RZ/A1H Group

OS Porting Layer "OSPL" Sample Program

R01AN1887EJ0160
Rev.1.60
Oct. 26, 2017

Introduction

This application note describes the sample program named `OSPL' which abstracts OS less and OS-using environment.

The OS Porting Layer "OSPL" sample program offers the following features:

- Provides common API to OS less and OS-using environment
- Localizes modifying code for new OS environment
- Allocates asynchronous event bit automatically and able to make the pipeline.
- Provides API of cache flush.
- Provides timer basic API by using the timer hardware.
- Provides API to measure CPU load (for OS less only)
- Provides API of error handling with debug functions.
- If "#define R_OSPL_NDEBUG 1" was set, the debug function and a part of assertion is disabled and software become faster and compact.

Target Device

RZ/A1H Group

RZ/A1M Group

RZ/A1LU Group

RZ/A1L Group

When applying the sample program covered in this application note to another microcomputer, modify the program according to the specifications for the target microcomputer and conduct an extensive evaluation of the modified program.

Please contact us for the project for RSK board.

This package contains OSPL for test. See 4.4.17. R_OSPL_TEST_CODE.

Table of Contents

1. Specifications

Table 1.1 shows the Peripheral Functions to be Used and their Uses. Figure 1-1 shows an Operation environment of driver example and test.

Table 1.1 Peripheral Functions to be Used and their Uses

Peripheral functions	Uses
OS Timer (OSTM0, 1) or Multi-Function Timer Pulse Unit 2 (MUT2 - ch1, 2)	Uses in Free running timer (4.6.15. (2)). Using channel can be selected by "R_OSPL_FTIMER_IS" macro. And, uses in the sample driver.
Interrupt Controller (INTC) (Interrupt ID : DMAINT0~3)	DMAC interrupts (Uses in the sample driver)
Direct Access Controller (DMAC) Ch.(0~3)	DMA transfer (Uses in the sample driver)

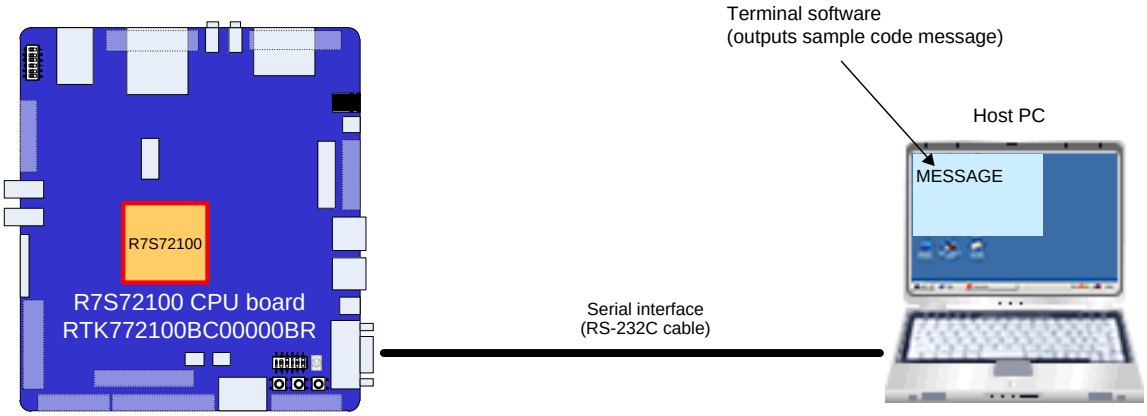


Figure 1-1 Operation environment

2. Operation Check Conditions

The sample code contained in this application note and test program have been checked under the conditions listed below. The way to change the target OS is described at the section of 4.8.1. Selecting the target (use_list.h, mcu_board_select.h).

Table 2.1 Operation Check Conditions

Item		Description
MCU used		RZ/A1H
Operating frequency		CPU clock (Iϕ): 400MHz Image processing clock (Gϕ): 266.67MHzϕ Internal bus clock (Bϕ): 133.33MHz Peripheral clock 1 (P1ϕ): 66.67MHz Peripheral clock 0 (P0ϕ): 33.33MHz
Operating voltage		Power supply voltage (I/O): 3.3V Power supply voltage (Internal): 1.18V
ARM	Integrated development environment	ARM® integrated development environment ARM Development Studio 5 (DS-5TM) Version 5.16
	C compiler	ARM C/C++ Compiler/Linker/Assembler Ver.5.03 [Build 102]
IAR	Integrated development environment	IAR Embedded Workbench for ARM 7.80.4.12495
	C compiler	
Renesas	Integrated development environment	e2 studio (Version: 5.3.0.023)
	C compiler	GNUARM-NONE-EABI v16.01
Operating mode		Boot mode 0 (CS0 space 16bit boot)
Communication setting of terminal software		● Communication speed: 115200bps ● Data length: 8 bits ● Parity: None ● Stop bit length: 1 bit ● Flow control: None
Board used		GENMAI board ● RTK772100BC00000BR (R7S72100 CPU board) ● RTK77210000B00000BR (R7S72100 Option board)
OS		● OS less ● CMSIS-RTOS RTX 4.80
Device used		Serial interface (D-sub 9-pin connector)

3. Reference Application Note(s)

For additional information associated with this document, refer to the following application note(s).

- RZ/A1H Group Example of Initialization (R01AN1646EJ)
- RZ/A1H Group I/O definition header file <iodefine.h> (R01AN1860EJ)
- RZ/A1H Group CMSIS-RTOS RTX BSP V2.03 release note (R01AN2200EJ)
- RZ/A1H Group CMSIS-RTOS RTX BSP V2.06 (e2studio / KPITGCC) (R01AN3104EJ)
- RZ/A1H Group CMSIS-RTOS RTX BSP V2.03 ICCARM release note (R01AN2990EJ)
- Board Support Package Module Using Firmware Integration Technology (R01AN1685EU)

This document does not explain the guide of driver development.

4. Description of Software

4.1. Operation Outline

Figure 4-1 shows the Figure of system block.

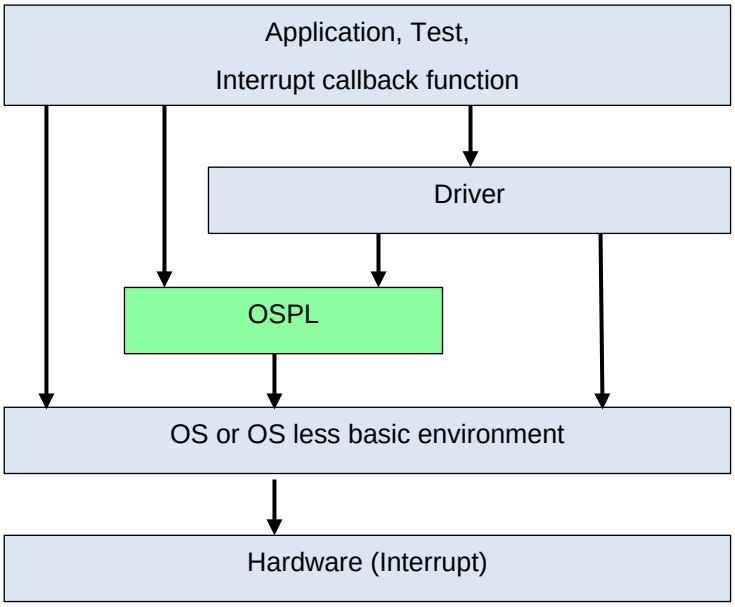


Figure 4-1 Figure of system block

Refer to (4.7.) Figure of sequence.

OSPL provides common API for OS less and for OS-using environment. It becomes easy to port between OS less and OS-using environment.

The target software is the driver which responds to interrupt, starts asynchronous operation or waits something.

It provides the synchronous API which waits something and the asynchronous API corresponding to it. OSPL helps to make usable software and to support pseudo multithreading. Even if the driver which does not support OSPL's common functions for driver's APIs and has the API related an interrupt was used, it becomes high portability by calling OSPL API from the application.

OSPL supports the interrupt response operation, the memory management, the time, the idle state and the error handling. It does not support the communication between threads. When these operations are used in OS-using environment, call each OS function directly from the "driver wrapper" over the driver for OS less.

Table 4.1 describes OSPL supported functions.

Table 4.1 Functions of OSPL

Category	Function	OS less	OS-using	Description
Thread	Thread management	✓	x	Creates a thread (Pseudo thread for OS less) Sets and gets the parameter of the thread Sets thread ID Sets the behavior on waiting
		✓	✓	Gets thread ID
	Lock	✓	✓	Lock and unlock
Event	Flag management	✓	✓	Sets, clears and waits thread attached flags Sets, clears and checks normal flags
	Queue	✓	✓	Transfers between threads
Interrupt	Disable and enable	✓	✓	Disables and enables all interrupts Registers a callback function Registers an interrupt handler Disables and enables each interrupt and sets priority of each interrupt
Object	Lock	✓	✓	Lock and unlock
Memory	Cache	✓	✓	Cache operation
	Address conversion	✓	✓	Converts an address of cached area and uncached area Converts an address of virtual and physical
Time	Wait	✓	✓	Waits during specified time
	Reference	✓	✓	Refers free running timer
Debug	CPU	✓	x	Measures and outputs CPU load
	Debug	✓	✓	Manages unrecoverable errors Error break Assertion (programming by contract) Saves and outputs error information Watch, fast logging and through counter Checks stack overflow
Coding	Readability	✓	✓	Avoid less readability by committing for static code analyzer

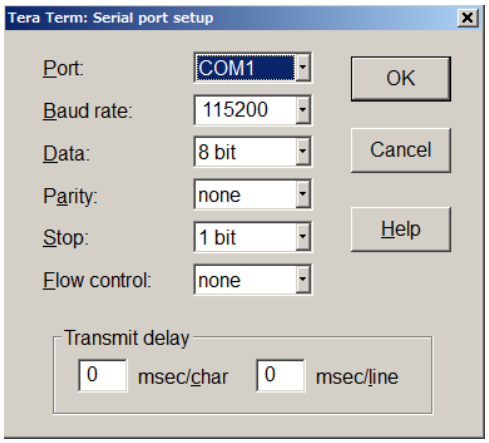
✓ : Functions supported with OSPL

x : Functions not supported with OSPL

4.1.1. Preparations

The following preparations in Sample Code.

Terminal software is started in a host PC and it's established as follows. (In the case of Tera Term)



When a sample program is executed, a message is output at a terminal as follows.

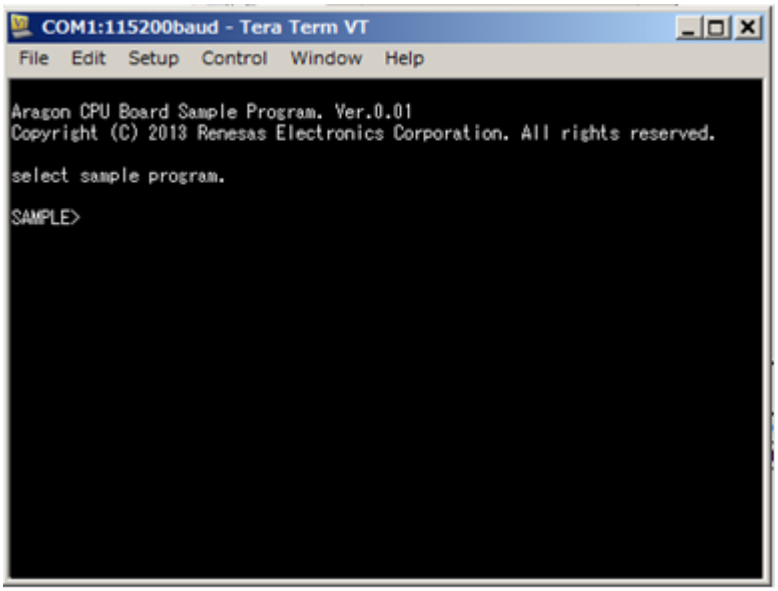


Figure 4-2 Message output at sample program execution

4.2. Interrupt

Table 4.2 shows Interrupts using by OSPL sample driver code. The operation in the interrupt is same as the operation of "R_DRIVER_OnInterrupting" and "R_DRIVER_OnInterrupted" function of each drivers.

Table 4.2 Interrupts using by OSPL sample driver code

Interrupt (Source ID)	Priority	Summary
DMAINT0~DMAINT3	7	Receives the interrupt of DMAINT#
OSTM#TINT(#=0..1)	2	Receives the interrupt of OSTM#TINT

4.3. Basic Types

Table 4.3 shows Basic types using in the example code.

