppCounter allocation configuration for Application

上图显示了 Application 配置界面中的应用计数器分配选项。上图所示的应用计数器分配选项被开启。用户可以鼠标点击“Add”按钮，将

应用计数器列表中的应用计数器分配给当前应用程序。如果用户需要将已经分配给当前应用程序的应用计数器删除，则需要在图中下方的应用计数器列表通过鼠标左键单击的形式选中需要被删除的应用计数器，然后鼠标左键单击“Del”按钮将其删除。每个应用计数器需要分配多少应用计数器由用户根据应用程序的需求而定。

The preceding figure shows the application counter allocation options in the Application configuration interface. The app counter allocation option shown in the image above is turned on. The user can click the "Add" button to assign the application counter in the application counter list to the current application. If the user needs to delete the application counter that has been assigned to the current application, you need to select the application counter to be deleted in the application counter list at the bottom of the figure by left-clicking it, and then click the Del button to delete it. How many app counters need to be assigned to each app counter is up to the user based on the needs of the application.

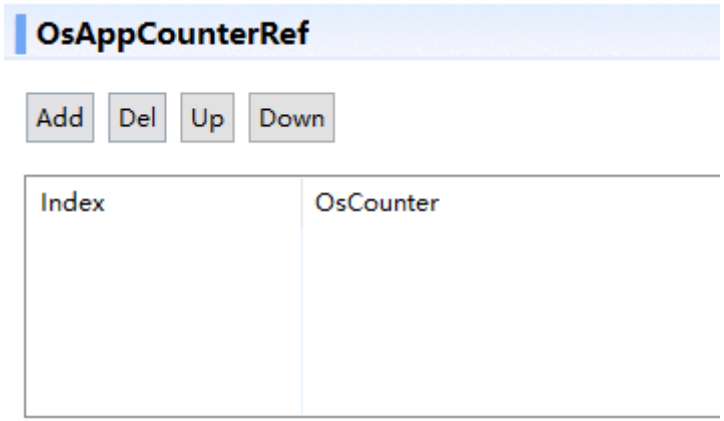


Figure5-46 The AppCounter allocation option for Application

5.10.5 AppIsr allocation configuration for Application

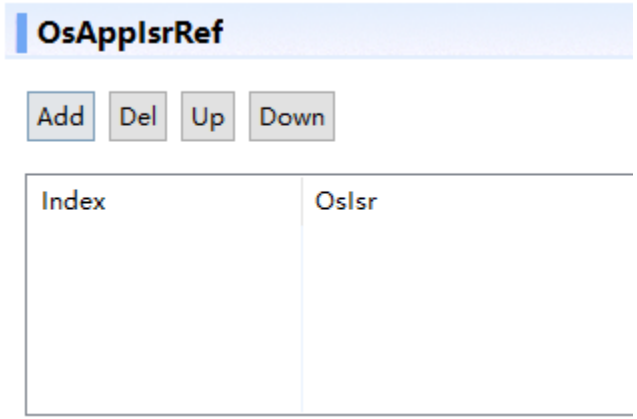


Figure5-55 The AppIsr allocation option for Application

上图显示了 Application 配置界面中的 AppIsr 分配选项。上图所示的应用 AppIsr 分配选项被开启。用户可以鼠标点击“Add”按钮，将 AppIsr 列表中的 AppIsr 分配给当前应用程序。如果用户需要将已经分配给当前应用程序的 AppIsr 删除，则需要在图中下方的 AppIsr 列表通过鼠标左键单击的形式选中需要被删除的 AppIsr，然后鼠标左键单击“Del”按钮将其删除。每个 AppIsr 需要分配多少 AppIsr 由用户根据应用程序的需求而定。

The figure above shows the AppIsr allocation options in the Application configuration interface. The AppIsr allocation option for the application shown in the image above is enabled. The user can click the "Add" button to assign AppIsr in the AppIsr list to the current application. If the user needs to delete the AppIsr that has been assigned to the current application, you need to select the AppIsr to be deleted in the AppIsr list at the bottom of the figure by left-clicking, and then click the "Del" button with the left mouse button to delete it. How many AppIsrs need to be allocated per AppIsr is determined by the user based on the needs of the application.

5.10.6 AppIsr allocation configuration for Application

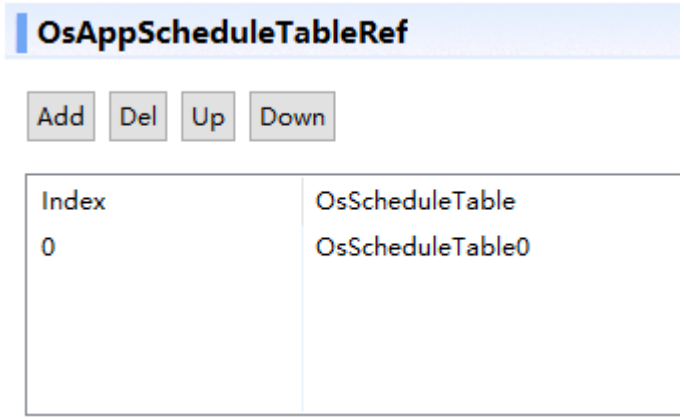


Figure5-56 The AppScheduleTable allocation option for Application

上图显示了 Application 配置界面中的 App 调度表分配选项。上图所示的 App 调度表分配选项被开启。用户可以鼠标点击“Add”按钮，将 App 调度表列表中的 App 调度表分配给当前应用程序。如果用户需要将已经分配给当前应用程序的 App 调度表删除，则需要在图中下方的 App 调度表列表通过鼠标左键单击的形式选中需要被删除的 App 调度表，然后鼠标左键单击“Del”按钮将其删除。每个 App 调度表需要分配多少 App 调度表由用户根据应用程序的需求而定。

The figure above shows the App schedule table allocation options in the Application configuration interface. The App schedule allocation option shown in the figure above is enabled. Users can click the "Add" button to assign the App schedule table in the App Schedule list to the current application. If you need to delete the App schedule table that has been assigned to the current application, you need to select the schedule table that needs to be deleted by left-clicking in the App schedule table list at the bottom of the figure, and then click the Del button with the left mouse button to delete it. How many schedule table need to be allocated to each schedule table is determined by the user based on the needs of the application.

5.10.7 AppSpinlock allocation configuration for Application

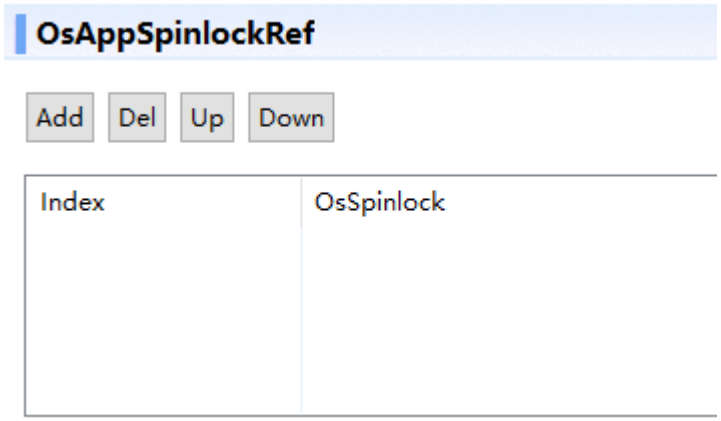


Figure5-57 AppSpinlock allocation option for Application

上图显示了 APP 自旋锁配置界面中的 APP 自旋锁分配选项。上图所示的 APP 自旋锁表分配选项被开启。用户可以鼠标点击“Add”按钮，将 APP 自旋锁列表中的 APP 自旋锁分配给当前应用程序。如果用户需要将已经分配给当前应用程序的 APP 自旋锁删除，则需要在图中下方的 APP 自旋锁列表通过鼠标左键单击的形式选中需要被删除的 APP 自旋锁，然后鼠标左键单击“Del”按钮将其删除。每个 APP 自旋锁需要分配多少 APP 自旋锁由用户根据应用程序的需求而定。

The figure above shows the APP spinlock assignment options in the APP spinlock configuration interface. The APP spinlock table allocation option shown in the figure above is enabled. The user can click the "Add" button to assign the APP spinlock in the APP spinlock list to the current application. If the user needs to delete the APP spinlock that has been assigned to the current application, you need to select the APP spinlock that needs to be deleted in the APP spinlock list at the bottom of the figure by left-clicking with the left mouse button, and then click the "Del" button to delete it. How many APP spinlocks need to be assigned to each APP spinlock is determined by the user according to the needs of the application.

5.10.8 Hook configuration items for Application

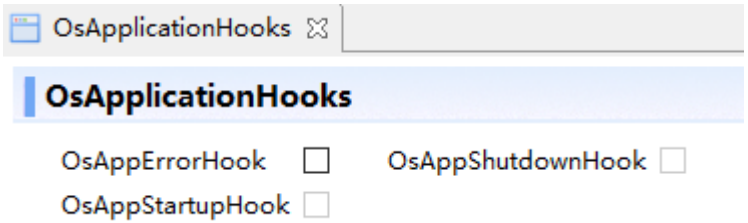


Figure5-47 Hook configuration parameters for Application

Figure above shows the configuration options for the Hook parameter for Application, and the configuration information is shown in table below.

Table5-22 Hook configuration parameter for Application information

Name	Type	Range	Default Value	Description
OsAppErrorHook	Boolean	False/True	False	选择系统是否使用 AppErrorHook 功能。可勾选： Select whether the system use the AppErrorHook function. Can be selected.
OsAppShutdownHook	Boolean	False/True	False	选择系统是否使用 AppShutdownHook 功能。可勾选： Select whether the system use the AppShutdownHook function. Can be selected.
OsAppStartupHook	Boolean	False/True	False	选择系统是否使用 AppStartupHook 功能。可勾选： Select whether the system use the AppStartupHook function. Can be selected.

5.11 Ioc configuration information

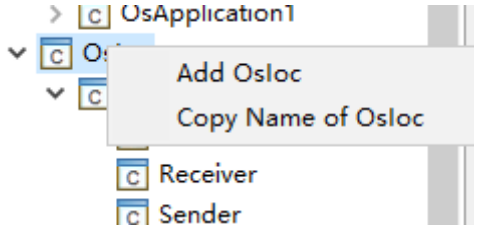


Figure5-48 Add Ioc

用户可以通过鼠标右键单击树形菜单“Os -> Ioc”，然后鼠标左键单击弹出的“Add Os Ioc”按钮在操作系统中应用程序。建立调度表时，需要在弹出的命名窗口中输入 Ioc 名称。之后，新建立的 Ioc 将作为树形结构的第三级菜单出现在第二级菜单“OsIoc”下面。用户可以通过鼠标右键单击“Os -> Ioc -> Ioc 名称”，然后在弹出的选项中通过鼠标左键单击“Del Ioc”或者是“Modify name of Ioc”对 Ioc 进行删除或者重命名。以上内容的图示信息请参考章节 5.2 中图 9—图 11。

The user can right-click the tree menu "Os-> Ioc " and then click the "Add Ioc " button popped up to set up the Ioc in the operating system. When the Ioc is set up, the Ioc name needs to be entered in the named window popped up.After that, the newly created Ioc will appear as the third level menu of the tree structure under the second menu " Ioc ".Users can through the right-click on the button of "Os - > Ioc - > Ioc name “to choose the Ioc, and then through the left-click of" Del Ioc” " or "Modify the name of Ioc " in the pop-up option to delete or rename the Ioc. Please refer to section 5.2 for detailed information above in Figure 9 - Figure 11.

5.11.1 Basic property configuration of Ioc

Figure5-49 Basic property configuration of Ioc

Figure above shows tow basic Ioc basic property configuration options in the Ioc configuration interface, as is shown in table below.

Table5-23 Basic attribute parameter information of Ioc

Name	Type	Range	Default Value	Description
IocQueue	Boolean	False/True	False	选择是否在消息中使用队列结构 Choose whether queue structure is used within the message.
OsIocBufferLength	Int	0 .. 4294967295	1	此属性定义要为查询通信分配的 IOC 内部队列的大小。 This attribute defines the size of the IOC internal queue to be allocated for a queued communication.

5.11.2 OsIocDataProperties configuration for Ioc

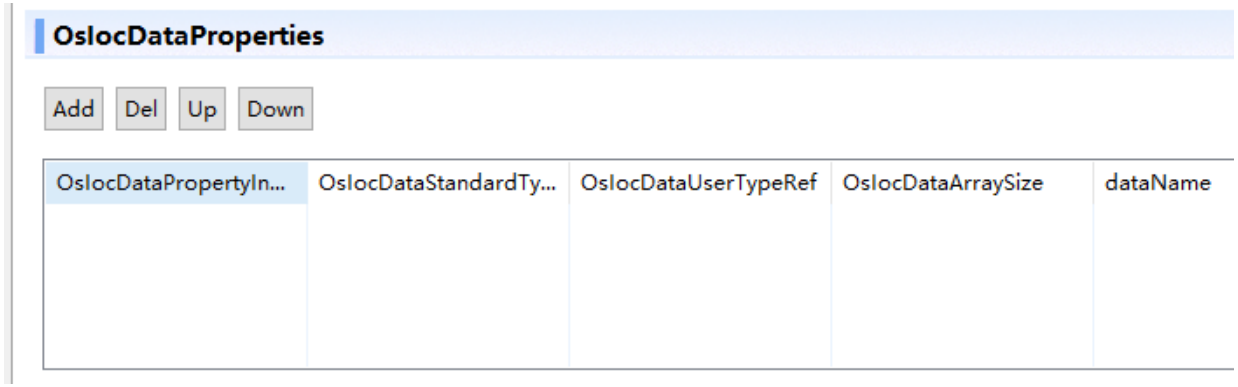


Figure5-50 OsIocDataProperties configuration for Ioc

上图显示了 Ioc 配置界面中数据优先级配置选项，在选择好各个数据类型的优先级和其他属性后，用户可以单击"Add"选项，将配置好的数据优先级应用到 Ioc。如果用户需要将当前 Ioc 的数据优先级配置删除，则需要在图下方的数据优先级配置列表中通过鼠标左键的形式选中要被删除的配置，然后鼠标单击“Del”按钮将其删除。每个 Ioc 需要配置多少中断优先级由用户根据 Ioc 的需求而定。

The figure above shows the data priority configuration options in the IOC configuration interface, after selecting the priority and other attributes of each data type, users can click the "Add" option to apply the configured data priority to the IOC. If you need to delete the data priority configuration of the current IOC, you need to select the configuration to be deleted by left-clicking in the data priority configuration list at the bottom of the figure, and then click the Del button to delete it. How much interrupt priority needs to be configured for each IOC depends on the needs of the user based on the needs of the IOC.

5.11.3 OsIocReceiverProperties configuration for Ioc

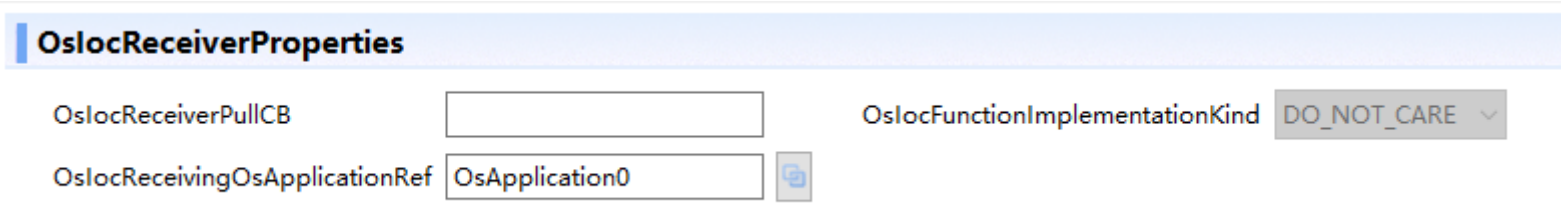


Figure5-51 OsIocDataProperties configuration for Ioc

Figure above shows the two OsIocReceiverProperties options in the Ioc configuration interface, and each configuration item information is shown in table below.