Table 4.3 Basic types using in the example code

Symbol	Description
char_t	8-bit character
bool_t	Boolean data type. The value is true (1) or false (0).
bool8_t	8bit boolean data type. The value is true (1) or false (0).
bool16_t	16bit boolean data type. The value is true (1) or false (0).
bool32_t	32bit boolean data type. The value is true (1) or false (0).
int_t	The signed integer for this library is a 32-bit signed integer.
int8_t	8-bit signed integer (defined by standard library)
int16_t	16-bit signed integer (defined by standard library)
int32_t	32-bit signed integer (defined by standard library)
int64_t	64-bit signed integer (defined by standard library)
uint8_t	8-bit unsigned integer (defined by standard library)
uint16_t	16-bit unsigned integer (defined by standard library)
uint32_t	32-bit unsigned integer (defined by standard library)
uint64_t	64-bit unsigned integer (defined by standard library)
int_fast8_t	Fastest 8-bit minimum-width signed integer
int_fast16_t	Fastest 16-bit minimum-width signed integer
int_fast32_t	Fastest 32-bit minimum-width signed integer
uint_fast8_t	Fastest 8-bit minimum-width unsigned integer
uint_fast16_t	Fastest 16-bit minimum-width unsigned integer
uint_fast32_t	Fastest 32-bit minimum-width unsigned integer
uintptr_t	Same as pointer bit width unsigned integer as physical address
size_t	Same as pointer bit width unsigned integer as byte size
ssize_t	Same as pointer bit width signed integer
ptrdiff_t	Same as pointer bit width signed integer as difference between pointers
bit_flags_fast32_t	Same as uint_fast32_t bit flags (bit field)
bit_flags32_t	Same as uint32_t bit flags (bit field)
bit_flags16_t	Same as uint16_t bit flags (bit field)
bit_flags8_t	Same as uint8_t bit flags (bit field)
byte_t	Type of 1 byte
float32_t	32-bit float (Defined by standard library when "__ARM_NEON__" defined)
float64_t	64-bit float (Defined by standard library when "__ARM_NEON__" defined)
float128_t	128-bit float

4.4. Constants, Enumerations and Error code

Section	Type Symbol	Description
4.4.1.	-	Version
4.4.2.	errnum_t	Error Codes
4.4.3.	r_ospl_async_state_t	State
4.4.4.	r_ospl_wait_t	The parameter of "R_OSPL_THREAD_SetOnWait" function
4.4.5.	r_ospl_flush_t	The parameter of "R_OSPL_MEMORY_Flush" function
4.4.6.	r_ospl_axi_cache_attribute_t	Cache attribute of AXI bus
4.4.7.	r_ospl_axi_protection_t	Protection attribute of AXI bus
4.4.8.	r_ospl_async_type_t	Kind of the asynchronous operation
4.4.9.	bsp_int_err_t	Error Codes of FIT*[1] BSP
4.4.10.	r_ospl_event_flags_t	Bit flags or ID of event
4.4.11.	r_ospl_table_flags_t	Option of table of array number
4.4.12.	r_ospl_if_not_t	Behavior when searching one was not found
4.4.13.	bsp_int_src_t	Interrupt number
4.4.14.	bsp_int_cb_t	Interrupt handler
4.4.15.	bsp_int_cmd_t	Control command related to the interrupt
4.4.16.	mcu_lock_t	Hardware channel number for locking
4.4.17.	-	The other constant values

4.4.1. Version

Symbol	Value	Description
R_OSPL_VERSION	155	Version number of OSPL Hundreds place is version number of OSPL specification. Tens place and one's place are minor version number in specified OS and board.
R_OSPL_VERSION_STRING	"1.55"	String of version number of OSPL. e.g.) "1.00"
R_OSPL_IS_PREEMPTION	0 or 1	Whether preemptive RTOS or not. This value is 0, if the environment was OS less. This value is 1, if preemption was occurred on interrupt or calling API, even if not round robin. It is necessary to configure to pseudo multithreading, if this value was 0.
BSP_CFG_RTOS_USED	0 or 1	Same as "R_OSPL_IS_PREEMPTION"
BSP_CFG_PARAM_CHECKING_ENABLE	0 or 1	Enable or Disable parameter check of the system 1=The system checks except for the code disabled by defined "R_OSPL_NDEBUG" 0=The system does not check all
R_OSPL_FOR_FREE_RTOS	80102	Target version of FreeRTOS. The digit in the ten-thousand's place is major version. The hundred's place is minor version. The one's place is build number.
R_OSPL_FOR_RTX	((4<<16)|74)	Target version of RTX. It is same as the value of "osCMSIS_RTX".
R_OSPL_FOR_RZ_A1_BSP	205	Target version of RZ/A1H RTX BSP. The value is 100 times.

*[1] Board Support Package Module Using Firmware Integration Technology (R01AN1685EU0260)

R_OSPL_FOR_RZ_A1_OS_LESS	101	Target version of OS less sample for RZ/A1H. The value is 100 times.
R_OSPL_FOR_RZ_A1_INTC	101	Target version of interrupt controller sample (INTC). The value is 100 times.
R_OSPL_FOR_RZ_A1_CACHE	101	Target version of cache controlling sample. The value is 100 times.

4.4.2. Error Codes

Symbol	Value	Description
0	0	No error is detected.
E_OTHERS	1	Others error
E_FEW_ARRAY	2	Error of few fixed length array
E_FEW_MEMORY	3	Few heap memory area
E_FIFO_OVER	4	Failed to enqueue
E_NOT_FOUND_SYMBOL	5	Not defined the symbol
E_NO_NEXT	6	There is not next element of list
E_ACCESS_DENIED	7	Error of denied read or write. The object is locked by other thread.
E_NOT_IMPLEMENT_YET	9	Not implemented yet
E_ERRNO	0x0E(=14)	Refer to "errno"
E_LIMITATION	0x0F(=15)	Limitation
E_STATE	0x10(=16)	Cannot do at this state
E_NOT_THREAD	0x11(=17)	Not a thread, cannot call from interrupt context.
E_PATH_NOT_FOUND	0x12(=18)	Not found file or folder
E_BAD_COMMAND_ID	0x16(=22)	Out of number of command ID
E_TIME_OUT	0x17(=23)	Time out
E_STACK_OVERFLOW	0x28(=24)	Stack overflow
E_NO_DEBUG_TLS	0x1D(=29)	Not set debug work area. Refer to "R_OSPL_SET_DEBUG_WORK"
E_EXIT_TEST	0x1E(=30)	Request of exit from the test

4.4.3. r_ospl_async_state_t type - State

Symbol	Value	Description
R_OSPL_UNINITIALIZED	0	State of not initialized. The initial value of global variables
R_OSPL_RUNNABLE	1	State to be able to call asynchronous operations
R_OSPL_RUNNING	2	State of running asynchronous operations or waiting interrupts
R_OSPL_INTERRUPTING	3	State with need to call "R_DRIVER_OnInterrupting" function
R_OSPL_INTERRUPTED	4	State with need to call "R_DRIVER_OnInterrupted" function
R_OSPL_LOCKED	5	State of using except for the driver

4.4.4. r_ospl_wait_t type - The parameter of R_OSPL_THREAD_SetOnWait function

Symbol	Value	Description
R_OSPL_WAIT_POLLING	0	Wait by polling

R_OSPL_WAIT_PM_THREAD	1	Wait by pseudo multi-threading If the function with waiting was called, time out argument is ignored and the function returns immediately. Call R_OSPL_THREAD_GetIsWaiting(4.6.3. (13)) after calling the function with waiting. However, if time out argument is "R_OSPL_INFINITE", the function does not return. Because 'R_OSPL_THREAD_GetIsWaiting' function for OS less was called from the synchronized function "R_DRIVER_Transfer", the driver is depended on OS less. This setting keeps CPU load 100%.

4.4.5. r_ospl_flush_t type - The parameter of R_OSPL_MEMORY_Flush function

Symbol	Value	Description
R_OSPL_FLUSH_WRITEBACK_INVALIDATE	2	Write back (clean) to the memory and invalidate L1 cache
R_OSPL_FLUSH_WRITEBACK_INVALIDATE_2ND	8	Write back (clean) to the memory and invalidate L2 cache
R_OSPL_FLUSH_INVALIDATE	0	Invalidate L1 cache

4.4.6. r_ospl_axi_cache_attribute_t type - Cache attribute of AXI bus

Setting how bus master uses cache on AXI bus. For RZ/A1H, except for 'R_OSPL_AXI_CACHE_ZERO' is used to set the attribute of cache operation to the memory out of L2 cache PL310. CPU accesses with L2 cache and bus master except for CPU accesses without L2 cache, it is necessary to flush L2 cache (R_OSPL_FLUSH_WRITEBACK_INVALIDATE_2ND). Refer to: RZ/A1H Group User's Manual: Hardware (5.8) AXI Protocol Control Signals.

Symbol	Value	Description
R_OSPL_AXI_CACHE_ZERO	0	For internal bus
R_OSPL_AXI_STRONGLY	0	Strongly ordered memory
R_OSPL_AXI_DEVICE	1	Device memory
R_OSPL_AXI_UNCACHED	3	Uncached normal access
R_OSPL_AXI_WRITE_BACK_W	11	Cached write back. Allocates when writing
R_OSPL_AXI_WRITE_BACK	15	Cached write back. Allocates when reading and writing

4.4.7. r_ospl_axi_protection_t type - Protection attribute of AXI bus

For RZ/A1H, this is secure attribute to SDRAM out of L2 cache PL310 on AXI bus. Secure attribute operates whether or not to hit L2 cache. Secure attribute of CPU is corresponding to NS bit in MMU (TTB). Refer to: RZ/A1H Group User's Manual: Hardware (5.8) AXI Protocol Control Signals.

Symbol	Value	Description
R_OSPL_AXI_PROTECTION_ZERO	0	For internal bus
R_OSPL_AXI_SECURE	0	Secure
R_OSPL_AXI_NON_SECURE	2	Non-secure

4.4.8. r_ospl_async_type_t

Symbol	Value	Description
R_OSPL_ASYNC_TYPE_NORMAL	1	Normal asynchronous operation
R_OSPL_ASYNC_TYPE_FINALIZE	2	Asynchronous operation for finalize

4.4.9. bsp_int_err_t

Symbol	Value	Description
BSP_INT_SUCCESS	0	No error is detected
BSP_INT_ERR_NO_REGISTERED_CALLBACK	0x2101	Not registered interrupt callback function
BSP_INT_ERR_INVALID_ARG	1	Parameter error
BSP_INT_ERR_UNSUPPORTED	15	Not supported

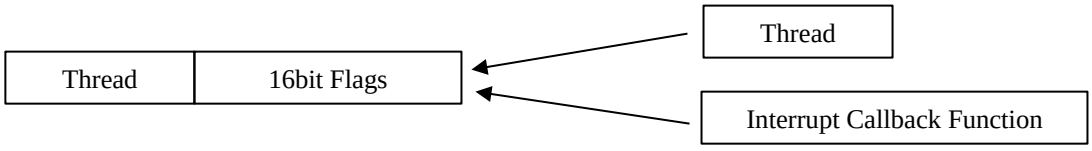
4.4.10. r_ospl_event_flags_t

"r_ospl_event_flags_t" is data type of thread attached event bit flags.

It is usually to set notify target to a variable as "r_ospl_event_flags_t" type in "r_ospl_async_t" type and pass the variable to an operation target.

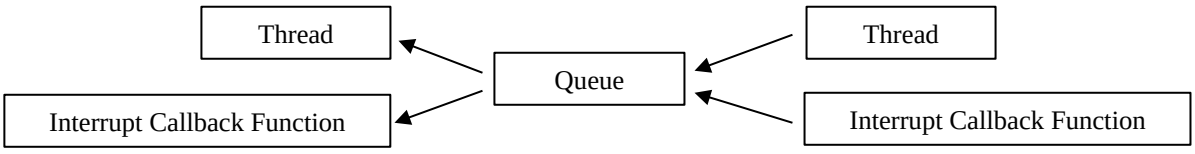
r_ospl_thread_id_t type's value of variable	r_ospl_event_flags_t type's value of variable
Notify target thread ID	Bit of thread attached event flags (+ check bits)

Thread attached event:



Queue (r_ospl_queue_id_t) is sometimes used to sending data (asynchronous communication). But queue is usually hidden in library API.

Queue:



4.4.11. r_ospl_table_flags_t

Symbol	Value	Description
R_OSPL_TABLE_I_LOCK	2	Whether I-lock area is made in each function.
R_OSPL_TABLE_T_LOCK	1	Whether T-lock area is made in each function.

4.4.12. r_ospl_if_not_t

Constant value of behavior, when searching key was not found.

Symbol	Value	Description
R_OSPL_ALLOCATE_IF_NOT	1	Searches and appends unused array number, if specified key was not found. "E_FEW_ARRAY" error is raised, if there is not unused array's index number. Returns the array number, if specified key was found.
R_OSPL_ERROR_IF_NOT	0	"E_NOT_FOUND_SYMBOL" error is raised, if specified key was not found . Returns the array's index number, if specified key was found.
R_OSPL_DO_NOTHING_IF_NOT	2	Keeps output variable's value as same as input value, if specified key was not found. Output the array's index number, if specified key was found. This is used for checking already key exists.
R_OSPL_OUTPUT_IF_NOT	3	Searches and appends unused array number, if not found specified key. "E_FEW_ARRAY" error is raised, if there is not unused array's index number. Keeps output variable's value as same as input value, if specified key was found. This is used for checking already key exists
R_OSPL_ALLOCATE_IF_EXIST_OR_IF_NOT	4	Searches and appends unused array number, even if specified key was found or not. "in_Key" argument of "R_OSPL_TABLE_GetIndex" function is ignored. "E_FEW_ARRAY" error is raised and any key is not appended, if there is not unused array's index number. After "R_OSPL_ALLOCATE_IF_EXIST_OR_IF_NOT" was specified, the other constant as "r_ospl_if_not_t" type cannot be specified. And "R_OSPL_TABLE_Free" function cannot be called. "R_OSPL_ALLOCATE_IF_EXIST_OR_IF_NOT" can be specified after calling "R_OSPL_TABLE_InitConst".

4.4.13. bsp_int_src_t

This is type of interrupt number.

Type of "bsp_int_src_t" is integer type or enumeration type by the environment.

The symbol of value is started from "BSP_INT_SRC_".

4.4.14. bsp_int_cb_t

This is type of interrupt handler.

Type of return value or arguments of function type of "bap_int_cb_t" is different from the environment.

Interrupt callbacks via "R_OSPL_CallInterruptCallback" in OSPL environment.

4.4.15. bsp_int_cmd_t

Symbol	Value	Description
BSP_INT_CMD_INTERRUPT_ENABLE	0	Enables the interrupt. "FIT_NO_PTR" should be set at "Parameter" argument of "R_BSP_InterruptControl" function.
BSP_INT_CMD_INTERRUPT_DISABLE	1	Disables the interrupt. "FIT_NO_PTR" should be set at "Parameter" argument of "R_BSP_InterruptControl" function.

4.4.16. mcu_lock_t

This is enumerated type about identity number of hardware channel for lock operation. Naming rule is to start from 'BSP_LOCK_' and upper snake case.

Define example:

```
typedef enum {
    BSP_LOCK_DMAC0,
    BSP_LOCK_DMAC1,
           :
    BSP_NUM_LOCKS /* This entry is not a valid lock. It is used for sizing
       g_bsp_Locks[] array below. Do not touch! */
} mcu_lock_t;
```

4.4.17. The other constant values

Symbol	Value	Description
R_OSPL_NDEBUG	Not defined or 1	Debug configuration (not defined) or Release configuration (1). This is same as "NDEBUG" of standard library. The system can run with the debug configuration OSPL and the release configuration application. If the library (compiled binary) called the debug configuration OSPL, compile the OSPL source with debug configuration.
R_OSPL_TEST_CODE	0 or 1	A macro of whether the code for test is enabled or not. "ospl" folder in OSPL package contains the code for test. There is the OSPL without the code for test in "workspace\src". The code for test is written in branches with R_OSPL_TEST_CODE macro. "*_test_*" file is source file for the test.
R_OSPL_ERROR_BREAK	0 or 1	A macro of whether to use error break function
R_OSPL_PRINTF	(See Description)	Setting of whether printf output is enabled or not in OSPL. Select from: ● R_OSPL_PRINTF_ENABLED ● R_OSPL_PRINTF_DISABLED ● R_OSPL_PRINTF_TO_INT_LOG
R_OSPL_INTERRUPT_HANDLER_IS	(See Description)	Kind of interrupt handler. Select from: ● R_OSPL_INTERRUPT_HANDLER_IS_GENERAL ➢ The interrupt handler does not have any arguments. ➢ The interrupt handler returns nothing. ➢ API is not called at the last of the interrupt hander, ● R_OSPL_INTERRUPT_HANDLER_IS_WITH_INT_SENSE ➢ The interrupt handler has "int_sence" argument. ● R_OSPL_INTERRUPT_HANDLER_IS_FOR_RTX

		➢ This is for RTX. It is same as "R_OSPL_INTERRUPT_HANDLER_IS_GENERAL".
R_OSPL_TLS_ERROR_CODE	0 or 1	Whether the error code stores in TLS(Thread Local Storage) This value must be 1 by some application or some library with OSPL.
R_OSPL_STACK_CHECK_CODE	0 or 1	Setting of whether stack check corresponding code is enabled or not. 1 means to do checking in the stack just before judging in "IF" macro. 0 means disabled following functions and constants. ● R_OSPL_GET_STACK_POINTER ● R_OSPL_SET_END_OF_STACK ● R_OSPL_MOVE_END_OF_STACK ● R_OSPL_RESET_MIN_FREE_STACK_SIZE ● R_OSPL_GET_MIN_FREE_STACK_SIZE ● R_OSPL_STACK_CHECK_CANARY_VALUE Some OS checks in the stack, when running thread was changing. The checking of the OS is not same as checking by OSPL.
R_OSPL_CPU_LOAD	0 or 1	Whether it is supported to measure CPU load (for OS less)
R_OSPL_DEBUG_TOOL	0 or 1	Whether debug tools function is defined
R_OSPL_TLS_EVENT_CODE	0 or 1	Setting of whether it is enabled or not to managing that thread attached event flags are used. See 4.6.4. (1) R_OSPL_EVENT_Allocate.
R_OSPL_DETECT_BAD_EVENT	0 or 1	Setting of checking no conflicted thread attached event flags by reading check bits.
R_OSPL_EVENT_MUST_ALLOCATE	0 or 1	Setting of whether "R_OSPL_RaiseUnrecoverable" function is called, when an event was used without allocation.
R_OSPL_ALL_EVENT_ALLOCATE	0 or 1	Whether it is supported to specify "R_OSPL_EVENT_ALL_BITS" at "R_OSPL_EVENT_Allocate".
R_OSPL_WAIT_OWNER_DEBUG	0 or 1	Setting of whether to record to Int log about the thread calling a function with waiting (For debug in OS-using environment). There is "R_OSPL_WAIT_OWNER_DEBUG" in OSPL for defined "R_OSPL_IS_PREEMPTION = 0" only. In pseudo multi-threading environment by calling "R_OSPL_THREAD_SetOnWait" function with "R_OSPL_WAIT_PM_THREAD", waiting thread must call same waiting function or "R_OSPL_THREAD_ExitWaiting" function. "R_OSPL_WAIT_OWNER_DEBUG" is used for investigating which function to be called.
R_OSPL_DEBUG_THREAD_COUNT	2 or more	Maximum count of threads managing in debug work area + maximum count of interrupt level. The debug work area is not made, when settings are "R_OSPL_ERROR_BREAK = 0" and "R_OSPL_TLS_ERROR_CODE = 0"
R_OSPL_LIBRARY_MAKING	0 or 1	This macro presents whether or not the library is making.

		1 means that member variables in work area structure cannot be referenced. "*_DEF" macro like "R_OSPL_QUEUE_DEF" cannot be referenced. It may affect define of "INLINE" macro.
R_OSPL_FLAG32_ALL_BITS	0xFFFFFFFF	All bits of "r_ospl_flag32_t" type
R_OSPL_STACK_CHECK_CANARY_VALUE	0x57AC512E	Canary value written at the end of stack and not used area.
BSP_CFG_USER_LOCKING_ENABLED	0 or 1	Whether to use user defined lock object. 0 = define following macros as using C-lock of OSPL 1 = define following macros by the user ● BSP_CFG_USER_LOCKING_TYPE = r_ospl_c_lock_t ● BSP_CFG_USER_LOCKING_HW_LOCK_FUNCTION = R_BSP_HardwareLock ● BSP_CFG_USER_LOCKING_HW_UNLOCK_FUNCTION = R_BSP_HardwareUnlock ● BSP_CFG_USER_LOCKING_SW_LOCK_FUNCTION = R_BSP_SoftwareLock ● BSP_CFG_USER_LOCKING_SW_UNLOCK_FUNCTION = R_BSP_SoftwareUnlock
R_OSPL_UNLOCKED_CHANNEL	0xFEE	This macro is for specifying unlocked channel number.
R_OSPL_FTIMER_IS	(See Description)	Channel using for free running timer function (4.6.15. (2)) Sample program uses following symbols: ● R_OSPL_FTIMER_IS_OSTM1 (OS less) ● R_OSPL_FTIMER_IS_MTU2_1_2 (with OS)
R_OSPL_FTIMER_IS_OSTM0	0	OS Timer - channel 0. One of the value for R_OSPL_FTIMER_IS.
R_OSPL_FTIMER_IS_OSTM1	1	OS Timer - channel 1. One of the value for R_OSPL_FTIMER_IS.
R_OSPL_FTIMER_IS_MTU2_1_2	2	MUT2 - channel 1 and 2 cascade connection. One of the value for R_OSPL_FTIMER_IS.

4.5. Structures and Unions

Section	Symbol	Outline
4.5.1.	r_ospl_thread_id_t	Pointer to a thread
4.5.2.	r_ospl_thread_def_t	Definition of a thread
4.5.3.	r_ospl_flag32_t	Flag having 32bit
4.5.4.	r_ospl_event_group_id_t	Type of internal event flag
4.5.5.	r_ospl_event_status_t	Status of event
4.5.6.	r_ospl_async_t	Setting of notifications
4.5.7.	r_ospl_async_status_t	Structure of driver's status and interrupt status
4.5.8.	r_ospl_queue_id_t	Queue between threads
4.5.9.	r_ospl_queue_def_t	Define of queue
4.5.10.	r_ospl_queue_status_t	Status of queue
4.5.11.	BSP_CFG_USER_LOCKING_TYPE	C-lock
4.5.12.	r_ospl_c_lock_t	C-lock for internal
4.5.13.	r_ospl_table_t	Array index table
4.5.15.	r_ospl_memory_spec_t	Specification of memory or cache memory
4.5.16.	r_ospl_ftimer_spec_t	Specification of free running timer
4.5.17.	r_ospl_caller_t	Manager of an interrupt callback function
4.5.18.	r_ospl_interrupt_t	Structure related to interrupt source. e.g. interrupt number

4.5.1. r_ospl_thread_id_t

This is the type of pointer to a structure of a thread.

Member variables should not be accessed.

This has 16bit event internal.

This is created by "R_OSPL_THREAD_Create" function.

In OS-using environment, the thread type defined by OS and "r_ospl_thread_id_t" type defined by OSPL are same type. Use API of OS except for "R_OSPL_THREAD_GetCurrentId" function.

In OS less, preemption does not be supported (pseudo multi thread). Usually "r_ospl_thread_id_t" is used for divide to some threads for identifying over 16bit (16 types) events.

"r_ospl_thread_id_t" type variable can be let "R_OSPL_THREAD_NULL".

4.5.2. r_ospl_thread_def_t

This is the type of definition of a thread.

Member variables should not be accessed.

This is defined by "R_OSPL_THREAD_DEF" macro.

4.5.3. r_ospl_flag32_t

This is the type of flags having 32bit.

Member variables should not be accessed.

This is initialized by "R_OSPL_FLAG32_InitConst" function.

4.5.4. r_ospl_event_group_id_t

This is the type of internal event flags that has ID not as same as ID of thread in OSPL.

Bits over 16 bits is not used from OSPL, even if OS was supported over 16bit event group. The function writing to event group can be called from not only thread but also interrupt context.

"r_ospl_event_group_id_t" type is internally defined a type as same as type of event group defined by OS that does not support thread attached event. "r_ospl_event_group_id_t" type is not defined, when "R_OSPL_EVENT_GROUP_CODE = 0".

OSPL user must calls API of thread attached event. The event group is used internally. "R_OSPL_EVENT_Allocate" function internally calls "R_OSPL_EVENT_GROUP_Create" function and "R_OSPL_EVENT_GROUP_Create" gets an event group that was created by calling "R_OSPL_EVENT_Allocate" function. "R_OSPL_EVENT_Free" function internally calls "R_OSPL_EVENT_GROUP_Delete" function and "R_OSPL_EVENT_GROUP_Delete" function releases to delete the event group. Timing of creating and deleting is depended on implementation of "R_OSPL_EVENT_GROUP_Create" and "R_OSPL_EVENT_GROUP_Delete" function. It is not possible to change the thread waiting for the event group to another event group.

" r_ospl_event_group_id_t " type variable can be let "R_OSPL_EVENT_GROUP_NULL".