Table5-24 basic attribute parameter information of ISR

Name	Type	Range	Default Value	Description
OsIocReceiverPullCB	FunctionNameDef	-	-	此属性定义回调函数的名称，IOC 应在每次数据接收时调用接收核心。 This attribute defines the name of a callback function that the

				IOC shall call on the receiving core for each data reception
OsIocFunctionImplementationKind	Enum	DO_NOT_CARE FUNCTION MACRO	DO_NOT_CARE	此参数被用做选择此 communication 是被实现为宏还是一个函数。 This parameter is used to select whether this communication is implemented as a macro or as a function.
OsIocReceivingOsApplicationRef	Ref	-	-	此属性是对操作系统配置文件中定义的接收 OsApplication 实例的引用。 This attribute is a reference to the receiving OsApplication instance defined in the configuration file of the OS

5.11.4 OsIocSenderProperties configuration for Ioc

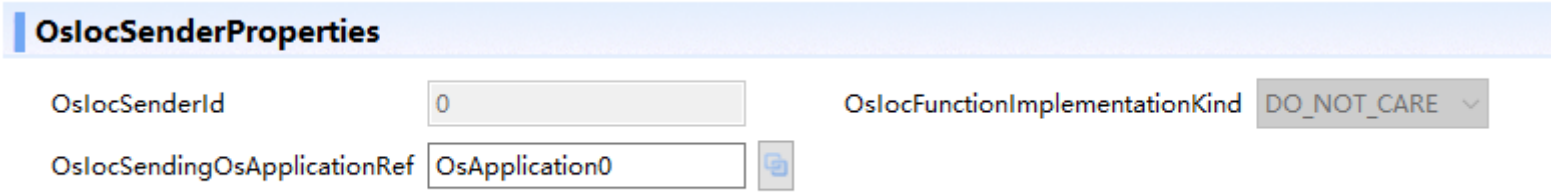


Figure5-52 OsIocSenderProperties configuration for Ioc

Figure above shows the two OsIocSenderProperties options in the Ioc configuration interface, and each configuration item information is shown in table below.

Table5-25 basic attribute parameter information of ISR

Name	Type	Range	Default Value	Description
OsIocSenderId	Int	1-255	1	此属性是对操作系统配置文件中定义的发送 A 实例的引用 This attribute is a reference to the sending OS-Application instance defined in the configuration file of the OS
OsIocFunctionImplementationKind	Enum	DO_NOT_CARE FUNCTION MACRO	DO_NOT_CARE	此参数被用做选择此 communication 是被实现为宏还是一个函数。 This parameter is used to select whether this communication is implemented as a macro or as a function.
OsIocSendingOsApplicationRef	Ref	-	-	此属性是对操作系统配置文件中定义的发送 OsApplication 实例的引用。 This attribute is a reference to the Sending OsApplication instance defined in the configuration file of the OS

5.12 Spinlock configuration information

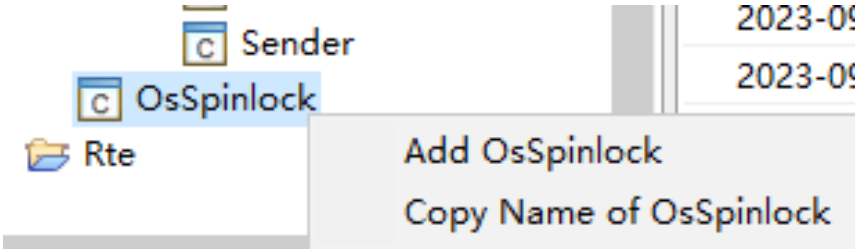


Figure5-53 Add Ioc

用户可以通过鼠标右键单击树形菜单“Os -> Spinlock”，然后鼠标左键单击弹出的“Add OsSpinlock”按钮在操作系统中应用程序。建立调度表时，需要在弹出的命名窗口中输入 Spinlock 名称。之后，新建立的 Spinlock 将作为树形结构的第三级菜单出现在第二级菜单“Os Spinlock”下面。用户可以通过鼠标右键单击“Os -> Spinlock -> Spinlock 名称”，然后在弹出的选项中通过鼠标左键单击“Del Spinlock”或者是“Modify name of Spinlock”对 Spinlock 进行删除或者重命名。以上内容的图示信息请参考章节 5.2 中图 9—图 11。

The user can right-click the tree menu "Os-> Spinlock " and then click the "Add Spinlock " button popped up to set up the Spinlock in the operating system. When the Spinlock is set up, the Spinlock name needs to be entered in the named window popped up.After that, the newly created Spinlock will appear as the third level menu of the tree structure under the second menu " Spinlock ".Users can through the right-click on the button of "Os - > Spinlock - > Spinlock name “to choose the Spinlock, and then through the left-click of" Del Spinlock”" or "Modify the name of Spinlock " in the pop-up option to delete or rename the Spinlock. Please refer to section 5.2 for detailed information above in Figure 9 - Figure 11.

5.12.1 Basic property configuration of Spinlock

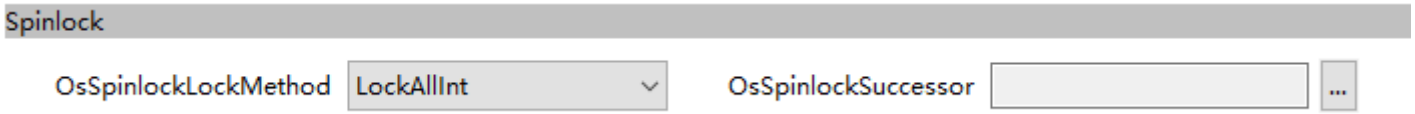


Figure5-54 Basic property configuration of Spinlock

Figure above shows tow basic Spinlock basic property configuration options in the Ioc configuration interface, as is shown in table below.

Table5-26 Basic attribute parameter information of Spinlock

Name	Type	Range	Default Value	Description
OsSpinlockLockMethod	Enum	LockALLInt LockOSInt None	LockALLInt	选择是否在消息中使用队列结构 Choose the spinlock method type.

		LockWithResScheduler		
OsSpinlockSuccessor	Ref	-	-	对有权访问此对象的 OsApplications 的引用。要检查是否可以（以嵌套方式）占用 spinlcok 而没有任何死锁的危险，可以定义一个自旋锁链表。旋转锁只能按链表的顺序占用。允许跳过旋转锁。如果未指定链表，则无法嵌套旋转锁。 Reference to OsApplications that have an access to this object. To check whether a spinlock can be occupied (in a nested way) without any danger of deadlock, a linked list of spinlocks can be defined. A spinlock can only be occupied in the order of the linked list. It is allowed to skip a spinlock. If no linked list is specified, spinlocks cannot be nested.

5.12.2 SpinlockAccessingApplication configuration

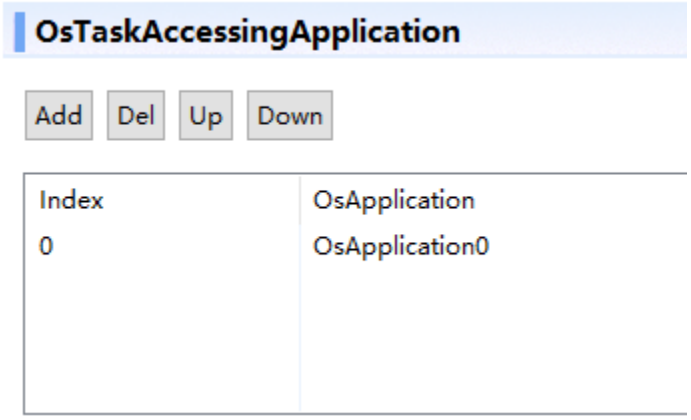


Figure5-55 SpinlockAccessingApplication configuration

上图显示了 Spinlock 配置界面中的 Application 访问选项。用户可以鼠标单击“Add”按钮，允许当前任务访问所选 Application。如果用户需要将 Spinlock 的 application 访问权限删除，则需要在图下方的列表中通过鼠标左键单击的形式选中此配置，然后鼠标左键单击“Del”按钮

将其删除。每个 Task 需要访问多少 Applications 由用户跟据 Task 的需求而定。

The figure above shows the Application access option in the Spinlock configuration interface. The user can click the "Add" button with the mouse to allow the current Spinlock to access the selected Application. If the user needs to delete the application access permission of the Spinlock, you need to select this configuration in the list below the figure by left-clicking with the left mouse button, click on the "Del" button is needed. How many Applications each Spinlock needs to access depends on the needs of the user and the Spinlock.

5.13 Error Information

Table5-27 Common Error List

Type	Error ID	Description
Error	EAS_Os_ERR000	OsIsrResourceLockBudget 没有配置。 The OsIsrResourceLockBudget is not configured.
Error	EAS_Os_ERR001	OsIsrResourceLockResourceRef 没有配置。 The OsIsrResourceLockResourceRef is not configured.
Error	EAS_Os_ERR002	OsTaskResourceLockBudget 没有配置。 The OsTaskResourceLockBudget is not configured.
Error	EAS_Os_ERR003	OsTaskResourceLockResourceRef 没有配置。 The OsTaskResourceLockResourceRef is not configured.
Error	EAS_Os_ERR004	EventMask 没有配置。 The property "EventMask" is not configured.
Error	EAS_Os_ERR005	预期的 SystemTickValue(us)通过当前配置的 TimerInputClockFrequency(MHz)和 SystemTimerPrescalerValue 无法生成。

		The expected SystemTickValue(us) cannot be generated. Through the current configuration values of the property TimerInputClockFrequency(MHz) and SystemTimerPrescalerValue.
Error	EAS_Os_ERR006	该任务有多个内部资源。 The Task has more than one INTERNAL resources.
Error	EAS_Os_ERR007	共享优先级 key，TaskMaxActivationTimes 的总和可能超过了最大值。总和不应大于 255。 Share priority key, the sum of their TaskMaxActivationTimes exceeds the maximum possible value. The sum should be no larger than 255.
Error	EAS_Os_ERR008	LinkedResource 没有配置。 The property LinkedResource is not configured.
Error	EAS_Os_ERR009	该资源没有被任何任务或中断使用。 The resource is not being used by any Task or ISR.
Error	EAS_Os_ERR0010	几个 resources 被链接成一个圆。 Several resources are linked as a circle.
Error	EAS_Os_ERR0011	OsAlarmAction 没有子节点。 The OsAlarmAction without children node.
Error	EAS_Os_ERR012	Event 没有被任何任务使用。 The event is not being used by any task.
Error	EAS_Os_ERR013	InterruptSource 没有配置。 The property InterruptSource is not configured.

Error	EAS_Os_ERR014	在 SC2 或者 SC4 时中断等级不能是最大。 The property Interrupt Level cannot be to max Level in SC2 or SC4. 暂不支持
Error	EAS_Os_ERR015	ISRFuncName 没有配置。 The property ISRFuncName is not configured.
Error	EAS_Os_ERR016	OsTrustedFunctionName 没有配置。 The property OsTrustedFunctionName is not configured.
Error	EAS_Os_ERR018	Isr 有内部资源。 The Isr of has INTERNAL resources.
Error	EAS_Os_ERR019	类别 1 的 ISR 的优先级比类别 2 的 ISR 的低。 Category 1 ISR has been configured a lower priority than Category 2 ISR.
Error	EAS_Os_ERR020	OsAlarmCounterRef 没有配置。 The property OsAlarmCounterRef is not configured.
Error	EAS_Os_ERR021	OsAlarmActivateTaskRef 没有配置。 The property OsAlarmActivateTaskRef is not configured
Error	EAS_Os_ERR022	OsAlarmSetEventTaskRef 没有配置。 The property OsAlarmSetEventTaskRef is not configured
Error	EAS_Os_ERR023	OsAlarmSetEventRef 没有配置。 The property OsAlarmSetEventRef is not configured.
Error	EAS_Os_ERR024	OsAlarmCallbackName 没有配置。 The property OsAlarmCallbackName is not configured.

Error	EAS_Os_ERR025	OsAlarmIncrementCounterRef 没有配置。 The property OsAlarmIncrementCounterRef is not configured.
Error	EAS_Os_ERR026	OsAlarmAppModeRef 没有配置。 The property OsAlarmAppModeRef is not configured.
Error	EAS_Os_ERR027	OsTaskAppModeRef 没有配置。 The property OsTaskAppModeRef is not configured.
Error	EAS_Os_ERR028	AlarmTime 的值必须比 CounterCycle 的值大，比 CounterMaxValue 值小。 The value AlarmTime must be greater than CounterCycle and less than CounterMaxValue. 暂不支持
Error	EAS_Os_ERR029	CycleTime 的值比 CounterMaxValue 的值大了。 The property CycleTime is larger than the CounterMaxValue. 暂不支持
Error	EAS_Os_ERR030	CycleTime 的值比 CounterCycle 值小了。 The property CycleTime is less than the CounterCycle. 暂不支持
Error	EAS_Os_ERR031	OsCounterMinCycle 的值比 OsCounterMaxAllowedValue 的值大了。 The property OsCounterMinCycle is greater than OsCounterMaxAllowedValue.

Error	EAS_Os_ERR032	当 ScalabilityClass 是 SC3,必须配置 Application。 When ScalabilityClass is SC3,must be application.
Error	EAS_Os_ERR033	未使用。 no use.
Error	EAS_Os_ERR034	OsAlarmCallbackName，ISRFuncName 重复。 The property OsAlarmCallbackName, ISRFuncName of is duplicative.
Error	EAS_Os_ERR035	EP 的数量不能为零。 The Ep number cannot be zero.
Error	EAS_Os_ERR036	OsScheduleTableCounterRef 没有配置。 The OsScheduleTableCounterRef is not configured.
Error	EAS_Os_ERR037	OsScheduleTableAppModeRef 没有配置。 The OsScheduleTableAppModeRef is not configured.
Error	EAS_Os_ERR038	FinalDelay 必须比 CounterMinCycle 大。 The property FinalDelay must larger than CounterMinCycle.
Error	EAS_Os_ERR039	Ep Action 的数量不能为零。 The Ep Action number cannot be zero.
Error	EAS_Os_ERR040	OsScheduleTableSetEventTaskRef 没有配置。 The property OsScheduleTableSetEventTaskRef is not configured.
Error	EAS_Os_ERR041	OsScheduleTableActivateTaskRef 没有配置。 The property OsScheduleTableActivateTaskRef is not configured.
Error	EAS_Os_ERR042	OsScheduleTableSetEventRef 没有配置。 The property OsScheduleTableSetEventRef is not configured.

Error	EAS_Os_ERR043	Offset 减去最近的 Ep Offset 小于 CounterMinCycle。 The property Offset minus last Ep Offset is less than the CounterMinCycle.
Error	EAS_Os_ERR044	Offset 减去最近的 Ep Offset 大于 CounterMaxValue。 The property Offset minus last Ep Offset is larger than the CounterMaxValue.
Error	EAS_Os_ERR045	Receiver 只能创建一个如果使能 IocQueue。 The number of Receiver must be 1 when IocQueue is true.
Error	EAS_Os_ERR046	当是 Core 只有一个，ScalabilityClass 是 SC1,必须没有 Application。 When the number of core is 1 and ScalabilityClass is SC1,must no application.
Error	EAS_Os_ERR047	必须是 8 的倍数。 Value must be multiples of 8.
Error	EAS_Os_ERR048	当 osTrusted 不勾选，所有的关联的 ISR 必须是 Category 2。 When osTrusted is false, all the reference isrs must be Category 2.
Error	EAS_Os_ERR049	有多种解释: (1)如果 Application 的数量不为零，那么 Object 必须与 Application 关联。 (2)Object（名字）没有配置 Application（名字） (1)The Object （OsAlarm,OsCounter,OsScheTable,OsResource,Task） must be reference by Application if the number of Application is not 0. (2)The " + Object.getEleName() + " without " + application.getEleName().

Error	EAS_Os_ERR050	Section 的名字不能为空。 Section name must not be null.
Error	EAS_Os_ERR051	Section 的名字在不同的 MemoryProtection 重复了。 Section name repeat in difference MemoryProtection.
Error	EAS_Os_ERR052	OsTrustedFunctionParam 没有参数。 OsTrustedFunctionParam is not parameter.
Error	EAS_Os_ERR053	OsIocDataProperties 没有参数。 OsTrustedFunctionParam is not parameter.
Error	EAS_Os_ERR054	在 IocCommunication 下没有 OsIocSenderPropertie and at least one Sender and one Rx. No OsIocSenderProperties in OsIocCommunication. 至少有一个 Sender 和一个 Rx.
Error	EAS_Os_ERR055	当有多个 Sender 在同一个 Ioc 下，在每个 Sender 中只允许有一个 Data。 When there are multiple Sender in one Ioc, only allowed one Data in each Sender.
Error	EAS_Os_ERR056	OsIocReceivingOsApplicationRef 没有配置。 The property OsIocReceivingOsApplicationRef is not configured.
Error	EAS_Os_ERR057	OsIocSendingOsApplicationRef 没有配置。 The property OsIocSendingOsApplicationRef is not configured.
Error	EAS_Os_ERR058	Trusted Function 的名字重复了。 The property Trusted Function Name is duplicate.
Error	EAS_Os_ERR059	任何两个模块的处于 DataList 的 DataName 不能重复。 The DataName of any two item in DataList cannot be repeated. TrustFun 的 Param，可以为空，但 name 不能重复。

Error	EAS_Os_ERR060	Sender 和 Receiver 关联的 Application 不能是同一个。 The Application referenced in the Sender and Receiver can't be the same.
Error	EAS_Os_ERR061	同一个 Task 的 Event Mask 不能相同。 The Event Mask in the same Task can't be the same.
Error	EAS_Os_ERR062	OsCounterTicksPerBase 比 OsCounterMaxAllowedValue 大。 The property OsCounterTicksPerBase is greater than OsCounterMaxAllowedValue.
Error	EAS_Os_ERR063	OsScheduleTableStartValue 比 OsCounterMaxAllowedValue 大。 The property OsScheduleTableStartValue is greater than OsCounterMaxAllowedValue.
Error	EAS_Os_ERR064	同一 Core 的 Isr，不能配置相同的中断源。 The ISR of same Core cannot be configured with the same InterruptSource
Error	EAS_Os_ERR065	Application 应该在多核时配置 The Application should configured when multi-cores.
Error	EAS_Os_ERR066	任务应该配置 Core。 Task should configurated Core .
Error	EAS_Os_ERR067	OsTaskInit 是非法名字，请输入其他名字。 OsTaskInit is invalid name, please input another name.
Error	EAS_Os_ERR068	Core 没有配置。 Core is not configured.
Error	EAS_Os_ERR069	OsSpinLockSuccessor 配置错误，不能成圈。 OsSpinLockSuccessor is incorrectly configured ,Not into a circle.
Error	EAS_Os_ERR070	所有 Ep 名字相同。

		All the Ep node name should not be same.
Error	EAS_Os_ERR071	所有 callback/hook 功能名字相同。 All the callback and hook function name should not be same.
Error	EAS_Os_ERR072	如果“OsIocDataStandardTypeRef”设置为“UserDefine”，则需要用户手工填写“OsIocDataUserTypeRef”中的数据类型名称。 If OsIocDataStandardTypeRef is set to UserDefine, the user needs to manually fill in the data type name in the OsIocDataUserTypeRef box.
Error	EAS_Os_ERR073	用户需要在“OsIocDataName”框中手动填写数据名称。 The user needs to manually fill in the data name in the OsIocDataName box.
Error	EAS_Os_ERR074	一个任务的资源只能被锁定一次。 Resources of a task can only be locked once.
Error	EAS_Os_ERR075	一个 ISR 的资源只能锁定一次。 Resources of a ISR can only be locked once.
Error	EAS_Os_ERR076	systemConformanceClass 是 OS_ECC1 或者 OS_ECC2，但没有扩展任务。 systemConformanceClass is OS_ECC1 or OS_ECC2, but no Extended Task.
Error	EAS_Os_ERR077	当 ScalabilityClass 是 SC3 或者 SC4，但 application 没有 ApplicationMemoryProtection。 When ScalabilityClass is SC3 or SC4 ,but application no ApplicationMemoryProtection.
Error	EAS_Os_ERR078	当有 StandardType 时，必须设置 InitValue。 InitValue must be set when there is StandardType .
Error	EAS_Os_ERR079	IsrFuncname 不能和 IsrModel 的命名一样

		IsrFuncname can't be the same as IsrModel's name.
Error	EAS_Os_ERR080	
Error	EAS_Os_ERR081	
Error	EAS_Os_ERR082	当 ScalabilityClass 是 SC2 或者 SC4 ,必须有 TimingProtection。 When ScalabilityClass is SC2 or SC4,must has TimingProtection.
Error	EAS_Os_ERR083	当 Task 是扩展的，OsTaskEventRef 必须至少配置一个。 when Task is Extended, OsTaskEventRef must configurated at last one.
Error	EAS_Os_ERR084	Application 的数量不能超过 32 个。 The number of Application cannot more than 32.
Error	EAS_Os_ERR085	如果是一个 trusted application 请配置 ApplicationMemoryProtection。 Please configurate ApplicationMemoryProtection if it not a trusted application.
Error	EAS_Os_ERR086	如果选择了 IocQueue，OsIocBuffer 应该被配置大于 1。 If IocQueue is true, OsIocBuffer should be configurated more than 1.
Error	EAS_Os_ERR087	如果是 SC3 或者 SC4，Memory Protection 的 SectionName 应该不同。 If SC3 or SC4,SectionName of Memory Protection should be different.
Error	EAS_Os_ERR088	EventMask 不能被选择，一个 task 被分配的 events 数量不能超过 32。 The EventMask cannot be solved, the number of events associated with a task cannot more than 32.

Error	EAS_Os_ERR089	PrivateDataSection 和 PrivateCodeSection 不能和 MemmapSection 相同。 PrivateDataSection and PrivateCodeSection cannot share the same MemmapSection.

6 Examples

6.1 Create an Autostart 10ms cycle Task

Step 1: Create a Task

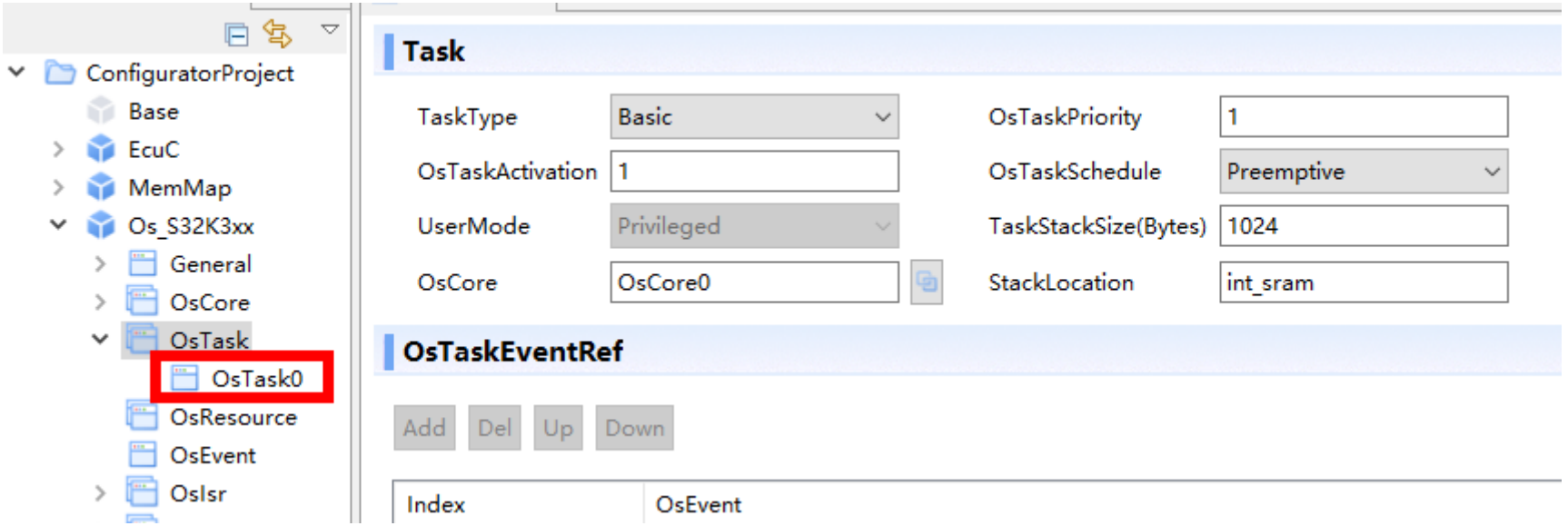


Figure6-1 Create a Task

Step 2: Create an Alarm

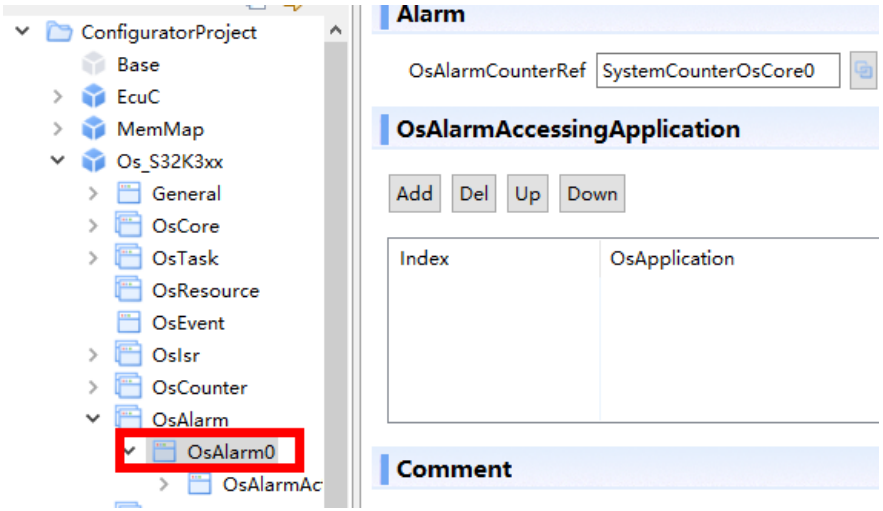


Figure6-2 Create an Alarm

Step 3: Set the AlarmAction to Activate the Task

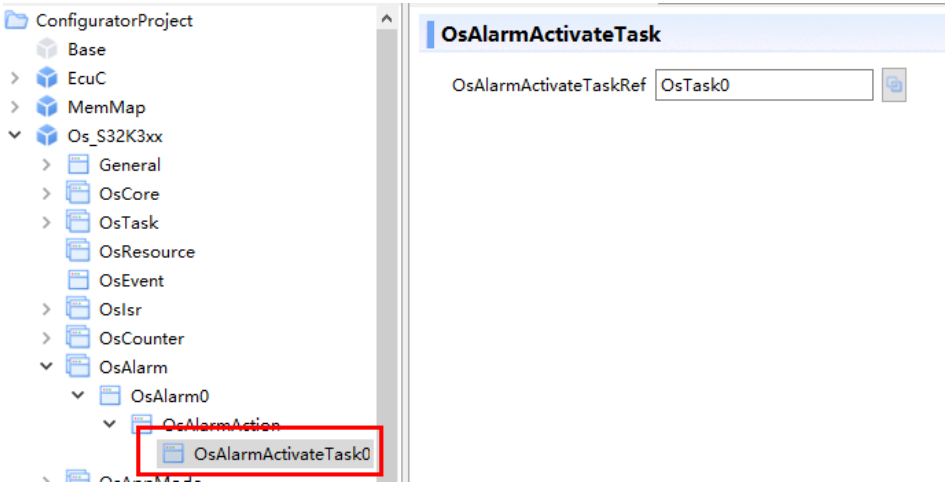


Figure6-3 Set the Alarm

Figure6-4

Step 4: Set the Alarm Autostart and cycletime (When systemcounter is 160MHz, 10ms = 1,600,000)

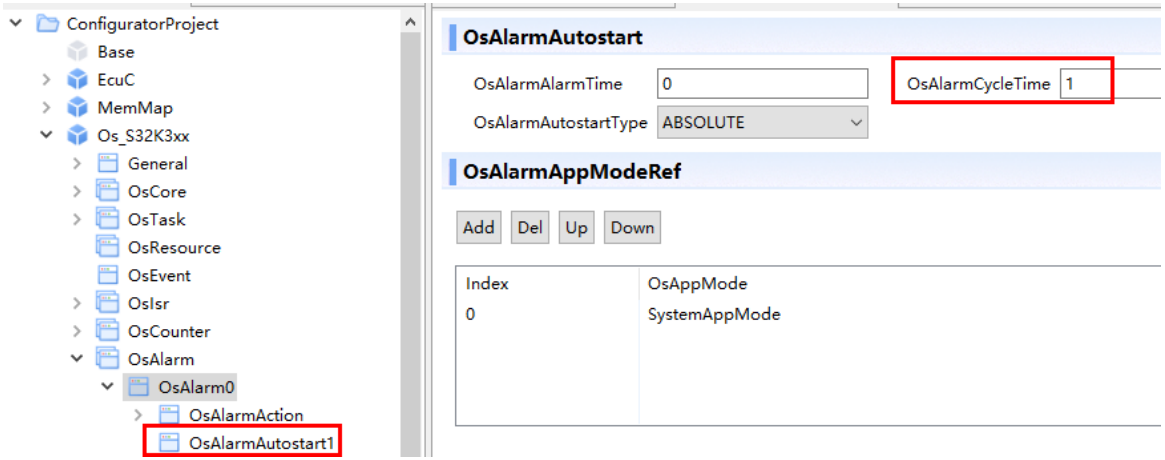


Figure6-5 Set the Alarm

6.2 How to use Extended Task

Step 1: Create an Extended Task

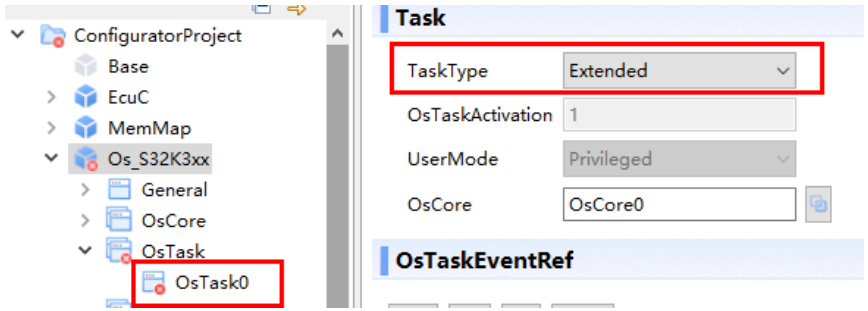


Figure6-6 Create an Extended Task

Step 2: Create an Event

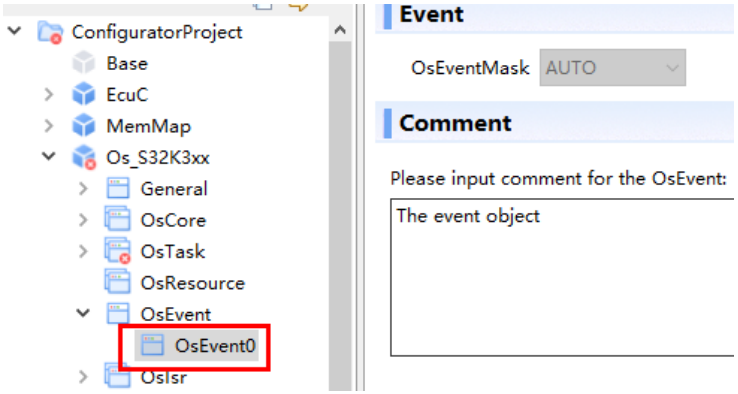


Figure6-7 Create an Event

Step 3: Associate the Task and Event

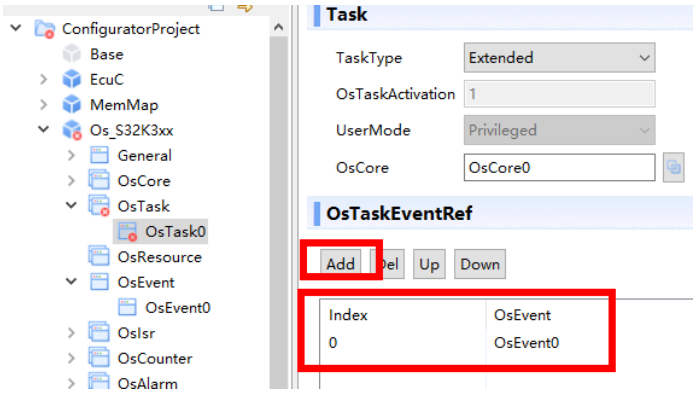


Figure6-8 Associate the Task and Event

Step 4: Add User Code

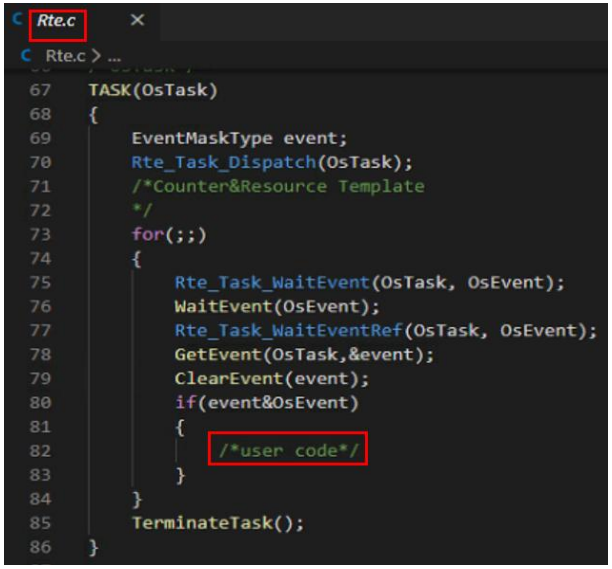


Figure6-9 Add User Code

When User Set an Event, the OS will Run the User Code.