4.5.5. r_ospl_event_status_t

Outline	Status of event	
Header	r_ospl.h	
Description	This can be gotten by "R_OSPL_EVENT_GetStatus" function. This is enabled, when "R_OSPL_DETECT_BAD_EVENT= 1".	
Member variable	bit_flags32_t AllocatedEvents	Bits allocated event. See R_OSPL_EVENT_Allocate
	bit_flags32_t UnexpectedEvents	Bits set as not expected

4.5.6. r_ospl_async_t

Outline	Setting of notifications	
Header	r_ospl.h	
Description	This is set to "R_DRIVER_TransferStart" function. See. R_OSPL_EVENT_Wait (4.6.4. (4)). Details of notification sometimes get from API of the module having queue.	
Member variable	bit_flags_fast32_t Flags	Flagged structure parameters. Refer to Section (4.8.2.) ● R_F_OSPL_A_Thread ● R_F_OSPL_A_EventValue ● R_F_OSPL_I_Thread ● R_F_OSPL_I_EventValue ● R_F_OSPL_InterruptCallback ● R_F_OSPL_Delegate The following preset (set of member's values) can be set with above flags. ● R_F_OSPL_AsynchronousPreset See "R_OSPL_ASYNC_SetDefaultPreset" function.
	void* Delegate	The value defined by applications. Drivers and OSPL does not access it. It is able to reference from the argument of interrupt callback function "R_F_OSPL_Delegate" set with "Flags" member variable is ignored.
	r_ospl_thread_id_t A_Thread	The A-Thread waiting the A-Event receiving notifications when any asynchronous operations were end. This member variable must be set with "R_F_OSPL_A_Thread".

	The default value is R_OSPL_THREAD_NULL. If R_OSPL_THREAD_NULL was set, notifications are not received. Normally, the return value of "R_OSPL_THREAD_GetCurrentId" function is set. Normally, call "R_OSPL_EVENT_Wait" function, if you want to receive notifications.
r_ospl_event_flags_t A_EventValue	The notify target set to the A-Event. This member variable must be set with "R_F_OSPL_A_EventValue". Event flags set to notify target. Flags are 16bit. Default is "R_OSPL_UNUSED_FLAG". However, the default value is changed by "R_DRIVER_SetDefaultAsync" function. This member variable will be changed by calling "R_OSPL_EVENT_Allocate" or "R_OSPL_EVENT_Free" in target driver.
r_ospl_thread_id_t I_Thread	I-Thread having the I-Event receiving notifications needs to do interrupt response operation. This member variable usually does not have to be specified by application. This member variable must be set with "R_F_OSPL_I_Thread". It is able to set the value as same as "A_Thread". The default value is R_OSPL_THREAD_NULL. If R_OSPL_THREAD_NULL was set, "R_DRIVER_OnInterrupted" function is called automatically from the inside of interrupt callback function. Otherwise a thread received I-Event must call "R_DRIVER_OnInterrupted" function. In OS-using environment, if R_OSPL_THREAD_NULL was set, the driver's specification may be to notify to I-Thread created in the driver inside. Look at the code of "R_DRIVER_SetDefaultAsync" function. Refer to : Figure of sequence
r_ospl_event_flags_t I_EventValue	The value of flag pattern set to the I-Event. This member variable usually does not have to be specified by application. This member variable must be set with "R_F_OSPL_I_EventValue". Event flags set to notify target. Flags are 16bit. Default is 0, if "I_Thread == R_OSPL_THREAD_NULL", otherwise default is 0x0002 = R_OSPL_I_FLAG. However, the default value is changed by "R_DRIVER_SetDefaultAsync" function.
r_ospl_callback_t InterruptCallback	The interrupt callback function. Specified function is callbacked from internal interrupt. This member variable usually does not have to be specified by application. This member variable must be set with "R_F_OSPL_InterruptCallback". The default value is NULL. If NULL was set, the default callback function created by the driver.

	This function will be callbacked in the interrupt context that the driver receives. But some driver sometimes calls out of the interrupt context.
errnum_t ReturnValue	The return value of the asynchronous operation. The value set at starting the asynchronous operation will be deleted. "ReturnValue" must be checked, when the asynchronous operation was ended.

4.5.7. r_ospl_async_status_t

Synopsis	Structure of driver's status and interrupt status defined by OSPL
Header	r_ospl.h
Description	"r_driver_async_status_t" type defined by the driver may have new member variable or do not have some member variables.

Member variable

r_ospl_async_state_t State	The state of the driver. e.g.) running asynchronous operation, interrupt response operation and so on. This variable must be with volatile
bool_t IsEnabledInterrupt	Whether all interrupt line related the channel is enabled or not
r_ospl_flag32_t InterruptEnables	Interrupt lines in all interrupt lines related with the channel
r_ospl_flag32_t InterruptFlags	Copy of the interrupt status register. Symbols corresponded each bit are defined in the driver internally. These symbols may not be public in some drivers.
r_ospl_flag32_t CancelFlags	Internal driver's flags of status of cancel the operation
union LockOwner	Locking owner. There is this member variable only when "R_OSPL_IS_PREEMPTION" = 0 defined.
r_ospl_thread_id_t LockOwner.Thread	Thread type locking owner. R_OSPL_THREAD_NULL=Unlocked.
void* LockOwner.Context	Context type locking owner. NULL=Unlocked.

4.5.8. r_ospl_queue_id_t

This is the type of queue that can be communicated between threads.

Each sizes of elements in a queue are all same. Element size can be different in each queue.

Member variables should not be accessed.

This is created by "R_OSPL_QUEUE_Create" function.

"r_ospl_queue_id_t" type variable can be let "R_OSPL_QUEUE_NULL".

4.5.9. r_ospl_queue_def_t

This is the type of define of queue.

Member variables should not be accessed.

This is defined by "R_OSPL_QUEUE_DEF" macro.

4.5.10. r_ospl_queue_status_t

Synopsis	Status of queue
Header	r_ospl.h

Description	This structure can be get from "R_OSPL_QUEUE_GetStatus" function.		
Member variable	int_fast32_t	UsedCount	Count of allocated element in the queue
	int_fast32_t	MaxCount	Maximum count of allocatable element in the queue

4.5.11. BSP_CFG_USER_LOCKING_TYPE

This is the type of C-lock. Lock operation does exclusive control for internal variable. Operations which denies not owner of C-lock is user's responsibility.

This manages channel of hardware and permission to use software. It cannot be used for waiting as T-lock.

If "BSP_CFG_USER_LOCKING_ENABLED" was 0, "BSP_CFG_USER_LOCKING_TYPE" is defined as "r_ospl_c_lock_t". If 1, OSPL does not define the type.

If C-lock became successful, the owner gets some right. The owner can operate something to the lock target object. If there is not the right, E_ACCESS_DENIED error is raised for example. The owner is a thread or a context. This is depended on the specification.

- When the target can be locked, set the owner ID in the lock target
- Before the target is unlocked, unset the owner ID in the lock target
- In the API of the lock target, check the owner ID

If there was other driver that operates same peripheral, OSPL may delegate locking management to the driver.

4.5.12. r_ospl_c_lock_t

This is the type of C-lock for OSPL internal. This does not manage exclusive control. See "BSP_CFG_USER_LOCKING_TYPE" for API.

"r_ospl_c_lock_t*" type variable can be let NULL.

4.5.13. r_ospl_table_t

This is the type of array index table.

Member variables should not be accessed.

This is created by "R_OSPL_TABLE_DEF" macro or "R_OSPL_TABLE_InitConst" function 4.6.1. (12) .

This is converted to array's index number from key as "void*" type or "uintptr_t" type (NULL is available). Array's index number is integer in a range between 0 and (n-1), and n can be set any number. User can select to do I-lock or T-lock in every function. "r_ospl_table_t" checks no invalid reentrant, if "R_OSPL_NDEBUG" was not defined. It raises "R_OSPL_RaiseUnrecoverable(E_STATE)" function, when invalid reentrant was detected.

This can be used for like thread local storage by converting from ID of "r_ospl_thread_id_t" to array's index number. "r_ospl_table_t" can be used like fixed length memory pool, too.

4.5.14. r_ospl_table_status _t

Outline	Structure of status of array index table.		
Header	r_ospl.h		
Description	This is set by "R_OSPL_TABLE_GetStatus" function. (4.6.13. (8))		
Member variable	int_fast32_t	Count	Count of elements in the table.
	int_fast32_t	MaxCount	Max count of elements in the table.

4.5.15. r_ospl_memory_spec_t

Outline	Structure of the specification of the memory and cache memory
Header	r_ospl.h
Description	

Member variable	uint_fast32_t CacheLineSize	Size of cache line (byte)

4.5.16. r_ospl_ftimer_spec_t

Outline	Specification of free running timer
Header	r_ospl.h
Description	This is able to get from "R_OSPL_FTIMER_InitializeIfNot" function. If the precision was 1 millisecond, "msec_Numerator" = 1 and "msec_Denominator" = 1. If the precision was 1 microsecond, "msec_Numerator" = 1 and "msec_Denominator" = 1000.

Member variable		
	uint32_t msec_Numerator	The numerator of millisecond by 1 count.
	uint32_t msec_Denominator	The denominator of millisecond by 1 count.
	uint32_t MaxCount	Max value of the count
	uint32_t ExtensionOfCount	The period from the target time. An error is raised, when "Now" argument is greater than "TargetTime" argument + "ExtensionOfCount" with "R_OSPL_FTIMER_IsPast" function.

4.5.17. r_ospl_caller_t

This is the type of managing an interrupt callback function.

Member variables should not be accessed.

4.5.18. r_ospl_interrupt_t

Outline	Structure related to interrupt source. e.g. interrupt number
Header	r_ospl.h
Description	-

Member variable		
	bsp_int_src_t IRQ_Num	Interrupt number
	int_fast32_t ChannelNum	Channel number
	int_fast32_t Type	The number defined by the driver internal
	void* Delegate	The pointer defined by the driver internal

4.6. Functions

4.6.1. List

Section	Outline
(1)	Functions for versions of OSPL
(2)	Functions for threads - r_ospl_thread_id_t type
(3)	Functions for thread attached events
(4)	Functions for flags - r_ospl_flag32_t type
(5)	Functions for bit flags - bit_flags_fast32_t type
(6)	Functions for asynchronous notification - r_ospl_async_t type
(7)	Functions for queue - r_ospl_queue_id_t type
(8)	Functions for the area disabled all interrupts
(9)	Functions for interrupt handling
(10)	Functions for BSP_CFG_USER_LOCKING_TYPE type
(11)	Functions for r_ospl_c_lock_t type
(12)	Functions for array index table - r_ospl_table_t type
(13)	Functions for the memory
(14)	Functions for time
(15)	Functions for the idle state
(16)	Functions for interrupt callback functions - r_ospl_caller_t type
(17)	Functions for error handling and debugging
(18)	Functions for reviewed tags for the static code analyzer
(19)	Multi compiler support
(20)	Functions for the layer under OSPL
(21)	Common functions for driver's APIs
(22)	Common functions under the driver

(1) Functions for versions of OSPL

Section	Function Name	Outline
4.6.2. (1)	R_OSPL_GetVersion	Returns version number of OSPL
4.6.2. (2)	R_OSPL_IsPreemption	Returns whether the environment is supported preemption

(2) Functions for threads - r_ospl_thread_id_t type

Section	Function Name	Outline
4.6.3. (1)	R_OSPL_THREAD_DEF	Defines the work area and initial value of the thread (for OS less)
4.6.3. (2)	R_OSPL_THREAD	Returns the work area of the thread (for OS less)
4.6.3. (3)	R_OSPL_THREAD_Create	Creates a pseudo thread (for OS less)
4.6.3. (4)	R_OSPL_THREAD_Destroy	Destroys a pseudo thread (for OS less)
4.6.3. (5)	R_OSPL_THREAD_GetArgument	Returns parameters passed when the running thread was created (for OS less)
4.6.3. (6)	R_OSPL_THREAD_SetCurrentId	Set running thread ID (for OS less)
4.6.3. (7)	R_OSPL_THREAD_GetCurrentId	Get running thread ID (for OS less and OS-using environment)
4.6.3. (8)	R_OSPL_THREAD_GetMainId	Returns main thread ID (for OS less)
4.6.3. (9)	R_OSPL_THREAD_SetDelegate	Sets a specified value to pointer type variable attached with thread (for OS less)

4.6.3. (10)	R_OSPL_THREAD_GetDelegate	Gets a specified value from pointer type variable attached with thread (for OS less)
4.6.3. (11)	R_OSPL_THREAD_SetOnWait	Set waiting behavior of current thread
4.6.3. (12)	R_OSPL_THREAD_GetOnWait	Get waiting behavior of current thread
4.6.3. (13)	R_OSPL_THREAD_GetIsWaiting	Get whether the current thread is waiting or not
4.6.3. (14)	R_OSPL_THREAD_ExitWaiting	Exit waiting state of current thread

(3) Functions for thread attached events

The synchronous primitive as 16bit flags attached the thread.

It notifies to a thread by set one or some flags (set bit to 1) by other threads or interrupt callback functions.