6.3 How to use IOC

6.3.1 How to config no IocQueue

Step 1: Select OsScalabilityClass to SC3 or SC4

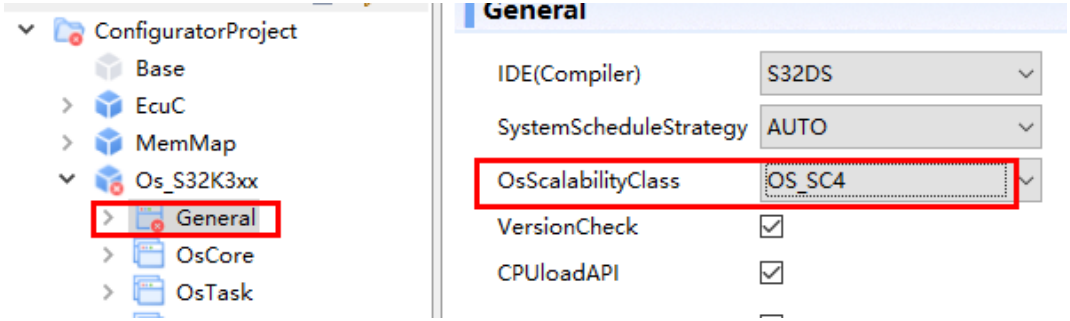


Figure6-10 Select SC

Step 2: Add two Os-Application at least

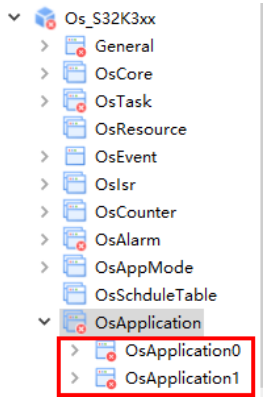


Figure6-11 Add OS-Application

Step 3: Add IOC

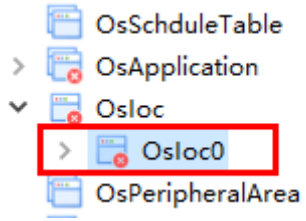


Figure6-12 Add IOC

Step 4: Select Receiver and Sender

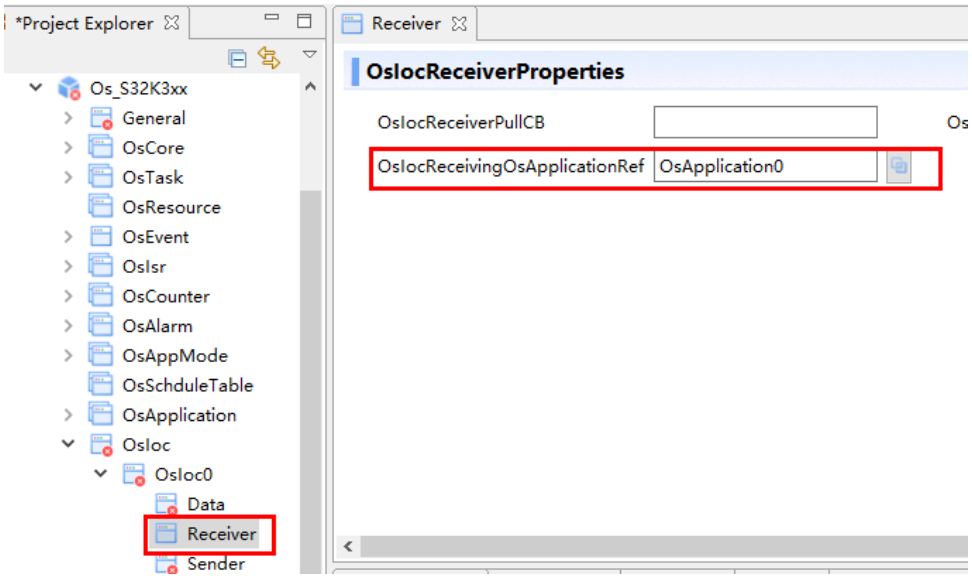


Figure6-13 Select Receiver

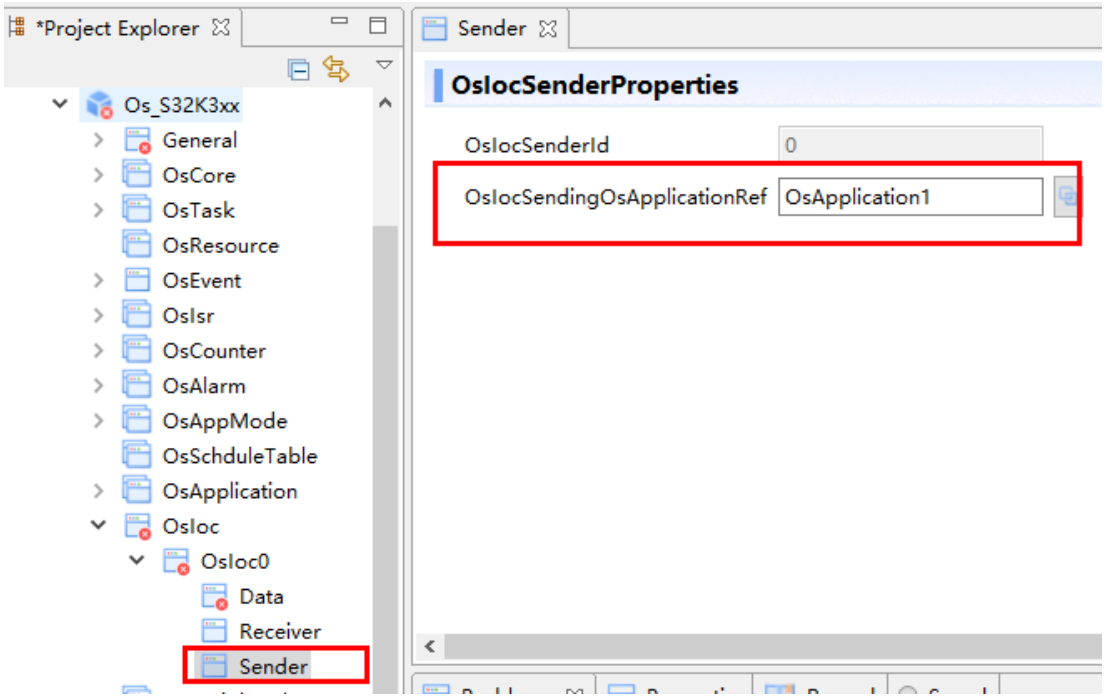


Figure6-14 Select Sender

Step 5: Add data

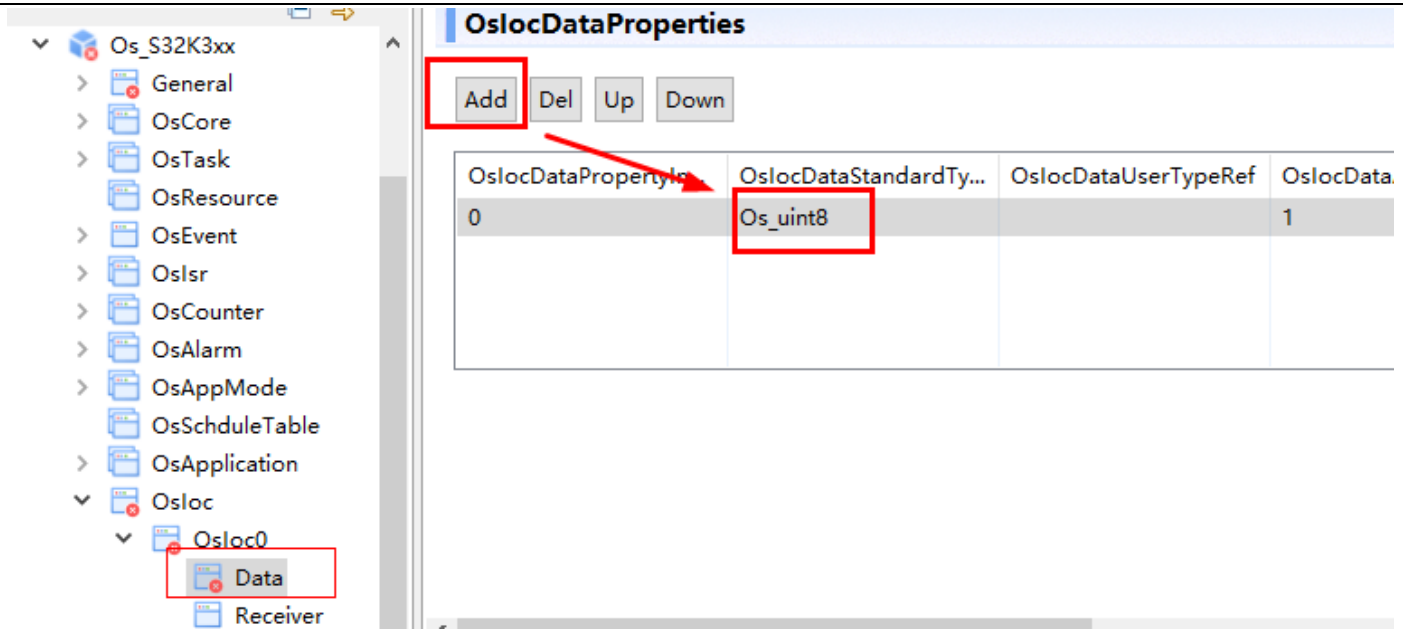


Figure6-15 Add data

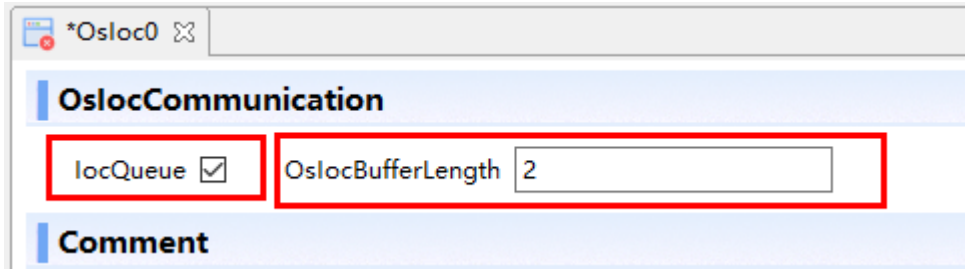
Step 6: Generate Code

Then user can use IocWrite_IocId0_Sender0() and IocRead_IocId0_Receiver0() API in Os_GenIoc.c

6.3.2 How to config IocQueue and OsIocDataArraySize

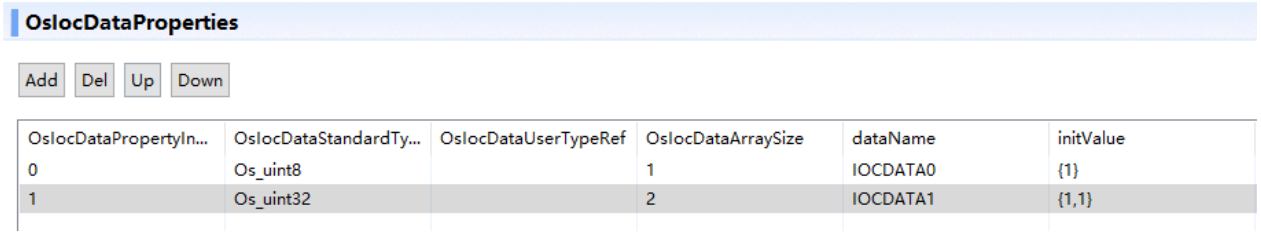
Step 1: Select IocQueue and OsIocBufferLength:

OsIocBufferLength represent how many times data can be sent(config in OsIocDataProperties) at one time, please refer to step3.



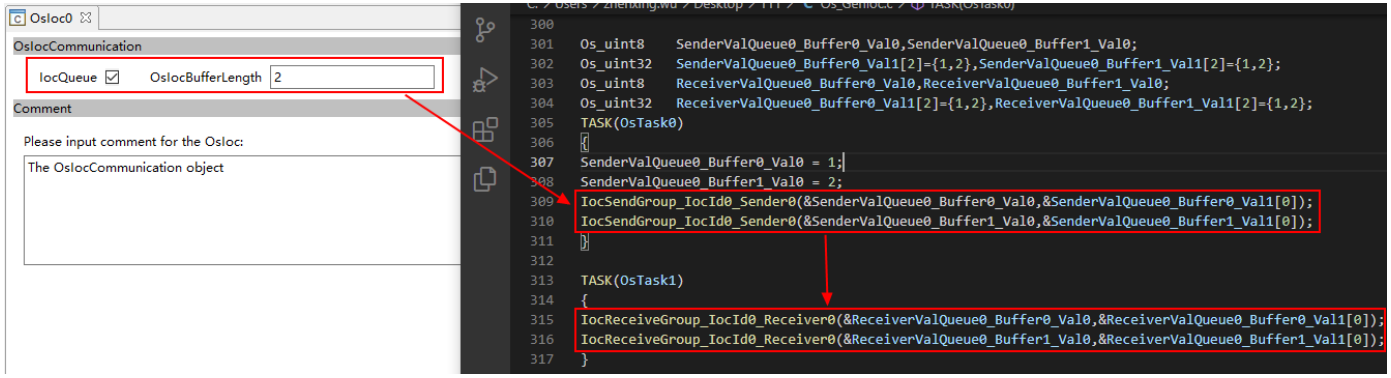
Step 2: Add Data:

OsIocDataArraySize represent array elements,

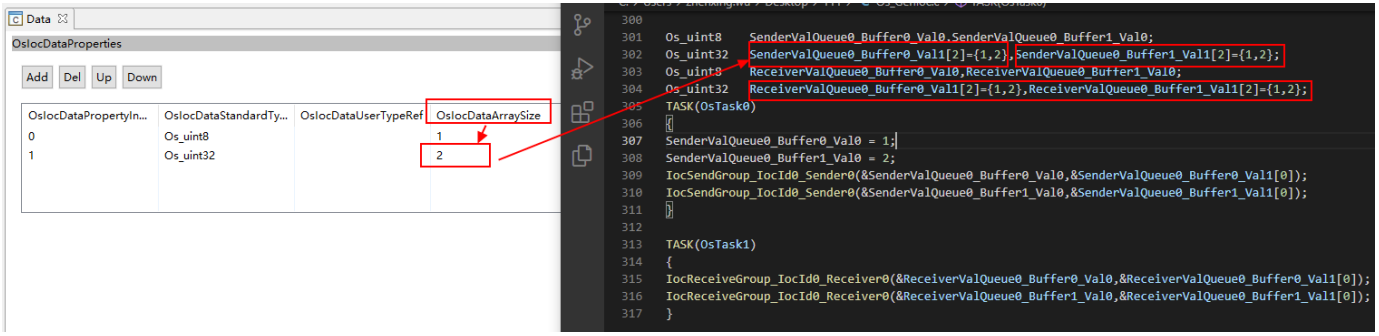


Step 3: Add Data in code

As follows, when configuring OsIocBufferLength as 2, you can call Ioc_Sender twice at one time, and at receiving end, it must be also received twice.

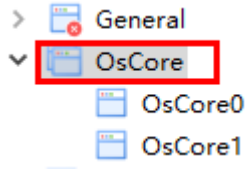


As follows, when configuring OsIocDataArraySize as 2, it represents the transmitted parameter is an array, and the elements of the array are 2.

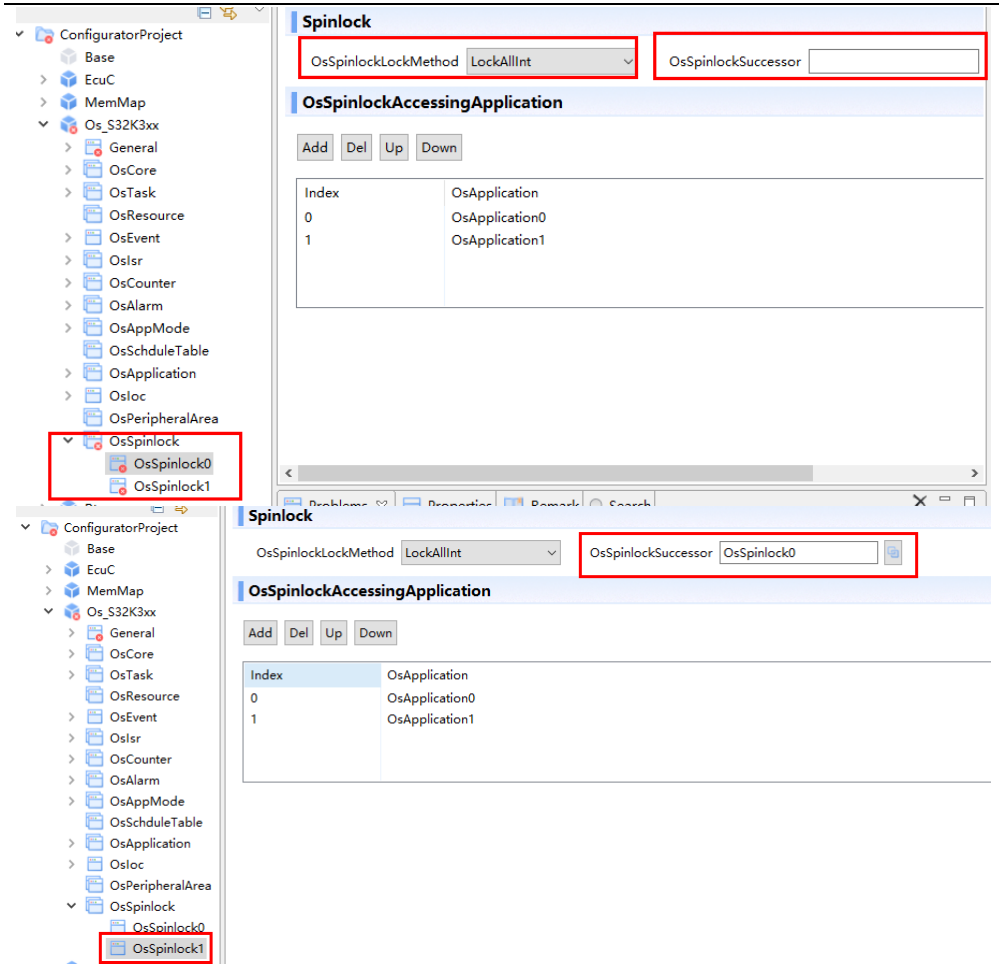


6.4 How to use Spinlock

Step 1: Add Multicore



Step 2: Config Spinlock



Step 3: Add Data in code

As follows, OsSpinlock1 configures OsSpinlockSuccessor as OsSpinlock0, so must acquire OsSpinlock0 first, and release spinlock follows lifo policy.

```
/*Low task priority*/
TASK(OsTask0_Core0)
{
GetSpinlock(OsSpinlock0);/*OsSpinlock1 successor is OsSpinlock0,must get OsSpinlock0 first*/
GetSpinlock(OsSpinlock1);
ActivateTask(OsTask1);
ReleaseSpinlock(OsSpinlock1);/**release spinlock follows lifo*/
ReleaseSpinlock(OsSpinlock0);
}
/*High task priority*/
TASK(OsTask1_Core1)
{
/*OsTask1 cannot get OsSpinlock0*/
GetSpinlock(OsSpinlock0);
}
```

6.5 Config an auto starts Schedule Table

Step 1: Add a Schedule Table

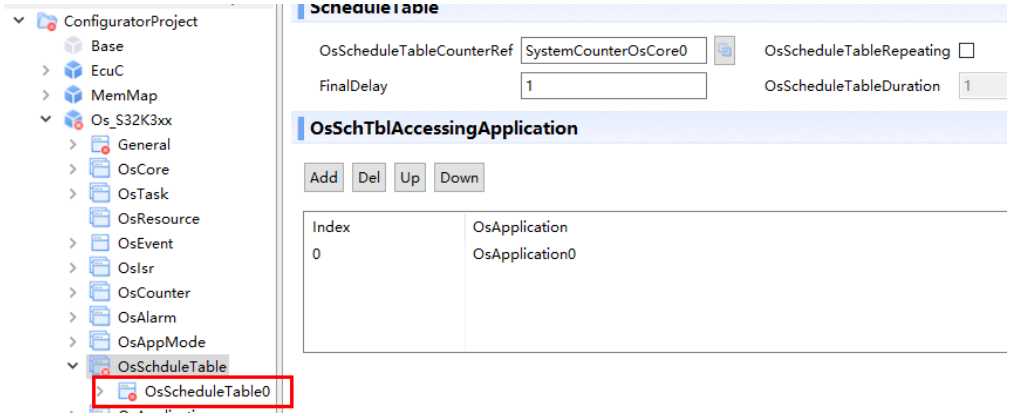


Figure6-16 Add Schedule Table

Step 2: Add three EP

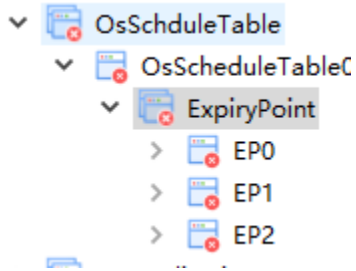


Figure6-17 Add EP

Step 3: Add three Task

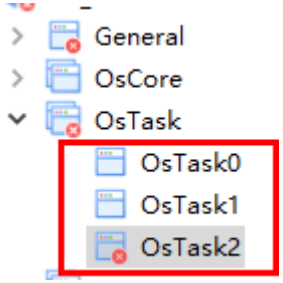


Figure6-18 Add Task

Step 4: Set every EP Activate a Task

Set EP0 Activate Task0, Set EP1 Activate Task1, Set EP2 Activate Task2

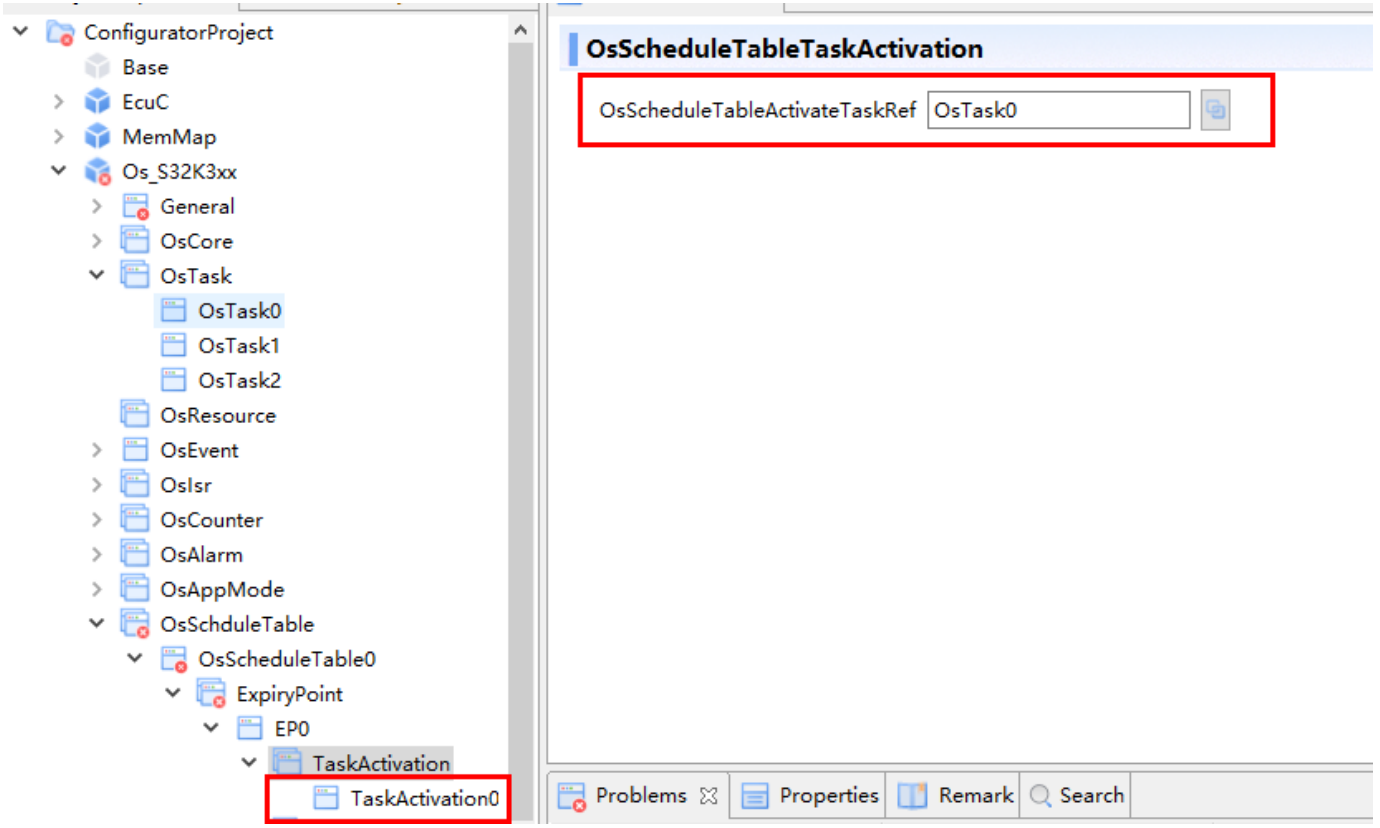


Figure6-19 Activate Task

Step 5: Set EP Offset

Set EP0 Offset 1000 EP1 Offset 2000 EP2 Offset 3000.

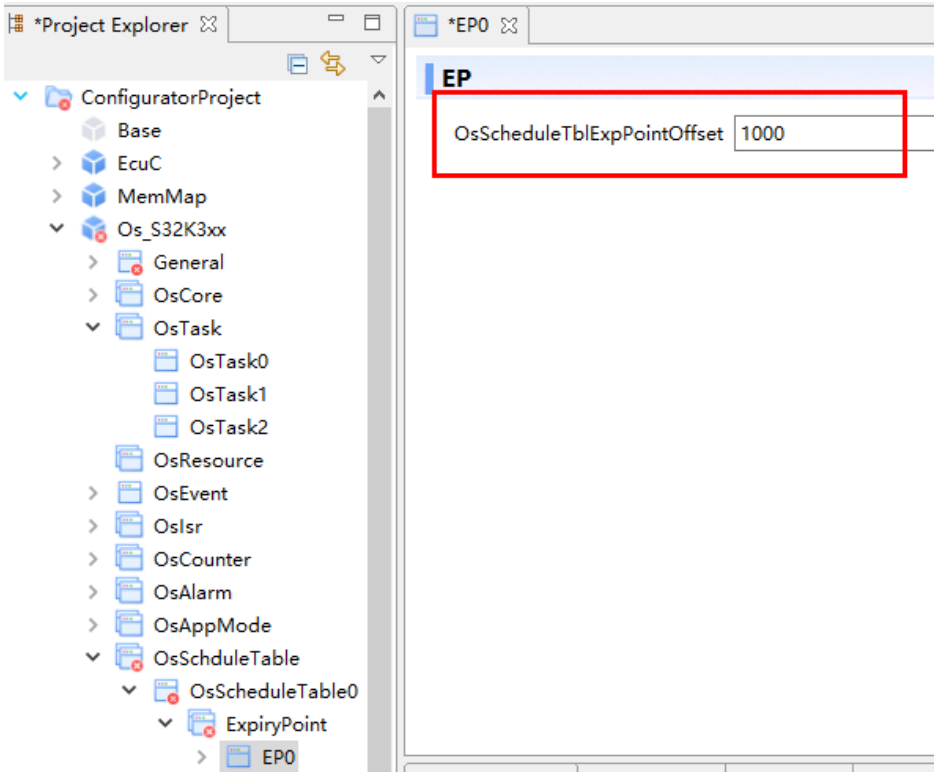


Figure6-20 Set Offset

Step 6: Set the Schedule Table Auto Start

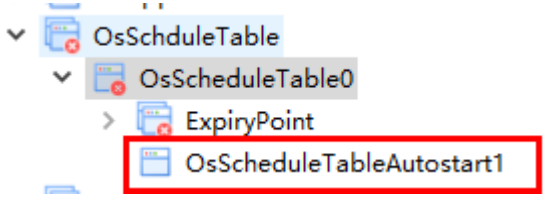


Figure6-21 Set Auto Start

Then StartOS, Task0 Task1 Task2 will run at 1000ms,200ms,300ms.

6.6 Protect a Task Execution Time

Config Timing Protection, Protect a Task Execution Time 10ms.

Step 1: Select OsScalabilityClass SC2 or SC4

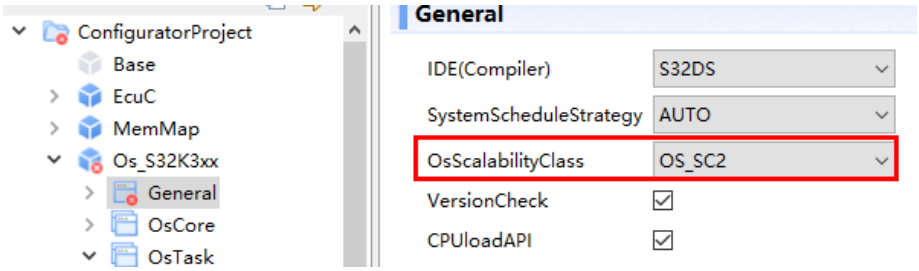


Figure6-22 Select SC

Step 2: Add Timing Protection

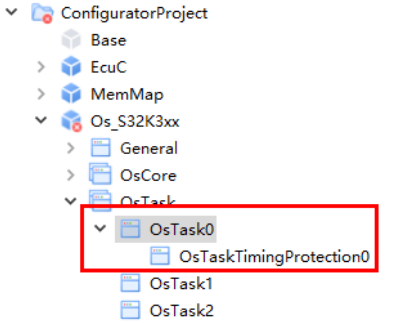


Figure6-23 Add Timing Protection

Step 3: Set OsTaskExecutionBudget 1000

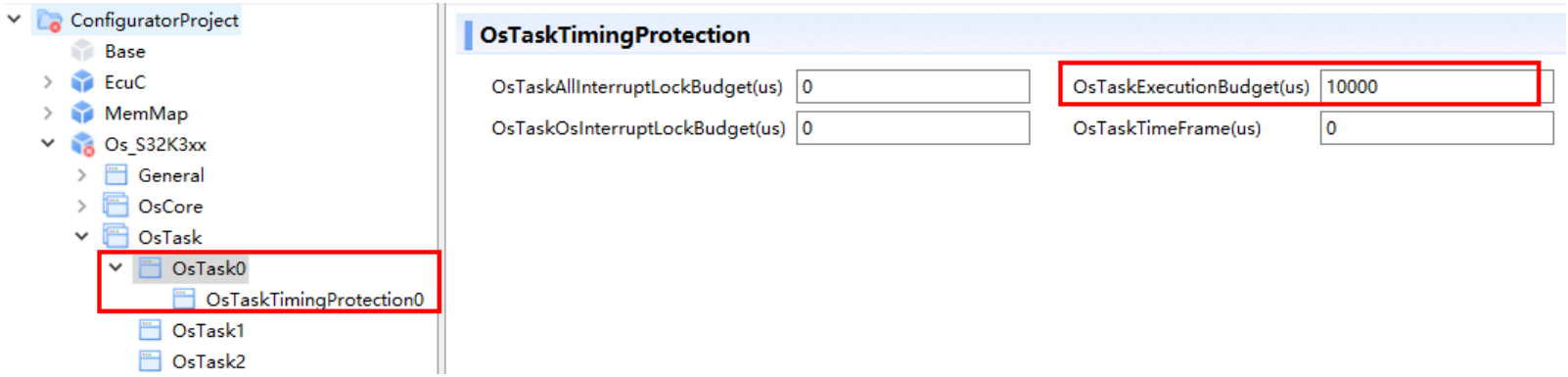


Figure6-24 Set ExecutionBudget

Timing protection errors are then triggered if the Task0 is performed longer than 10 ms.