Section	Function Name	Outline
4.6.4. (1)	R_OSPL_EVENT_Allocate	Allocates a bit of thread attached event flags
4.6.4. (2)	R_OSPL_EVENT_Set	Set one or some bits to 1
4.6.4. (3)	R_OSPL_EVENT_Clear	Set one or some bits to 0
4.6.4. (4)	R_OSPL_EVENT_Wait	Waits for setting the flags in 16bit and clear received flags
4.6.4. (5)	R_OSPL_EVENT_GetStatus	Gets status of event
4.6.4. (6)	R_OSPL_EVENT_Free	Returns a bit of thread attached event flags

The reason of not using 32bit is that some OS has only 16bit and able to define time out and check bit in higher 16 bit like the following table

bit	Description
31	Reserved
30	R_OSPL_TIMEOUT
29	Reserved
28	R_OSPL_UNUSED_FLAG
27:20	Check bit - value which is changed by allocating event
19:16	Check bit - bit number of allocating event flag
15:3	Thread attached event flag
2	Thread attached event flag - R_OSPL_FINAL_A_FLAG
1	Thread attached event flag - R_OSPL_I_FLAG
0	Thread attached event flag - R_OSPL_A_FLAG

(4) Functions for flags - r_ospl_flag32_t type

The 1 level buffer of 32bit flags for notifying interrupt status and so on.

API is not atomic. It is necessary to exclusive control.

You can create any number of flags similar to "bit_flags32_t" type's integer variable.

Section	Function Name	Outline
4.6.5. (1)	R_OSPL_FLAG32_InitConst	Clears all flags in 32bit to 0
4.6.5. (2)	R_OSPL_FLAG32_Set	Set one or some bits to 1
4.6.5. (3)	R_OSPL_FLAG32_Clear	Set one or some bits to 0
4.6.5. (4)	R_OSPL_FLAG32_Get	Get 32bit flags value
4.6.5. (5)	R_OSPL_FLAG32_GetAndClear	Returns the value of flags and clears all bits to 0

(5) Functions for bit flags - bit_flags_fast32_t type

bit_flags_fast32_t type is the bit flags type of uint_fast32_t type. This type is collection of bits in binary as integer type.

Section	Function Name	Outline
4.6.6. (1)	IS_BIT_SET	Evaluate whether passed bit is 1 or not
4.6.6. (2)	IS_ANY_BITS_SET	Evaluate whether any passed bits are 1 or not
4.6.6. (3)	IS_ALL_BITS_SET	Evaluate whether all passed bits are 1 or not
4.6.6. (4)	IS_BIT_NOT_SET	Evaluate whether passed bit is 0 or not
4.6.6. (5)	IS_ANY_BITS_NOT_SET	Evaluate whether any passed bits are 0 or not
4.6.6. (6)	IS_ALL_BITS_NOT_SET	Evaluate whether all passed bits are 0 or not

(6) Functions for asynchronous notification - r_ospl_async_t type

Section	Function Name	Outline
4.6.7. (1)	R_OSPL_ASYNC_SetDefaultPreset	Sets each member variable to preset values, if preset flag was specified in "r_ospl_async_t" type structure.

(7) Functions for queue - r_ospl_queue_id_t type

Section	Function Name	Outline
4.6.8. (1)	R_OSPL_QUEUE_DEF	Defines attributes of queue and work area
4.6.8. (2)	R_OSPL_QUEUE	Returns initial attributes of queue and work area
4.6.8. (3)	R_OSPL_QUEUE_Create	Initializes a queue
4.6.8. (4)	R_OSPL_QUEUE_GetStatus	Gets status of the queue
4.6.8. (5)	R_OSPL_QUEUE_Allocate	Allocates an element from the queue object
4.6.8. (6)	R_OSPL_QUEUE_Put	Sends the element to the queue
4.6.8. (7)	R_OSPL_QUEUE_Get	Receives the element from the queue
4.6.8. (8)	R_OSPL_QUEUE_Free	Returns received memory area to the queue
4.6.8. (9)	R_OSPL_GetQueueAsSingletonLock	Returns a T-lock object that is locked when singleton was created and deleted

(8) Functions for the area disabled all interrupts

Do exclusive control to the interrupt handler by disabling or enabling all interrupts.

However, NMI is not disabled. See 4.8.3. regarding nested interrupt.

Section	Function Name	Outline
4.6.9. (1)	R_OSPL_EnableAllInterrupt	Releases all disabled interrupts
4.6.9. (2)	R_OSPL_DisableAllInterrupt	Disables all interrupts
4.6.9. (3)	R_OSPL_GetIsAllInterruptEnabled	Returns whether all interrupts are enabled

(9) Functions for interrupt handling

Section	Function Name	Outline
4.6.10. (1)	R_BSP_InterruptWrite	Registers an interrupt handler
4.6.10. (2)	R_BSP_InterruptRead	Returns registered interrupt handler

4.6.10. (3)	R_BSP_InterruptControl	Controls related to the interrupt
4.6.10. (4)	R_OSPL_SetInterruptPriority	Sets the priority of the interrupt.
4.6.10. (5)	R_OSPL_END_OF_INTERRUPT	Macro that is called at last of interrupt handler

(10) Functions for BSP_CFG_USER_LOCKING_TYPE type

Section	Function Name	Outline
4.6.11. (1)	R_OSPL_LockChannel	Locks by channel number
4.6.11. (2)	R_OSPL_UnlockChannel	Unlocks by channel number
4.6.11. (3)	R_BSP_HardwareLock	Locks by hardware identify number
4.6.11. (4)	R_BSP_HardwareUnlock	Unlocks by hardware identify number
4.6.11. (5)	R_BSP_SoftwareLock	Locks the target
4.6.11. (6)	R_BSP_SoftwareUnlock	Unlocks the target

(11) Functions for r_ospl_c_lock_t type

Section	Function Name	Outline
4.6.12. (1)	R_OSPL_C_LOCK_InitConst	Initializes the C-lock object
4.6.12. (2)	R_OSPL_C_LOCK_Lock	Locks the target, if lockable state
4.6.12. (3)	R_OSPL_C_LOCK_Unlock	Unlocks the target

(12) Functions for array index table - r_ospl_table_t type

Section	Function Name	Outline
4.6.13. (1)	R_OSPL_TABLE_DEF	Declares work area and initial value of array index table.
4.6.13. (2)	R_OSPL_TABLE	Returns work area and initial value of array index table.
4.6.13. (3)	R_OSPL_TABLE_InitConst	Initializes an array index table.
4.6.13. (4)	R_OSPL_TABLE_SIZE	Calculate size of area for the array index table.
4.6.13. (5)	R_OSPL_TABLE_GetIndex	Returns array index number from specified key.
4.6.13. (6)	R_OSPL_TABLE_Free	Remove from array index table and separate index from key. (key specified)
4.6.13. (7)	R_OSPL_TABLE_FreeByIndex	Remove from array index table and separate index from key. (index specified)
4.6.13. (8)	R_OSPL_TABLE_GetStatus	Gets (pointer to structure that describes) status of array index table.

(13) Functions for the memory

Section	Function Name	Outline
4.6.14. (1)	R_OSPL_MEMORY_Flush	Flushes cache memory
4.6.14. (2)	R_OSPL_MEMORY_RangeFlush	Flushes cache memory with the range of virtual address
4.6.14. (3)	R_OSPL_MEMORY_GetLevelOfFlush	Gets the level of cache flush for the memory
4.6.14. (4)	R_OSPL_MEMORY_GetMaxLevelOfFlush	Gets the level of all cache flush for the memory
4.6.14. (5)	R_OSPL_MEMORY_GetSpecification	Gets the specification about memory and cache memory

4.6.14. (6)	R_OSPL_ToPhysicalAddress	Changes to physical address
4.6.14. (7)	R_OSPL_ToCachedAddress	Changes to the address in the cached area
4.6.14. (8)	R_OSPL_ToUncachedAddress	Changes to the address in the uncached area
4.6.14. (9)	R_OSPL_MEMORY_Barrier	Set data memory barrier
4.6.14. (10)	R_OSPL_InstructionSyncBarrier	Set instruction synchronization barrier
4.6.14. (11)	R_OSPL_AXI_Get2ndCacheAttribute	Gets L2 cache attribute of AXI bus from the address
4.6.14. (12)	R_OSPL_AXI_GetProtection	Gets protection attribute of AXI bus from the address

(14) Functions for time

Section	Function Name	Outline
4.6.15. (1)	R_OSPL_Delay	Waits for a while until passed time
4.6.15. (2)	R_OSPL_FTIMER_InitializeIfNot	Set up the free running timer
4.6.15. (3)	R_OSPL_FTIMER_Get	Get current time of free running timer
4.6.15. (4)	R_OSPL_FTIMER_IsPast	Returns whether specified time was passed
4.6.15. (5)	R_OSPL_FTIMER_TimeToCount	Change from millisecond unit to free running timer unit
4.6.15. (6)	R_OSPL_FTIMER_CountToTime	Change from free running timer unit to millisecond unit
4.6.15. (7)	R_OSPL_FTIMER_GetSpecification	Get the specification of the free running timer

(15) Functions for the idle state

It is used for measuring CPU load.

When CPU load was measured on OS-using environment, make the lowest priority idle thread, count up the variable in the thread and refer to the count value at the cyclic timer interrupt instead of using OSPL. Calculate CPU load as 100% is defined to the value of count up when only idle thread was running. Also, there is sometimes ICE which can measure CPU load.

Section	Function Name	Outline
4.6.16. (1)	R_OSPL_IDLE_Start_CPU_Load	Starts to measure CPU load (for OS less)
4.6.16. (2)	R_OSPL_IDLE_Print_CPU_Load	Show the CPU load by printf (for OS less)

(16) Functions for interrupt callback functions - r_ospl_caller_t type

Section	Function Name	Outline
4.6.17. (1)	R_OSPL_CallInterruptCallback	Calls the interrupt callback function. It is called from OS porting layer in the driver
4.6.17. (2)	r_ospl_callback_t	The function type of interrupt callback

(17) Functions for error handling and debugging

Section	Function Name	Outline
4.6.18. (1)	CHK	Enters infinite loop, if error was raised. It is for debugging
4.6.18. (2)	R_OSPL_RaiseUnrecoverable	Raises the error of system unrecoverable
4.6.18. (3)	R_DEBUG_BREAK	Breaks here
4.6.18. (4)	R_DEBUG_BREAK_IF_ERROR	Breaks here, if it is error state

4.6.18. (5)	IF	Breaks and transits to error state, if condition expression is not 0
4.6.18. (6)	IF_D	It is same as "IF" (for Debug configuration only)
4.6.18. (7)	ASSERT_R	Assertion (Programming by Contract)
4.6.18. (8)	ASSERT_D	Assertion (Programming by Contract) (for Debug configuration only)
4.6.18. (9)	R_STATIC_ASSERT	Assertion (Programming by Contract). It raises compiling error, if static condition was false
4.6.18. (10)	R_STATIC_ASSERT_GLOBAL	"R_STATIC_ASSERT" for global scope
4.6.18. (11)	R_NOOP	The function doing nothing
4.6.18. (12)	R_OSPL_MergeErrNum	Merge the error code raised in the finalizing operation
4.6.18. (13)	R_OSPL_SetErrNum	Sets the error code to TLS
4.6.18. (14)	R_OSPL_GetErrNum	Returns the error code from TLS
4.6.18. (15)	R_OSPL_CLEAR_ERROR	Clears the error state
4.6.18. (16)	R_OSPL_NOTIFY_ERROR	Notifies an error to another thread and clears an error of current thread.
4.6.18. (17)	R_OSPL_SET_BREAK_ERROR_ID	Register to break at raising error at the moment
4.6.18. (18)	R_OSPL_GET_ERROR_ID	Returns the number of current error
4.6.18. (19)	R_OSPL_DEBUG_WORK_SIZE	Calculates the size of debug work area
4.6.18. (20)	R_OSPL_GetCurrentThreadError	Returns debug information of current thread
4.6.18. (21)	R_OSPL_FreeCurrentThreadError	Releases an area of error and debug information in current thread.
4.6.18. (22)	R_OSPL_CHANGE_THREAD_LOCKED_COUNT	Modifies value of counter that can be accessed by current thread only
4.6.18. (23)	R_OSPL_GET_THREAD_LOCKED_COUNT	Returns value of counter that can be accessed by current thread only
4.6.18. (24)	R_OSPL_GET_THREAD_LOCKED_COUNT	Returns value of current stack pointer for debugging.
4.6.18. (25)	R_OSPL_SET_END_OF_STACK	Sets the end of stack area of current thread for debugging.
4.6.18. (26)	R_OSPL_MOVE_END_OF_STACK	Moves end of stack area
4.6.18. (27)	R_OSPL_CHECK_STACK_OVERFLOW	Checks that stack overflow occurred.
4.6.18. (28)	R_OSPL_RESET_MIN_FREE_STACK_SIZE	Resets minimum free stack size in current thread.
4.6.18. (29)	R_OSPL_GET_MIN_FREE_STACK_SIZE	Counts minimum free stack size in current thread.
4.6.18. (30)	R_OSPL_GET_MIN_STACK_POINTER	Searches the position of stack pointer when largest stack area was used by now
4.6.18. (31)	R_D_Add	Registers watching integer variable or pointer variable
4.6.18. (32)	R_D_Watch	Show and Check watching variable's value
4.6.18. (33)	R_D_AddToIntLog	Records to the log fast
4.6.18. (34)	R_D_Counter	Count the through count