6.7 How to get Cpuload

Step 1: Enable CPUloadAPI

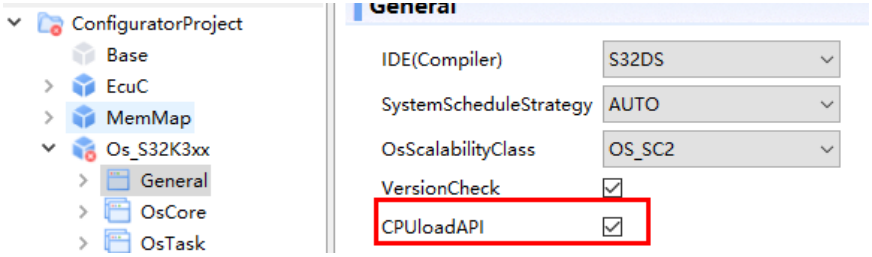


Figure6-25 Enable CPUloadAPI

Step 2: Set Basetime

It means: how often to check it out. (Unit:ms)

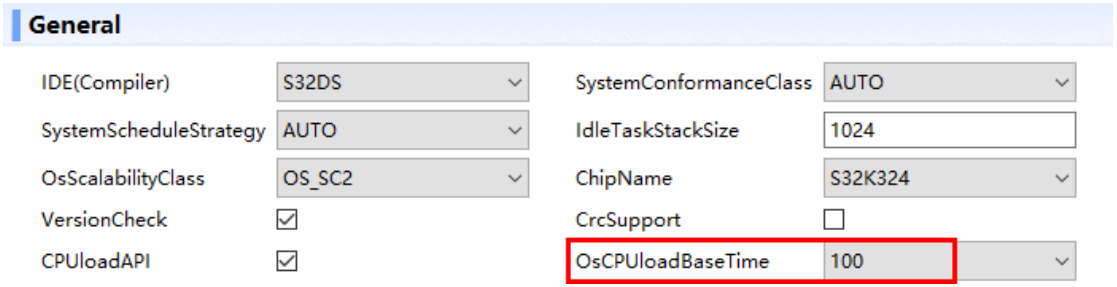


Figure6-26 Set OsCPUloadBase

Step 3: Add cycle task According to OsCPUloadBaseTime

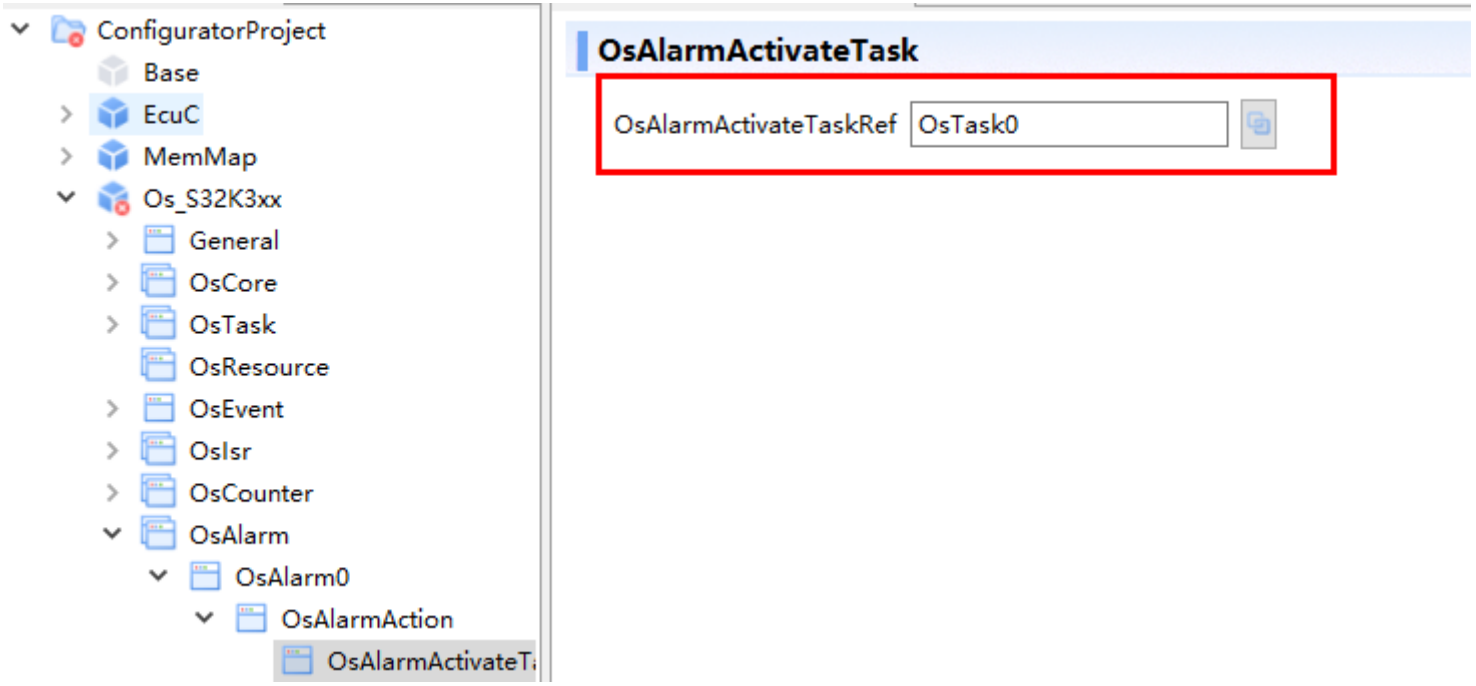


Figure6-27 Activate a Task

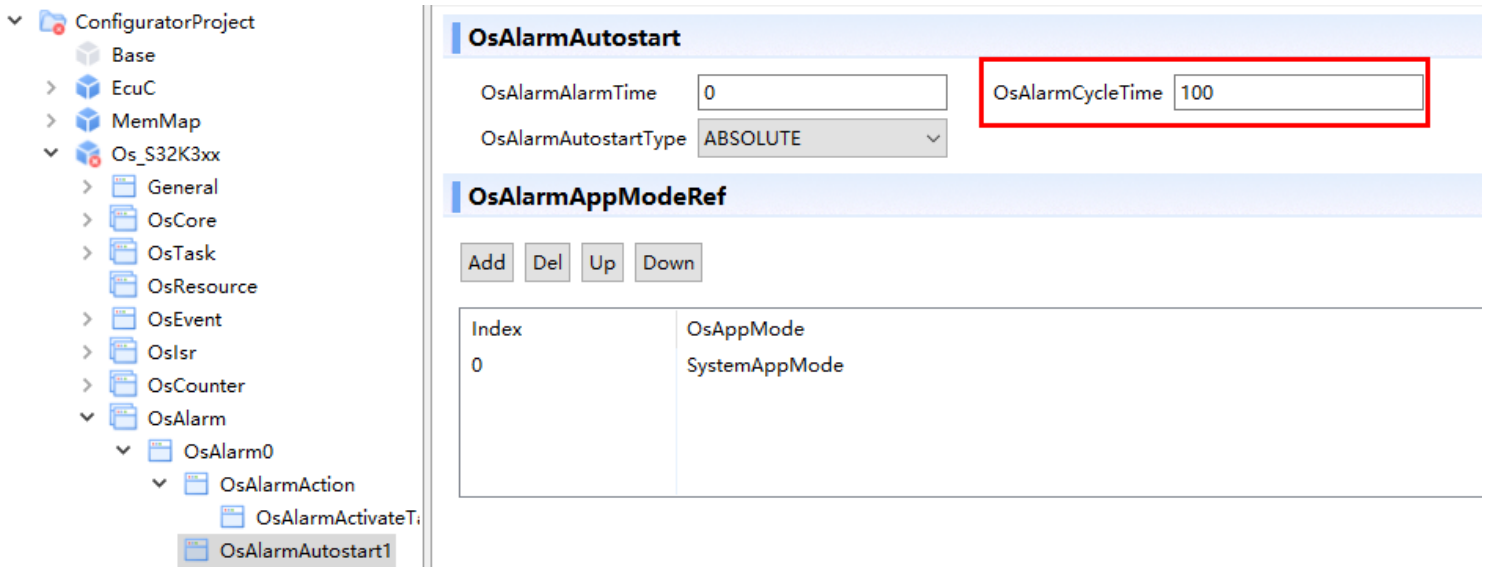


Figure6-28 Set Auto Start

Step 4: Add the API in cycle Task

```
62  /*************************************
63  *   Os Task
64  *************************************
65  Os_uint8 testValue;
66  /*OsTask0*/
67  TASK(OsTask0)
68  {
69      Rte_Task_Dispatch(OsTask0);
70      /*Counter&Resource Template
71      */
72      GetCpuLoadValue(&testValue)
73      TerminateTask();
74  }
75
```

Figure6-29 Demo Code

Add global variable test, call GetCpuLoadValue() every 100ms, then get the value of testValue, which represents the load rate. For example, testValue is 10, it means the load rate is 10%.

6.8 How to use TrustedFunction

Step 1: Add TrustedFunction

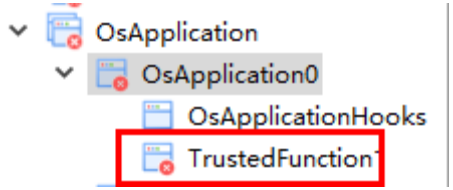
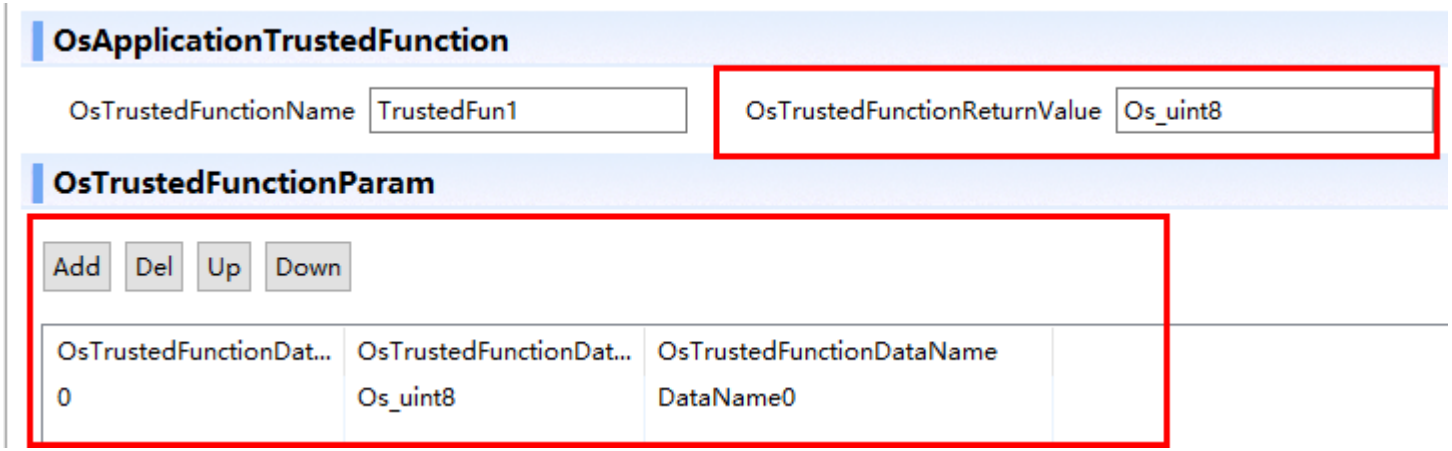


Figure6-30 Add TrustedFunction

Step 2: Configure TrustedFunctionParam



Step 3: Use CallTrustedFunction interface and add user code in TrustedFun handler

```
Os_uint8  test_a;
TASK(Task0)
{
    CallTrustedFunction(0,&test_a);
    TerminateTask();
}
```

Os_GenHook.c 2

Os_GenHook.c > ...

```
56
57
58  FUNC(Os_uint8, OS_CALLOUT_CODE)TrustedFun1(Os_uint8 DataName0)
59  {
60      /*user code*/
61  }
```

Figure6-31 add user code in TrustedFun handler

6.9 How to use Memory Protection

选择 SC3/SC4 后，可以配置任务或二类中断和 application 的内存保护段，其中 application 的内存保护数据段是必须配置的。

Step 1: Select OsScalabilityClass OS_SC3

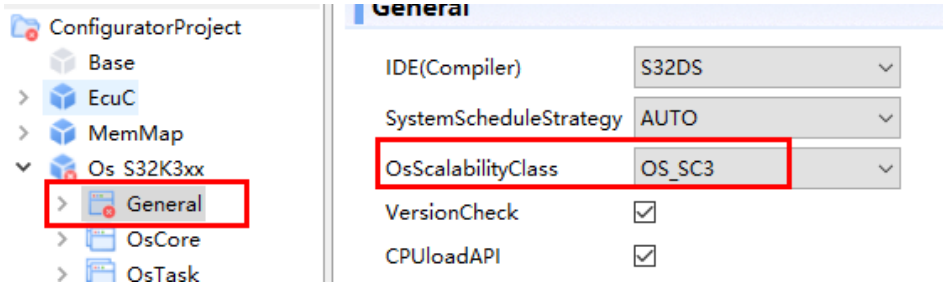


Figure6-32 select OS_SC3 in General

Step 2: Add an OS-Application

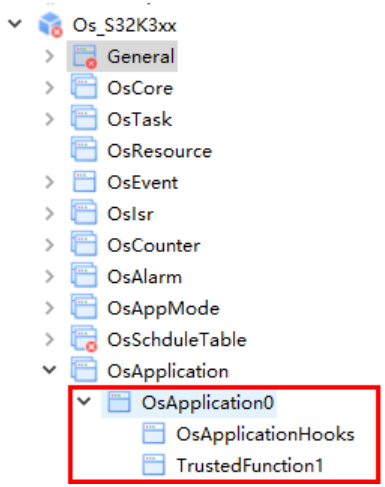


Figure6-33 add an OS-Application

Step 3: Select OsTrusted in OS-Application

在 OS-Application 中可以选择该应用集下所属的任务或二类中断为可信还是不可信，如果是不可信的，则任务和二类中断只可以访问自己所配置的和所处应用集的 Section 段，以及自己的栈空间；如果应用集为可信的，则对其访问区域不做限制。

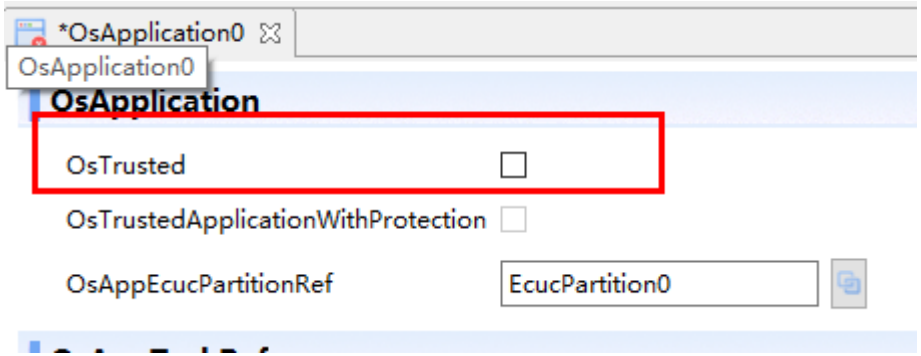


Figure6-34 select OsTrusted in OsApplication

Step 4: Add Memory Protection for the OS-Application

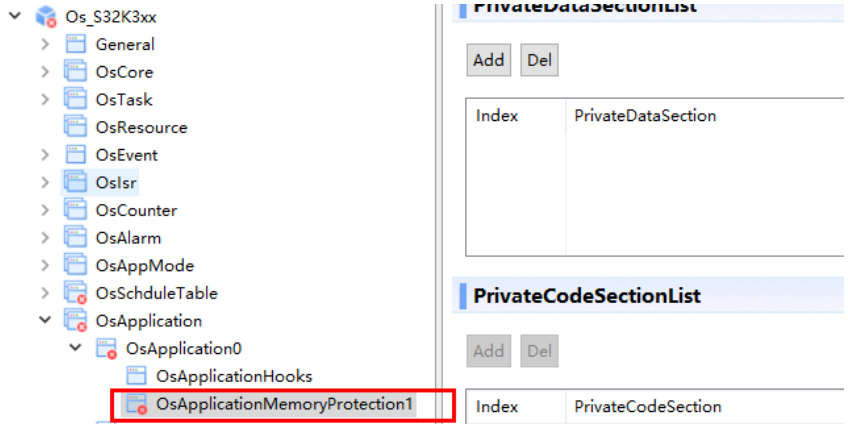


Figure6-35 add OsApplicationMemoryProtection

Step 5: Choose Data Section for the OS-Application from Memmap Module

OS 中所使用的内存保护段均为从下拉列表中选择，列表中的段为 Memmap Module 中配置的内存段，Memmap Module 配置内存段的方式在 Step A 中说明。

任务(Task)或二类中断(ISR)的 DataSection 段可以选配，但是 OS-Application 的 DataSection 在 SC3/SC4 工况下是至少配置一个的(可以配置多个 DataSection)。

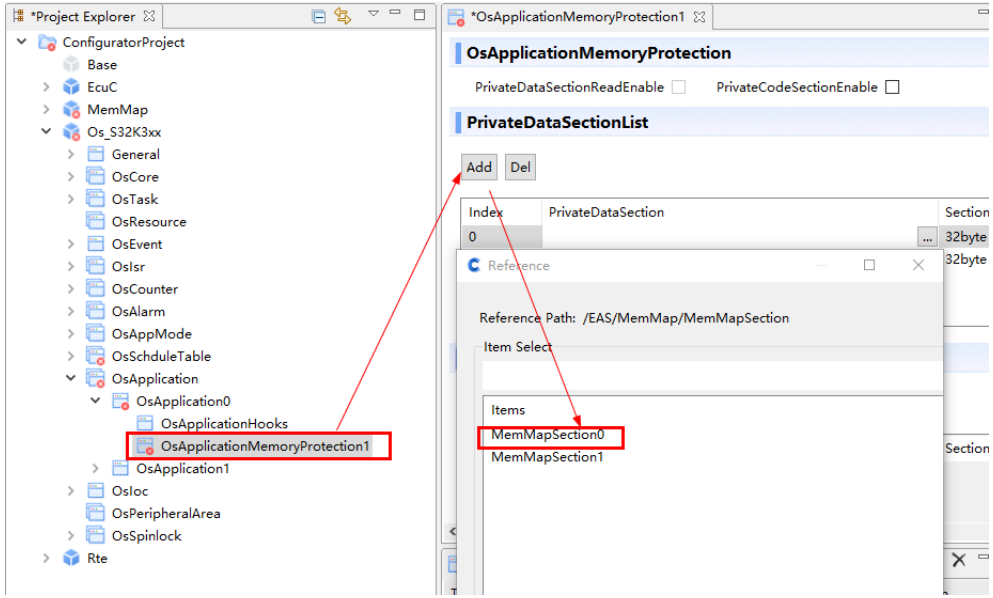


Figure6-36 choose data section for OS-Application

Step 6: Choose Code Section for the OS-Application from Memmap Module

若需要代码保护功能，则需使能 CodeSectionEnable，然后配置 CodeSection(配置方法同 Step5)