(18) Functions for reviewed tags for the static code analyzer

Section	Function Name	Outline
4.6.19. (1)	IS	Changes the code accepted with MISRA 13.2 to readable
4.6.19. (2)	R_OSPL_ReturnFalse	Countermeasure for the warning, when "if" block was disabled.
4.6.19. (3)	R_UNREFERENCED_VARIABLE	Suppress the warning of not referenced variable

4.6.19. (4)	R_UNREFERENCED_VARIABLE_2	"R_UNREFERENCED_VARIABLE" with 2 arguments
4.6.19. (5)	R_UNREFERENCED_VARIABLE_3	"R_UNREFERENCED_VARIABLE" with 3 arguments
4.6.19. (6)	R_UNREFERENCED_VARIABLE_4	"R_UNREFERENCED_VARIABLE" with 4 arguments

(19) Multi compiler support

Section	Macro Name	Outline
4.6.20. (1)	R_OSPL_SECTION	Names section name to function or variable
4.6.20. (2)	R_OSPL_ALIGNMENT	Alignments first address of global variable
4.6.20. (3)	R_COUNT_OF	Returns element count of the array
4.6.20. (4)	INLINE	Inline function (C99 specification compatible)
4.6.20. (5)	STATIC_INLINE	Static inline function (C99 specification compatible)
4.6.20. (6)	R_ADDRESS_Add	Adds at a value of address (pointer) by byte unit.
4.6.20. (7)	R_OSPL_CountLeadingZeros	Counts bits which is set 0 from most significant bit (MSB).
4.6.20. (8)	R_OSPL_IsSetBitsCount1	Returns whether there is one bit which is set 1.

(20) Functions for the layer under OSPL

Section	Function Name	Outline
4.6.21. (1)	R_DebugBreak	The function callbacked from OSPL for breaking
4.6.21. (2)	R_OSPL_OnIdleDefault	The default callback function on idle state (for OS less)
4.6.21. (3)	R_OSPL_Start_T_Lock	The function callbacked from OSPL internal, when T-Lock started
4.6.21. (4)	R_OSPL_End_T_Lock	The function callbacked from OSPL internal, when T-Lock ended
4.6.21. (5)	R_OSPL_EVENT_GROUP_Create	Creates an event group
4.6.21. (6)	R_OSPL_EVENT_GROUP_Delete	Delete an event group

(21) Common functions for driver's APIs

Section	Function Name	Outline
4.6.22. (1)	R_DRIVER_Transfer	Does synchronously the asynchronous operation of the peripheral function
4.6.22. (2)	R_DRIVER_TransferStart	Starts the asynchronous operation of the peripheral function
4.6.22. (3)	R_DRIVER_OnInterrupting	Receives the interrupt
4.6.22. (4)	R_DRIVER_OnInterrupted	Does the interrupt response operation
4.6.22. (5)	R_DRIVER_GetAsyncStatus	Get the pointer to the structure indicated the status of interrupts and the asynchronous operation
4.6.22. (6)	R_DRIVER_Initialize	Initializes the driver and changes driver's state to usable
4.6.22. (7)	R_DRIVER_Finalize	Finalizes the driver
4.6.22. (8)	R_DRIVER_LockChannel	Locks a channel (Changes to used state in the driver)
4.6.22. (9)	R_DRIVER_UnlockChannel	Unlocks a channel (Changes to not used state in the driver)

(22) Common functions under the driver

Section	Function Name	Outline
4.6.23. (1)	R_DRIVER_SetDefaultAsync	Sets default value in "r_ospl_async_t" type
4.6.23. (2)	R_DRIVER_I_LOCK_Replace	Replaces I-lock object to the integrated driver's I-lock object
4.6.23. (3)	R_DRIVER_DisableInterrupt	Disables an interrupt for staring I-lock
4.6.23. (4)	R_DRIVER_EnableInterrupt	Enables an interrupt for ending I-lock

4.6.2. Functions for versions of OSPL

(1) R_OSPL_GetVersion

Outline	Returns version number of OSPL	
Header	r_ospl.h	
Declaration	int32_t R_OSPL_GetVersion();	
Description	Return value is same as "R_OSPL_VERSION" macro.	
Arguments	None	
Return value	Version number of OSPL	

(2) R_OSPL_IsPreemption

Outline	Returns whether the environment is supported preemption	
Header	r_ospl.h	
Declaration	bool_t R_OSPL_IsPreemption();	
Description	Return value is same as "R_OSPL_IS_PREEMPTION" macro.	
Arguments	None	
Return value	Whether the environment is RTOS supported preemption.	

4.6.3. Functions for threads - r_ospl_thread_id_t type

(1) R_OSPL_THREAD_DEF

Outline	Defines the work area and initial value of the thread (for OS less)	
Header	r_ospl.h	
Declaration	#define R_OSPL_THREAD_DEF(Name)	
Description	There is this function only when "R_OSPL_IS_PREEMPTION" = 0 defined. It is not possible to use this macro in the library. If implement of OSPL was changed, the library must be recompiled.	
Arguments	Name	Thread name. Do not bracket by ""
Return value	None	

(2) R_OSPL_THREAD

Outline	Returns the work area of the thread (for OS less)	
Header	r_ospl.h	
Declaration	r_ospl_thread_def_t* R_OSPL_THREAD(Name);	
Description	There is this function only when "R_OSPL_IS_PREEMPTION" = 0 defined. It is not possible to use this macro in the library. If implement of OSPL was changed, the library must be recompiled.	
Arguments	Name	Thread name defined by "R_OSPL_THREAD_DEF". Do not bracket by ""
Return value	Work area of the thread and initial value	

(3) R_OSPL_THREAD_Create

Outline	Creates a pseudo thread (for OS less)
Header	r_ospl.h
Declaration	errnum_t R_OSPL_THREAD_Create(r_ospl_thread_def_t* ThreadDef, void* Argument, r_ospl_thread_id_t* out_ThreadId);
Description	There is this function only when "R_OSPL_IS_PREEMPTION" = 0 defined. Usually, use it for creating more than one event. To delete a thread, please call "R_OSPL_THREAD_Destroy" function.

Example:

```
r_ospl_thread_id_t  g_ThreadA_Id;
R_OSPL_THREAD_DEF( g_ThreadA_Def );

void  main()
{
    errnum_t  e;

    e= R_OSPL_THREAD_Create( R_OSPL_THREAD( g_ThreadA_Def ),
        NULL, &g_ThreadA_Id );
        IF(e){goto fin;}
}
```

Arguments	r_ospl_thread_def_t* ThreadDef	Work area of the thread and initial value returned by "R_OSPL_THREAD" function.
	void* Argument	The parameter passing to "R_OSPL_THREAD_GetArgument" function.
	r_ospl_thread_id_t* out_ThreadId	Output: Thread ID
Return value	Error code. If there is no error, the return value is 0.	

(4) R_OSPL_THREAD_Destroy

Outline	Destroys a pseudo thread (for OS less)
Header	r_ospl.h
Declaration	errnum_t R_OSPL_THREAD_Destroy(r_ospl_thread_id_t* in_out_ThreadId);
Description	There is this function only when "R_OSPL_IS_PREEMPTION" = 0 defined. This does nothing and returns 0 (no error), if thread ID specified by "in_out_ThreadId" argument was "R_OSPL_THREAD_NULL". Attention: "R_OSPL_FreeCurrentThreadError" must be called at the end of the thread.

Arguments	r_ospl_thread_id_t* in_out_ThreadId	ID of deleting thread
Return value	Error code. If there is no error, the return value is 0.	

(5) R_OSPL_THREAD_GetArgument

Outline	Returns parameters passed when the running thread was created (for OS less)
Header	r_ospl.h
Declaration	void* R_OSPL_THREAD_GetArgument();
Description	There is this function only when "R_OSPL_IS_PREEMPTION" = 0 defined. Call "R_OSPL_THREAD_SetCurrentId" function from the message loop before calling this function.

Arguments	None	
Return value	"Argument" argument of "R_OSPL_THREAD_Create" function	

(6) R_OSPL_THREAD_SetCurrentId

Outline	Set running thread ID (for OS less)
Header	r_ospl.h
Declaration	void R_OSPL_THREAD_SetCurrentId(r_ospl_thread_id_t ThreadId);
Description	There is this function only when "R_OSPL_IS_PREEMPTION" = 0 defined. Call this function from the message loop on the pseudo multi-thread environment. Example: `R_OSPL_THREAD_SetCurrentId( ChildThreadId );`

```
        ChildThreadFunction();
        R_OSPL_THREAD_SetCurrentId( MainThreadId );
```

Arguments	r_ospl_thread_id_t ThreadId	Thread ID setting to current running state.
Return value	None	

(7) R_OSPL_THREAD_GetCurrentId

Outline	Get running thread ID (for OS less and OS-using environment)	
Header	r_ospl.h	
Declaration	r_ospl_thread_id_t R_OSPL_THREAD_GetCurrentId();	
Description	It is possible to use this function, even if "R_OSPL_IS_PREEMPTION" was any define. For OS less, returns thread ID passed to "R_OSPL_THREAD_SetCurrentId" function. "R_OSPL_THREAD_NULL" is returned, if in interrupt context.	
Arguments	None	
Return value	The current running thread ID	

(8) R_OSPL_THREAD_GetMainId

Outline	Returns main thread ID (for OS less)	
Header	r_ospl.h	
Declaration	r_ospl_thread_id_t R_OSPL_THREAD_GetMainId();	
Description	There is this function only when "R_OSPL_IS_PREEMPTION" = 0 defined. Call "R_OSPL_THREAD_SetCurrentId" function passed main thread ID at the first of "main" function.	
Arguments	None	
Return value	Main thread ID	

(9) R_OSPL_THREAD_SetDelegate

Outline	Sets a specified value to pointer type variable attached with thread (for OS less)	
Header	r_ospl.h	
Declaration	void R_OSPL_THREAD_SetDelegate(r_ospl_thread_id_t in_ThreadId, void* in_Delegate);	
Description	The variable attached with thread is not accessed from OSPL except for initializing the thread.	
Arguments	r_ospl_thread_id_t in_ThreadId	Thread ID
	void* in_Delegate	Setting value
Return value	None	

(10) R_OSPL_THREAD_GetDelegate

Outline	Gets a specified value from pointer type variable attached with thread (for OS less)	
Header	r_ospl.h	
Declaration	void* R_OSPL_THREAD_GetDelegate(r_ospl_thread_id_t in_ThreadId);	
Description		
Arguments	r_ospl_thread_id_t in_ThreadId	Thread ID
Return value	Value of thread attached variable. Initial value is NULL.	

(11) R_OSPL_THREAD_SetOnWait

Outline	Set waiting behavior of current thread.
Header	r_ospl.h

Declaration	errnum_t R_OSPL_THREAD_SetOnWait(r_ospl_wait_t OnWait);
Description	This function should not be called from any drivers. Application can call this function. Set "R_OSPL_WAIT_PM_THREAD" by this function to change from code for multi-thread (OS using) to code for pseudo multi-thread (OS less). And change code of time out from R_OSPL_INFINITE (OS using) to 0 (OS less). It is not necessary to change code of time out from not R_OSPL_INFINITE (OS using). Case of "R_OSPL_IS_PREEMPTION" = 0: Initial value is "R_OSPL_WAIT_POLLING". If this function was called from the interrupt context, E_STATE error is raised. Case of "R_OSPL_IS_PREEMPTION" = 1: This function is for compatibility only. Arguments are ignored. This function returns 0. Refer to: R_OSPL_THREAD_GetIsWaiting

Arguments	r_ospl_wait_t OnWait	Behavior on waiting
Return value	Error code. If there is no error, the return value is 0.	