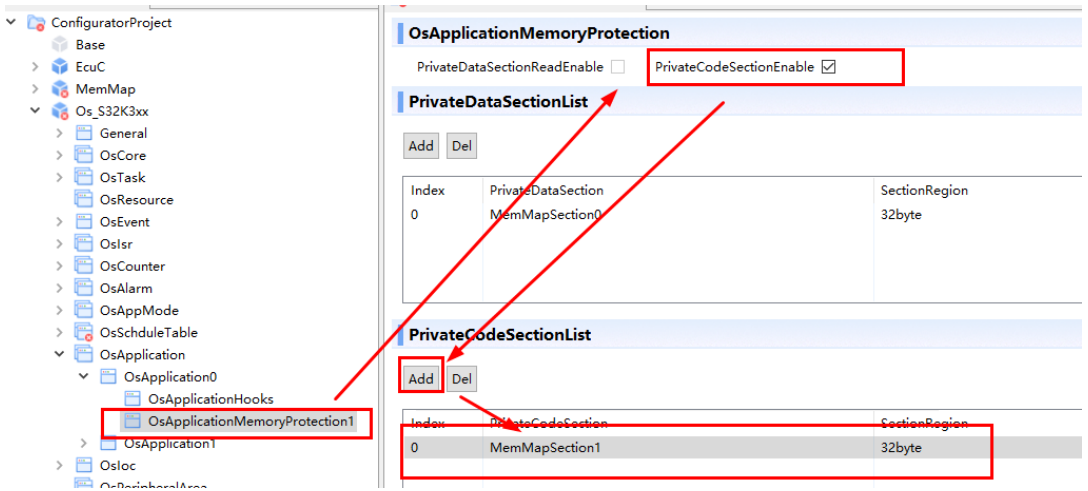


Figure6-37 choose code section for OS-Application

Step 7: Add a Task in the OS-Application

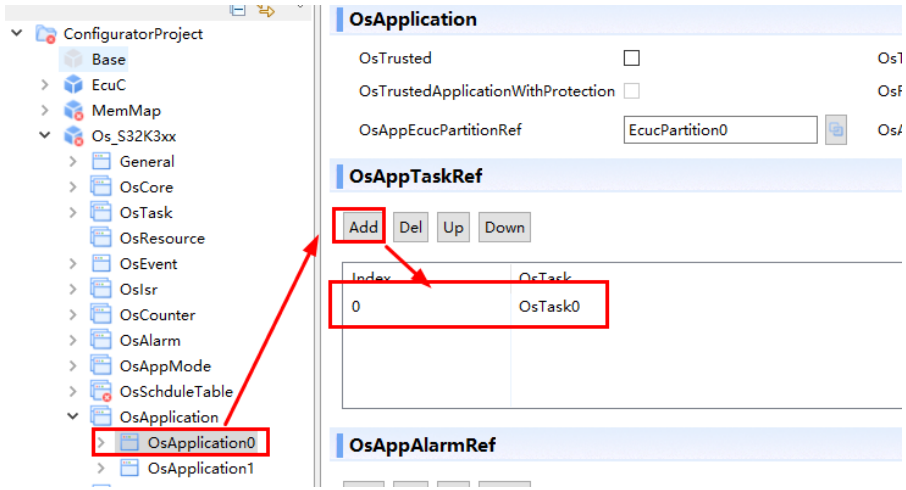


Figure6-38 add task for OS-Application

Step 8: Add Memory Protection for the Task

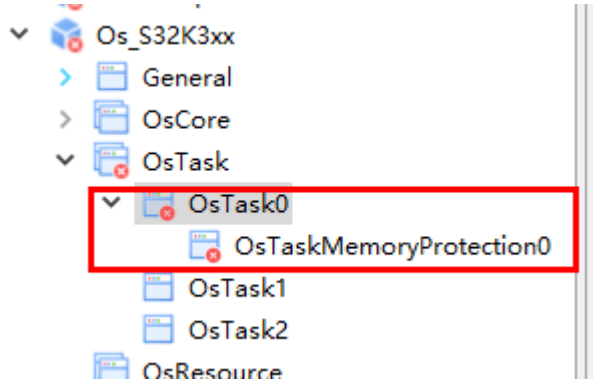


Figure6-39 add OsTaskMemoryProtection

Step 9: Choose Data Section for the Task from Memmap Module

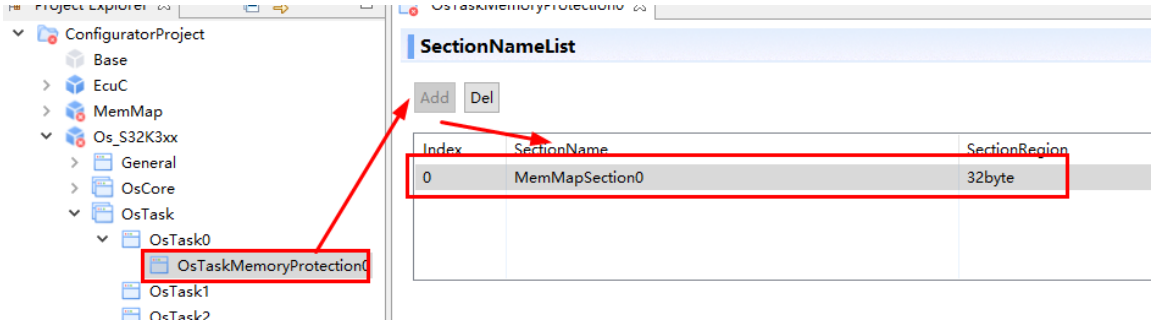


Figure6-40 choose data section for task

Step 10: Add Memory Protection for the ISR

为 ISR 添加内存保护段的方法与 Task 相同，参照 Step7-Step9。

Step A: Add Section in Memmap Module

1. 先添加 MemMapAddressingModeSet

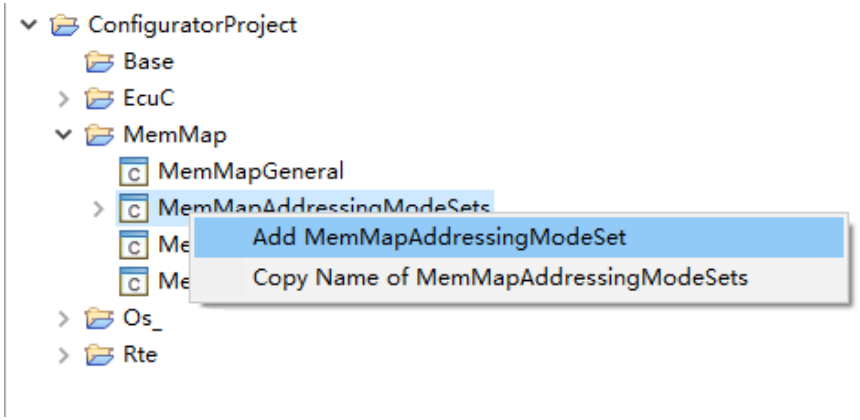


Figure6-41 add MemMapAddressingModeSet

2. 在 MemMapAddressingMode 中可以修改内存段的#pragma 预处理指令

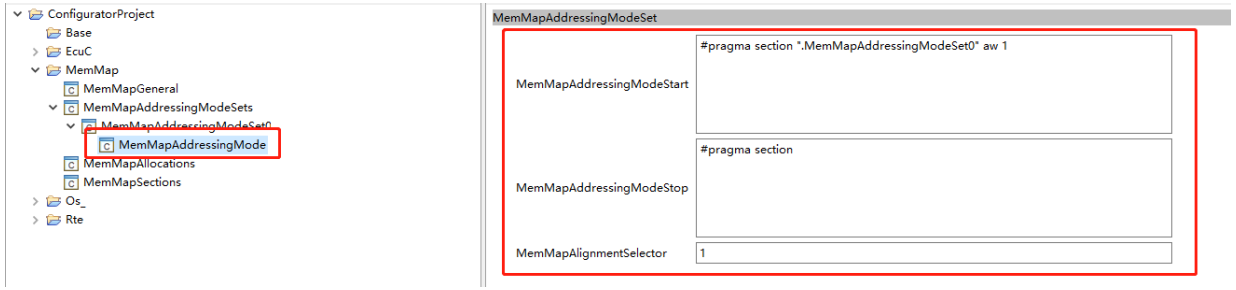


Figure6-41 Modify #pragma

3. 添加 MemMapSections

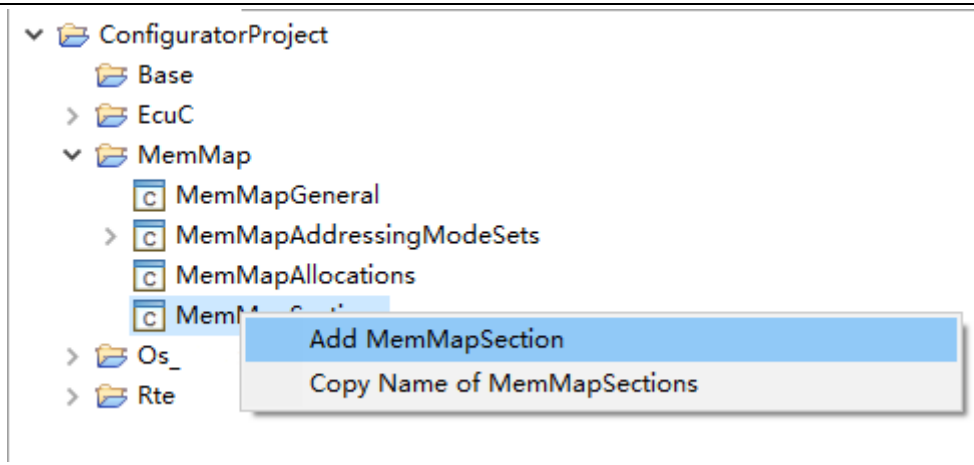


Figure6-41 add MemMapSection

4. 在新添加的 MemmapSection 中定义所需的内存映射宏定义

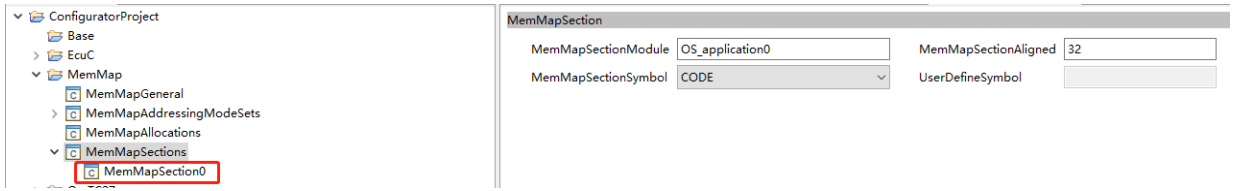


Figure6-42 define MemMapSection

5. 添加 MemMapAllocation

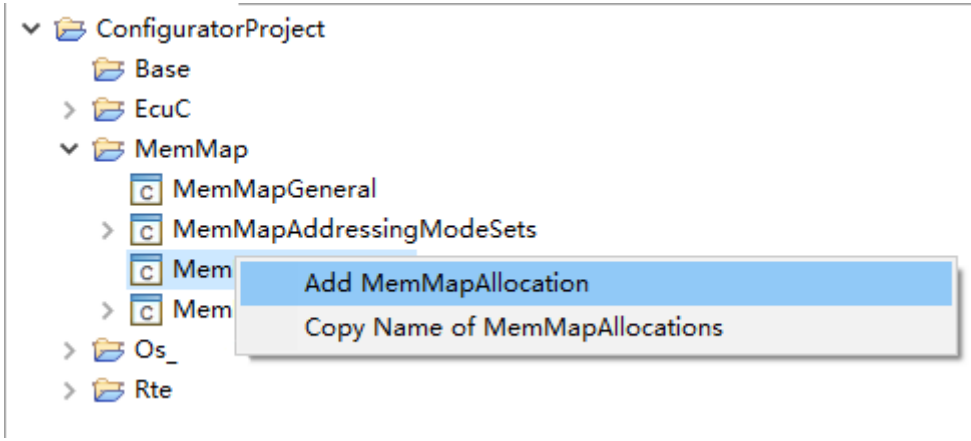


Figure6-43 add MemMapAllocation

6. 添加 MemMapSectionSpecificMapping

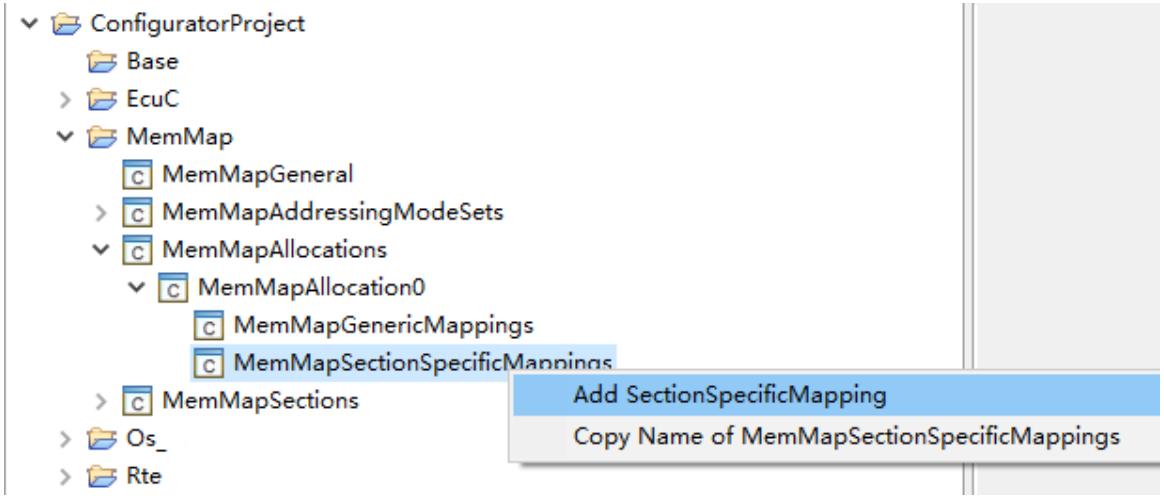


Figure6-44 add SectionSpecificMappling

7. 在 MemMapSectionSpecificMapping 中匹配#pragma 语句（MemMapAddressingModeSet）与内存分配关键字（MemMapSection）

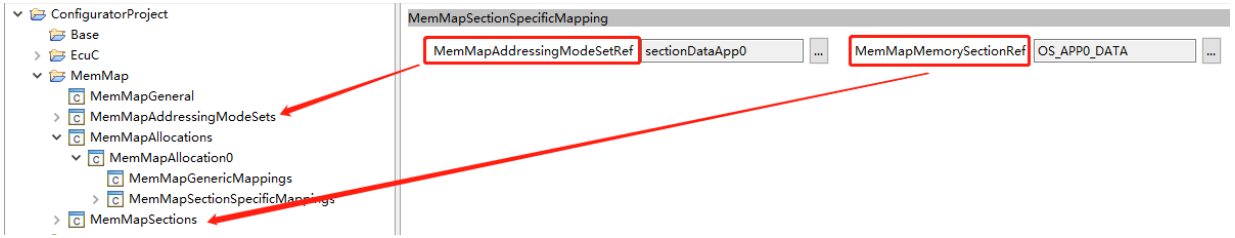


Figure6-45 mapping MemMapSection and MemMapAddressingModeSet

Step B: How to use ld file and locate data to specific section, please refer to specific chip user manual

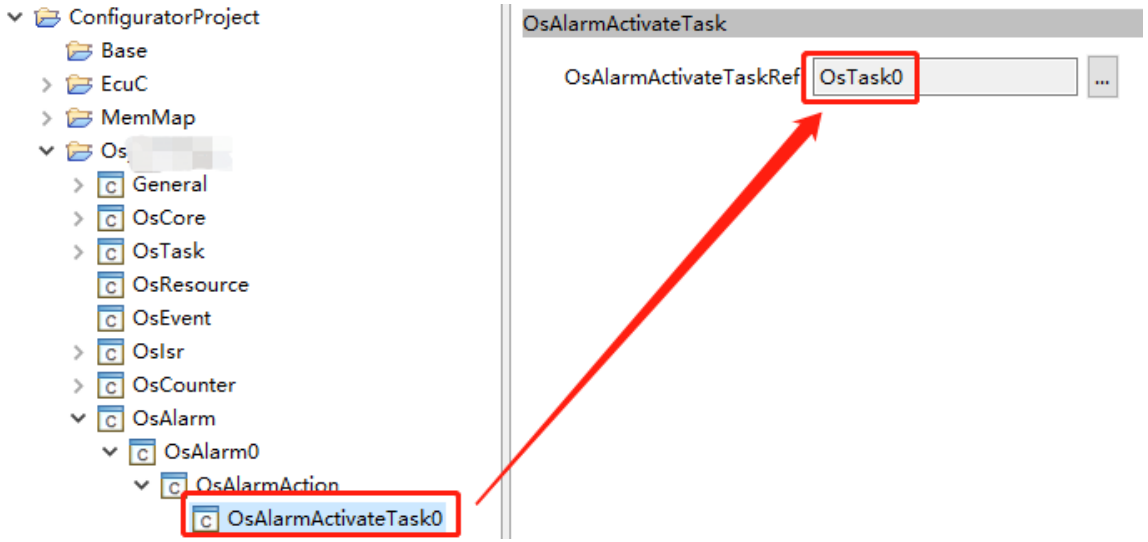
6.10 How to use Alarm to set offset of task startup

为了避免同时激活多个任务造成的堵塞，可以通过 Alarm 为各任务的启动时间设置偏移，例如：

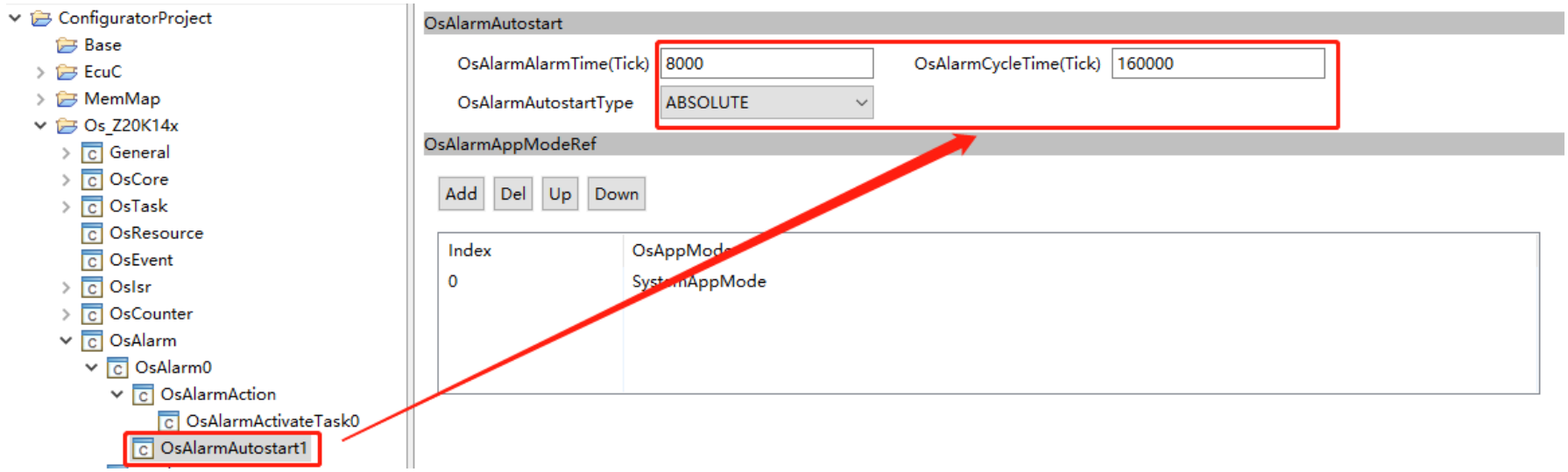
注：1 tick = 1/Frequency s，此处以 160MHz 为例

OsTask	OsTaskCycle(ms)	OsTaskOffset(us)
OsTask0	1	50
OsTask1	-	50
OsTask2	-	100
OsTask3	-	100

Step1：Set AutostartAlarm to Activate Task



Step2：Set AlarmTime and CycleTime of Autostart



Or Set Alarm in user codes:

```
TASK(OsTask0)
{
    SetAbsAlarm(OsAlarm1, 16000, 0);
    SetAbsAlarm(OsAlarm2, 32000, 0);
    SetAbsAlarm(OsAlarm3, 48000, 0);

    TerminateTask();
}
```

[Notice]

考虑到系统调度的开销，所有操作的时间间隔，包括但不限于启动时间，周期等，均应保证不小于 50 微秒。

The time interval for all operations, including but not limited to OsAlarmAlarmTime, OsAlarmCycleTime, etc., should be guaranteed not to be less than 50us, taking into account the overhead of system scheduling.

7 Notice

使用 OS 的注意事项，包括如下 3 部分：

（1）本文中有[Notice]标记的内容。

（2）EAS470_OS_SafetyManual 中列举的注意事项。

（3）参考不同芯片平台 OS 的使用说明。

The precautions for using the OS include the following 3 parts:

(1) There is content marked with [Notice] in this article.

(2) Precautions listed in EAS470_OS_SafetyManual.

(3) Refer to the user manual of different chip platform OS.

8 Limitations

OS 在进行平台测试时，以下表中的数据为上限进行了测试验证。如果用户使用的数量超过了下表，应在项目中进行进一步的边界测试。

When the OS is performing platform testing, the data in the following table is the upper limit for testing and verification. If the number of users used exceeds the following table, further boundary tests should be carried out in the project.

Table8-1 Limit on the number of each module

Objects	Maximum
OsCore	Ref Chip User Manual
Task	single core: 300, multi core: 500
Resource	single core: 300, multi core: 500
Event	100, Each task supports up to 32 events.
ISR	Ref Chip User Manual
Counter	single core: 5, multi core: 5 * core number
Alarm	single core: 300, multi core: 500
ScheduleTable	single core: 10, multi core: 10 * core number
OS-Application	single core: 10, multi core: 10 * core number Support max to 32 Applications.
IOC	20
Spinlock	20
Trusted Function	20

9 Appendix

Table9-1 Error Format

Service ID	Error Code	Error Info	Error Parameter	Description
OSServiceId_OsKernel_UpdateTaskHighestPri	E_OS_SYSFATA_READYQUEUE	OS_IE_READYQUEUE_EMPTY	OS_IE_NO_ERRPAR	任务就绪队列的数据异常。 The data in the task ready queue is abnormal.
OSServiceId_OsKernel_InsertTask		OS_IE_READYQUEUE_FULL		
OSServiceId_OsKernel_RemoveTask		OS_IE_READYQUEUE_DATAERR		
OSServiceId_OsKernel_CheckTaskSwitch		OS_IE_NOMORE_INFO		
OSServiceId_OsKernel_EnterTask	E_OS_SYSFATA_CONTEXTCRC_ERR			任务保存上下文校验失败。 Task context CRC check error.
OSServiceId_OsApp_InternalTerminateApplication	E_OS_SYSFATA_RESTART_TASK_INVALID	OS_IE_NOMORE_INFO	Application	重启 OS-Application 的 TaskId 无效。 Invalid TaskId to restart OS-Application/
	E_OS_SYSFATA_RESTART_TASK_	OS_IE_NOMORE_INFO	Application	重启 OS-Application 时激活任务失败。 Activation task failed when OS-Application was

	FAIL			restarted.
OSServiceId_StartOS OSServiceId_TerminateTask OSServiceId_ChainTask OSServiceId_OsCpu_TaskSwitch	E_OS_SYSFATA_UNREACHABLE	OS_IE_NOMORE_INFO	OS_IE_NO_ERRPAR	The task schedule did not execute as expected.
OSServiceId_OsCnt_TreeInsert	E_OS_SYSFATA_CNTTREE	OS_IE_NOMORE_INFO	OsPrex	插入 Counter 数据的队列异常。 Queue exception inserting counter data.
OSServiceId_OsCnt_TreeDel			OsChilD	删除 Counter 数据的队列异常。 Queue exception deleting counter data.
OSServiceId_OsCnt_TreeRemove				移除 Counter 数据的队列异常。 Queue exception removing counter data.
OSServiceId_OsIsr_EnterISR	E_OS_SYSFATA_CONTEXTCRC_ERR	OS_IE_NOMORE_INFO	OS_IE_NOMORE_INFO	中断上下文 CRC 校验错误。 C2 ISR context CRC check error.
OSServiceId_OsIsr_InISR1 OSServiceId_OsIsr_OutISR1	E_OS_SYSFATA_ISR_NEST_LIMIT_ERROR	OS_IE_NOMORE_INFO	Os_C1IntNestDepth[CoreID]	一类中断嵌套深度超范围。 C1 ISR nesting depth is beyond the range.

OSServiceId_OsIsr_InISR2			Os_IntNestDepth[CoreID]	二类中断嵌套深度超范围。 C2 ISR nesting depth is beyond the range.
OSServiceId_OsIsr_OutISR2	E_OS_SYSFATA_ISR_NEST_ID_ERROR		currentIsrId	中断嵌套保存的 ISRID 不合法。 Illegal ISRID to C2 ISR nested save.
OSServiceId_SuspendAllInterrupts	E_OS_SYSFATA_ISRAPI_NEST_LIMIT_ERROR		OS_IE_NO_ERRPAR	开关中断 API 调用超限。 Suspend/Resume interrupt API call overrun.
OSServiceId_ResumeAllInterrupts				
OSServiceId_SuspendOSInterrupts				
OSServiceId_ResumeOSInterrupts				
OSServiceId_OsMultiCore_TriggerActivateTask	E_OS_SYSFATA_RES_CROSSCORE_TIMEOUT			跨核操作超时。 Cross core operation timed out.
OSServiceId_OsMultiCore_TriggerSetEvent				
OSServiceId_OsMultiCore_TriggerSetAbsAlarm				
OSServiceId_OsMultiC				

ore_TriggerSetAbsAlarm				
OSServiceId_OsMultiCore_TriggerCancelAlarm				
OSServiceId_OsMultiCore_TriggerShutdown				
OSServiceId_OsMultiCore_TriggerShutdown				
OSServiceId_GetResource	E_OS_SYSFATA_RES_NEST_LIMIT_ERROR			中断资源嵌套深度超限。 The nesting depth of interrupt resource exceeds the limit.
OSServiceId_OsTp_DelTpInfo	E_OS_SYSFATA_TP_TIMER_ERROR		elapsedTime	时间保护计数器值错误。 Timing protection counter value error.
OSServiceId_OsTp_AddNewTpInfo				
OSServiceId_OsTp_PauseInternal				
OSServiceId_OsTp_TskExeBudgetStart	E_OS_SYSFATA_TP_INDEX_ERROR		TaskId	记录的时间保护的嵌套信息不合法。 Illegal nested information for time protection of record.
OSServiceId_OsTp_TskExeBudgetStop				

OSServiceId_OsTp_TskResourceLocktimeStart				
OSServiceId_OsTp_TskResourceLocktimeStop				
OSServiceId_OsTp_IsrExeBudgetStop			ISRID	
OSServiceId_OsTp_IsrLockTimeStart				
OSServiceId_OsTp_IsrLockTimeStart				
OSServiceId_OsTp_IsrLockTimeStop				
OSServiceId_OsTp_IsrLockTimeStop				
OSServiceId_OsTp_IsrResourceLocktimeStart				
OSServiceId_OsTp_IsrResourceLocktimeStop				