(12) R_OSPL_THREAD_GetOnWait

Outline	Get waiting behavior of current thread.
Header	r_ospl.h
Declaration	r_ospl_wait_t R_OSPL_THREAD_GetOnWait();
Description	Case of "R_OSPL_IS_PREEMPTION" = 0: Initial value is "R_OSPL_WAIT_POLLING". If this function was called from the interrupt context, this function returns "R_OSPL_WAIT_POLLING". Case of "R_OSPL_IS_PREEMPTION" = 1: This function is for compatibility only. This function returns R_OSPL_WAIT_POLLING. But it does not do the polling on waiting Refer to: R_OSPL_THREAD_SetOnWait

Arguments	None	
Return value	Behavior on waiting	

(13) R_OSPL_THREAD_GetIsWaiting

Outline	Get whether the current thread is waiting or not.
Header	r_ospl.h
Declaration	bool_t R_OSPL_THREAD_GetIsWaiting();
Description	Case of "R_OSPL_IS_PREEMPTION" = 0: If "R_OSPL_WAIT_PM_THREAD" was set by "R_OSPL_THREAD_SetOnWait" function, some waiting functions return soon even if the state is waiting and "R_OSPL_THREAD_GetIsWaiting" function returns true. If time out was set to 0, "R_OSPL_THREAD_GetIsWaiting" function returns true at time out, even if any value was passed to "R_OSPL_THREAD_SetOnWait" function, "R_OSPL_THREAD_GetIsWaiting" function cannot be called because of not returning from waiting function, if time out was "R_OSPL_INFINITE". Replace from R_OSPL_INFINITE to 0 for porting to pseudo multi thread environment. If this function was called from the interrupt context, this function returns false and "ASSERT_D" in this function notifies in debug configuration.

Case of "R_OSPL_IS_PREEMPTION" = 1:
This function is for compatibility only.
This function returns false.

Example:
```
e= R_OSPL_THREAD_SetOnWait( R_OSPL_WAIT_PM_THREAD );
     IF(e){goto fin;}
e= R_OSPL_Delay( 100 ); IF(e){goto fin;}
    if ( R_OSPL_THREAD_GetIsWaiting() ) { e=0; goto fin; }
    IF(e){goto fin;}
```

Arguments	None	
Return value	Whether the current thread is waiting or not	

(14) R_OSPL_THREAD_ExitWaiting

Outline	Exit waiting state, if current thread was waiting state.
Header	r_ospl.h
Declaration	void R_OSPL_THREAD_ExitWaiting();
Description	Case of "R_OSPL_IS_PREEMPTION" = 0:

The thread returned true from "R_OSPL_THREAD_GetIsWaiting" function after a waiting function must call the waiting function again. After exiting waiting state, other operation and other waiting can be done. If time out was set to 0, exiting does not have to do.

If it was detected in OSPL API that necessary exiting was not done, E_STATE error is raised. However sometimes the state cannot be detected. In this case, time out will be not correct.

This function does not do anything called from the interrupt context. "ASSERT_D" in this function notifies in debug configuration.

Case of "R_OSPL_IS_PREEMPTION" = 1:
This function is for compatibility only.
This function does not do anything.

Refer to: R_OSPL_THREAD_SetOnWait

Arguments	None	
Return value	None	

4.6.4. Functions for thread attached events

(1) R_OSPL_EVENT_Allocate

Outline	Allocates a bit of thread attached event flags
Header	r_ospl.h
Declaration	errnum_t R_OSPL_EVENT_Allocate(r_ospl_thread_id_t* out_ThreadId, r_ospl_thread_id_t in_ThreadId, bit_flags32_t* out_SetFlag, r_ospl_event_flags_t in_SetFlag);
Description	This function allocates an event flag for setting and waiting an event. Only current thread can receive allocated event. Any thread or interrupt context can send, if they had check bit output from this function. Return allocated event flag by "R_OSPL_EVENT_Free" function.

Call this function, if thread attached event was used in synchronous (blocking) function in library.

Allocating and returning is usually executed in blocking function out of non-blocking function.

There is not check bit in "out_SetFlag" argument , if "R_OSPL_DETECT_BAD_EVENT= 0".

Return the event and reallocate again before waiting for not same kind of event. Check bit has ID that is changed value by each allocating event and disabled by returning event. Raised error, if sender was sent returned old ID. ID is counted up from 1 to 255 by using global variable.

"E_ACCESS_DENIED" error is raised, if allocating event was already signal state.

One bit is allocated within not allocated event flag bits dynamically, if "in_SetFlag" argument was set to "R_OSPL_UNUSED_FLAG". "R_OSPL_UNUSED_FLAG" cannot be specified, if "R_OSPL_TLS_EVENT_CODE = 1" only.

This function allocates one bit within 12bit (bit 15 to 4) except lower 4bit (e.g. R_OSPL_A_FLAG), if "R_OSPL_UNUSED_FLAG" was specified.

All bits are allocated, if "R_OSPL_EVENT_ALL_BITS" was set to "in_SetFlag" argument. Current thread cannot be find duplicated flags, but it is necessary to allocate each bit.
"E_ACCESS_DENIED" error is raised, even if only one bit was already allocated.
"*out_SetFlag" does not have check bits. It is not possible to specify "R_OSPL_EVENT_ALL_BITS", if "R_OSPL_ALL_EVENT_ALLOCATE != 1".

Arguments	r_ospl_thread_id_t* out_ThreadId	Output: ID of thread (= current thread) that is notify target from event source
	r_ospl_thread_id_t in_ThreadId	ID of thread (= current thread) that is notify target from event source
	bit_flags32_t* out_SetFlag	Output: allocated bit + check bit
	r_ospl_event_flags_t in_SetFlag	Allocating bit of thread attached event or "R_OSPL_UNUSED_FLAG"
Return value	Error code. If there is no error, the return value is 0.	

(2) R_OSPL_EVENT_Set

Outline	Set one or some bits to 1
Header	r_ospl.h
Declaration	void R_OSPL_EVENT_Set(r_ospl_thread_id_t in_ThreadId, r_ospl_event_flags_t in_SetFlags);
Description	For OS less, there is the area disabled all interrupts. For OS-using environment, the thread waiting in "R_OSPL_EVENT_Wait" function might wake up soon. Do nothing, when "in_ThreadId = R_OSPL_THREAD_NULL" "R_OSPL_RaiseUnrecoverable(E_OTHERS)" is called, if "in_SetFlags" argument did not have check bit and "R_OSPL_DETECT_BAD_EVENT= 1". Check bit can be gotten from "R_OSPL_EVENT_Allocate".

Arguments	r_ospl_thread_id_t in_ThreadId	The thread ID attached the target event source
	r_ospl_event_flags_t in_SetFlags	The value of bit flags that target bit is 1
Return value	None	

(3) R_OSPL_EVENT_Clear

Outline	Set one or some bits to 0
Header	r_ospl.h
Declaration	void R_OSPL_EVENT_Clear(r_ospl_thread_id_t ThreadId, r_ospl_event_flags_t ClearFlags1);
Description	It is not necessary to call this function after called "R_OSPL_EVENT_Wait" function. The way that all bit flags is cleared is setting "R_OSPL_EVENT_ALL_BITS" (=0x0000FFFF) at "ClearFlags1" argument. When other thread was notified by calling "R_OSPL_EVENT_Set", "R_OSPL_EVENT_Clear" must not be called from caller (notifier) thread. For OS less, there is the area disabled all interrupts. Do nothing, when "ThreadId = R_OSPL_THREAD_NULL" or "ClearFlags1" argument is 0. Thread attached event specified at "ClearFlags1" argument does not have to be allocated by "R_OSPL_EVENT_Allocate" function.

Arguments	r_ospl_thread_id_t ThreadId	The thread ID attached the target event source
	r_ospl_event_flags_t ClearFlags1	The value of bit flags that clearing bit is 1
Return value	None	

(4) R_OSPL_EVENT_Wait

Outline	Waits for setting the flags in 16bit and clear received flags. Or waits for signal state and return to not signal state after receiving.
Header	r_ospl.h
Declaration	errnum_t R_OSPL_EVENT_Wait(r_ospl_event_flags_t in_WaitingFlags, bit_flags32_t* out_GotFlags, uint32_t in_Timeout_msec);

Description	Waits in this function until the event flags become passed flags pattern by "R_OSPL_EVENT_Set" function. Received flags are cleared.
	
	In OS-using environment, the other threads can run while a thread waits in "R_OSPL_EVENT_Wait" function. In OS less, the other threads cannot run while a thread waits in "R_OSPL_EVENT_Wait" function. If "in_Timeout_msec" argument was passed 0, pseudo multi-threading can be done.
	
	Check "r_ospl_async_t::ReturnValue", when the asynchronous operation was ended.
	
	Output value of "*out_GotFlags" does not have values corresponding flags not specified at "WaitingFlags".
	
	Example:
	`errnum_t e;`
	`errnum_t ee;`
	`r_ospl_async_t async;`
	
	`async.A_EventValue = 0; /* For fin block */`

```
    async.Flags = R_F_OSPL_A_Thread | R_F_OSPL_A_EventValue;
    e= R_OSPL_EVENT_Allocate(
       &async.A_Thread,  R_OSPL_THREAD_GetCurrentId(),
       &async.A_EventValue,  R_OSPL_UNUSED_FLAG );
       IF(e){goto fin;}

    e= R_DRIVER_TransferStart( &async ); IF(e){goto fin;}

    for (;;) {
       bit_flags32_t  got_flags;

       async.ReturnValue = 0;
       R_OSPL_EVENT_Set( async.A_Thread,
          async.A_EventValue );
           /* It is called from interrupt context *
           /* or other thread. */

       e= R_OSPL_EVENT_Wait( async.A_EventValue, &got_flags,
          R_OSPL_INFINITE );
          IF(e){goto fin;}

       if ( IS_BIT_SET( got_flags, async.A_EventValue ) ) {
          e = async.ReturnValue; IF(e){goto fin;}
            :
          /* Write a response operation here */
       }
       if ( IS_BIT_SET( got_flags, async_B.A_EventValue ) ) {
          e = async_B.ReturnValue; IF(e){goto fin;}
            :
          /* Write a response operation here */
       }
    }

    e=0;
fin:
    ee= R_OSPL_EVENT_Free( &async.A_Thread,
       &async.A_EventValue );
       e= R_OSPL_MergeErrNum( e, ee );
    ee= R_OSPL_EVENT_Free( &async_B.A_Thread,
       &async_B.A_EventValue );
       e= R_OSPL_MergeErrNum( e, ee );
    return  e;
}
```

OSPL recommends using event notification as finishing asynchronous operation. If callback function was used, the context (thread or interrupt) is unknown or depended on specification. Then a problem of exclusive control may be happened. An event notification makes that data of cooperation can be wrapped in local variable of thread function. Then it is usually necessary to writing code of exclusive control. Also, real-time performance, because code on interrupt context makes be reduced. Only high priority code in whole system should be moved to callback function.

Call "R_DRIVER_OnInterrupted", when I-thread called "R_OSPL_EVENT_Wait" function and received the timing of doing the interrupt response operation. Message loop can be exited, if the state of "r_ospl_async_state_t" type got by "R_DRIVER_GetAsyncStatus" function was "R_OSPL_UNINITIALIZED".

In OS less, there is the area of disabled all interrupts inside. The area of waiting inside is out of the area of disabled all interrupts. If "R_OSPL_EVENT_Wait" was called in the area of disabled all interrupts, an error may be raised.

In OS less, the internal function of measuring CPU load is callbacked many times while waiting.

About "in_WaitingFlags" argument:
This argument can be passed waiting 16bit flags and following constant value.

Symbol	Value	Description
R_OSPL_ANY_FLAG	0x00000000	Any flags in all flags

About time out and "in_Timeout_msec" argument:

"in_Timeout_msec" argument can be set not only time out but also the following symbol.

Symbol	Value	Description
R_OSPL_INFINITE	0xFFFFFFFF	No timeout. Current thread waits infinitely until received. This value is depended on the environment.
R_OSPL_INFINITE_PSEUDO	0xFFFFFFFF or 0	"R_OSPL_INFINITE" that is supported with pseudo multi thread. R_OSPL_IS_PREEMPTION = 1 : Current thread waits infinitely until received. R_OSPL_IS_PREEMPTION = 0 : Current thread does not wait (like in_Timeout_msec = 0), even if the thread was waiting state. But, it is necessary to branch to execute another thread by return value of "R_OSPL_THREAD_GetIsWaiting".

"R_OSPL_TIMEOUT" bit in "*out_GotFlags" is set to 1, when time out occurred, and 16bit flags bits are set to 0 despite flag bit value.

Symbol	Value	Description
R_OSPL_TIMEOUT	0x40000000	Time out occurred

In OS less, the other threads cannot run while a thread waits in "R_OSPL_EVENT_Wait" function. If "in_Timeout_msec" argument was passed 0, pseudo multi-threading can be done.
At the case of timeout, "R_OSPL_TIMEOUT" bit in "*out_GotFlags" becomes to 1 and 16bit flags becomes to 0, if real value of flag was any value. Return value is 0 if "in_Timeout_msec = 0", otherwise "E_TIME_OUT".

"R_OSPL_EVENT_Wait" function returns soon even if waiting state after called "R_OSPL_THREAD_SetOnWait" function with "R_OSPL_WAIT_PM_THREAD" on "R_OSPL_IS_PREEMPTION" = 0. In this case, the return value of "R_OSPL_EVENT_Wait" function is 0 and *out_GotFlags argument is 0x00000000.

Regarding "in_WaitingFlags" argument:

If you want to wait for OR condition, set 0 or "R_OSPL_ANY_FLAG" to "in_WaitingFlags" argument. It is necessary to respond the operation for not waiting flags, too, because waiting bits cannot be set.

If you want to wait for AND condition, set not 0 and not "R_OSPL_ANY_FLAG" to "in_WaitingFlags" argument.

If you want to wait for complex condition of AND and OR, set 0 or "R_OSPL_ANY_FLAG" to "in_WaitingFlags" argument. When the process returned from "R_OSPL_EVENT_Wait" function, send the flags to an automatic variable of "r_ospl_flag32_t" type and check the value of "bit_flags_fast32_t" type got by *out_GotFlags argument. If the condition was not met, call "R_OSPL_EVENT_Wait" function again.

Arguments	r_ospl_event_flags_t in_WaitingFlags	The bit flags set to 1 waiting for (AND condition) or "R_OSPL_ANY_FLAG"
	bit_flags32_t* out_GotFlags	NULL is permitted. Output: 16bit flags or "R_OSPL_TIMEOUT"
	uint32_t in_Timeout_msec	Time out (millisecond) or "R_OSPL_INFINITE"
Return value	Error code. If there is no error, the return value is 0.	

(5) R_OSPL_EVENT_GetStatus

Outline	Gets status of event.
Header	r_ospl.h
Declaration	errnum_t R_OSPL_EVENT_GetStatus(r_ospl_thread_id_t in_ThreadId, r_ospl_event_status_t** out_Status);

Description	There is this function, if "R_OSPL_DETECT_BAD_EVENT = 1".
	Status are momentary information, when it was gotten. They are modified by other thread running.
	
	Example:
	`r_ospl_event_status_t* status;`
	`e= R_OSPL_EVENT_GetStatus( R_OSPL_THREAD_GetCurrentId(), &status ); IF(e){goto fin;}`

Arguments	r_ospl_thread_id_t in_ThreadId	ID of thread having target event
	r_ospl_event_status_t** out_Status	Output: pointer to internal event structure. See 4.5.5.
Return value	Error code. If there is no error, the return value is 0.	

(6) R_OSPL_EVENT_Free

Outline	Returns a bit of thread attached event flags which current thread received
Header	r_ospl.h
Declaration	errnum_t R_OSPL_EVENT_Free(r_ospl_thread_id_t* in_out_ThreadId, r_ospl_event_flags_t* in_out_SetFlag);
Description	"*in_out_ThreadId" argument and "*in_out_SetFlag" argument is returned to the value specified at "in_ThreadId" argument and "in_SetFlag" argument. Returned event cannot be set and received. Event allocated by current thread only can be returned.

This function must be called for detecting conflict of flags by error, if thread attached event was used in library (binary) in synchronized function.

This function does nothing, if "*in_out_SetFlag" argument was set to 0.
Returned flags are cleared in this function.

Arguments	r_ospl_thread_id_t* in_out_ThreadId	Input/Output: ID of notify target thread
	r_ospl_event_flags_t* in_out_SetFlag	Input/Output: Returning notify target (+ check bit)
Return value	Error code. If there is no error, the return value is 0.	

4.6.5. Functions for flags - r_ospl_flag32_t type

(1) R_OSPL_FLAG32_InitConst

Outline	Clears all flags in 32bit to 0	
Header	r_ospl.h	
Declaration	void R_OSPL_FLAG32_InitConst(r_ospl_flag32_t* self);	

Description	This function is not thread safe. Please do exclusive control, if it needs.
	Operates following operation.
	`volatile bit_flags32_t self->flags;`
	`self->flags = 0;`	

Arguments	r_ospl_flag32_t* self	The value of 32bit flags
Return value	None	

(2) R_OSPL_FLAG32_Set

Outline	Set one or some bits to 1	
Header	r_ospl.h	
Declaration	void R_OSPL_FLAG32_Set(r_ospl_flag32_t* self, bit_flags32_t SetFlags);	

Description	This function is not thread safe. Please do exclusive control, if it needs.
	Operates following operation.
	This function is not atomic because "|=" operator is "Read Modify Write" operation.
	`volatile bit_flags32_t self->Flags;`
	`bit_flags32_t SetFlags;`
	`self->Flags |= SetFlags;`	

Arguments	r_ospl_flag32_t* self	The value of 32bit flags
	uint32_t SetFlags	The value of bit flags that target bit is 1
Return value	None	

(3) R_OSPL_FLAG32_Clear

Outline	Set one or some bits to 0	
Header	r_ospl.h	
Declaration	void R_OSPL_FLAG32_Clear(r_ospl_flag32_t* self, bit_flags32_t ClearFlags1);	

Description	This function is not thread safe. Please do exclusive control, if it needs.
	Operates following operation.
	Set "R_OSPL_FLAG32_ALL_BITS", if you wanted to clear all bits.
	This function is not atomic because "&=" operator is "Read Modify Write" operation.
	`volatile bit_flags32_t self->Flags;`
	`bit_flags32_t ClearFlags1;`	

```
self->Flags &= ~ClearFlags1;
```

Arguments	r_ospl_flag32_t* self	The value of 32bit flags
	uint32_t ClearFlags1	The value of bit flags that clearing bit is 1
Return value	None	

(4) R_OSPL_FLAG32_Get

Outline	Get 32bit flags value
Header	r_ospl.h
Declaration	bit_flags32_t R_OSPL_FLAG32_Get(r_ospl_flag32_t* self);
Description	This function is not thread safe. Please do exclusive control, if it needs. In receiving the event, call "R_OSPL_FLAG32_GetAndClear" function instead of "R_OSPL_FLAG32_Get" function or call "R_OSPL_FLAG32_Clear" function passed the NOT operated value of flags got by "R_OSPL_FLAG32_Get" function. Operates following operation.

```
volatile bit_flags32_t  self->Flags;
bit_flags32_t          return_flags;

return_flags = self->Flags;

return  return_flags;
```

Arguments	r_ospl_flag32_t* self	The value of 32bit flags
Return value	The value of 32bit flags	

(5) R_OSPL_FLAG32_GetAndClear

Outline	Returns the value of flags and clears all bits to 0
Header	r_ospl.h
Declaration	bit_flags32_t R_OSPL_FLAG32_GetAndClear(r_ospl_flag32_t* self);
Description	This function is not thread safe. Please do exclusive control, if it needs. Operates following operation. This function is not atomic because the value might be set before clearing to 0.

```
volatile bit_flags32_t  self->Flags;
bit_flags32_t          return_flags;

return_flags = self->Flags;
self->Flags = 0;

return  return_flags;
```

Arguments	r_ospl_flag32_t* self	The value of 32bit flags
Return value	The value of 32bit flags	

4.6.6. Functions for bit flags - bit_flags_fast32_t type

(1) IS_BIT_SET

Outline	Evaluate whether passed bit is 1 or not
Header	r_ospl.h
Declaration	bool_t IS_BIT_SET(bit_flags_fast32_t Variable, bit_flags_fast32_t ConstValue);
Description	If the count of bit in "ConstValue" argument that the value is 1 was not 1, return value is unspecified.

Arguments	bit_flags_fast32_t Variable	The value of target bit flags

	bit_flags_fast32_t ConstValue	The value that investigating bit is 1
Return value	Whether passed bit is 1	

(2) IS_ANY_BITS_SET

Outline	Evaluate whether any passed bits are 1 or not	
Header	r_ospl.h	
Declaration	bool_t IS_ANY_BITS_SET(bit_flags_fast32_t Variable, bit_flags_fast32_t ConstValue);	
Description	If the count of bit in "ConstValue" argument that the value is 1 was 0, return value is unspecified.	
Arguments	bit_flags_fast32_t Variable	The value of target bit flags
	bit_flags_fast32_t ConstValue	The value that investigating bits are 1
Return value	Whether the any passed bits are 1	

(3) IS_ALL_BITS_SET

Outline	Evaluate whether all passed bits are 1 or not	
Header	r_ospl.h	
Declaration	bool_t IS_ALL_BITS_SET(bit_flags_fast32_t Variable, bit_flags_fast32_t ConstValue);	
Description	If the count of bit in "ConstValue" argument that the value is 1 was 0, return value is unspecified.	
Arguments	bit_flags_fast32_t Variable	The value of target bit flags
	bit_flags_fast32_t ConstValue	The value that investigating bits are 1
Return value	Whether the all passed bit are 1	

(4) IS_BIT_NOT_SET

Outline	Evaluate whether passed bit is 0 or not	
Header	r_ospl.h	
Declaration	bool_t IS_BIT_NOT_SET(bit_flags_fast32_t Variable, bit_flags_fast32_t ConstValue);	
Description	If the count of bit in "ConstValue" argument that the value is 1 was not 1, return value is unspecified.	
Arguments	bit_flags_fast32_t Variable	The value of target bit flags
	bit_flags_fast32_t ConstValue	The value that investigating bit is 1
Return value	Whether passed bit is 0	

(5) IS_ANY_BITS_NOT_SET

Outline	Evaluate whether any passed bits are 0 or not	
Header	r_ospl.h	
Declaration	bool_t IS_ANY_BITS_NOT_SET(bit_flags_fast32_t Variable, bit_flags_fast32_t ConstValue);	
Description	If the count of bit in "ConstValue" argument that the value is 1 was 0, return value is unspecified.	
Arguments	bit_flags_fast32_t Variable	The value of target bit flags

	bit_flags_fast32_t ConstValue	The value that investigating bits are 1
Return value	Whether the any passed bits are 0	

(6) IS_ALL_BITS_NOT_SET

Outline	Evaluate whether all passed bits are 0 or not	
Header	r_ospl.h	
Declaration	bool_t IS_ALL_BITS_NOT_SET(bit_flags_fast32_t Variable, bit_flags_fast32_t ConstValue);	
Description	If the count of bit in "ConstValue" argument that the value is 1 was 0, return value is unspecified.	
Arguments	bit_flags_fast32_t Variable	The value of target bit flags
	bit_flags_fast32_t ConstValue	The value that investigating bits are 1
Return value	Whether the all passed bits are 0	

4.6.7. Functions for asynchronous notification - r_ospl_async_t type

(1) R_OSPL_ASYNC_SetDefaultPreset

Outline	Sets each member variable to preset values, if preset flag was specified in "r_ospl_async_t" type structure.	
Header	r_ospl.h	
Declaration	void R_OSPL_ASYNC_SetDefaultPreset(r_ospl_async_t* in_out_Async);	
Description	This function is usually called from "R_DRIVER_SetDefaultAsync". This function set member variables by a following preset flag. This function does nothing, if preset flag was not specified. Member variable is not set to preset value, if flag corresponding member variable was specified. This function sets flags corresponding member variable and clears flags corresponding preset, if value corresponding preset flag was set to member variable.	

Preset Flag	Description
R_F_OSPL_AsynchronousPreset	New notification to current thread. And calls "R_OSPL_EVENT_Allocate" function from this function inside. The module using "R_F_OSPL_AsynchronousPreset" must call "R_OSPL_EVENT_Free" with event value of "in_out_Async->A_EventValue". "R_OSPL_RaiseUnrecoverable" is called, if error was raised. Sets like the following description. "in_out_Async->A_Thread" is current thread. "in_out_Async->A_EventValue" is not used bit (bit by "R_OSPL_UNUSED_FLAG") + check bit. The other bits are not accessed.

Arguments	r_ospl_async_t* in_out_Async	Input/Output: r_ospl_async_t type structure
Return value	None	

4.6.8. Functions for queue - r_ospl_queue_id_t type

(1) R_OSPL_QUEUE_DEF

Outline	Defines attributes of queue and work area.	
Header	r_ospl.h	
Declaration	#define R_OSPL_QUEUE_DEF(Name, MaxCount, Type)	
Description	It is not possible to use this macro in the library. If implement of OSPL was changed, the library must be recompiled.	
Arguments	Name	Queue's name. Do not bracket by ""
	MaxCount	Max count of elements in the queue
	Type	Type of element
Return value	None	

(2) R_OSPL_QUEUE

Outline	Returns initial attributes of queue and work area.	
Header	r_ospl.h	
Declaration	r_ospl_queue_def_t* R_OSPL_QUEUE(Name);	
Description	It is not possible to use this macro in the library. If implement of OSPL was changed, the library must be recompiled.	
Arguments	Name	Queue's name. Do not bracket by ""
Return value	Initial attributes of queue and work area.	

(3) R_OSPL_QUEUE_Create

Outline	Initializes a queue	
Header	r_ospl.h	
Declaration	errnum_t R_OSPL_QUEUE_Create(r_ospl_queue_id_t** out_self, r_ospl_queue_def_t* in_QueueDefine);	
Description	It is not possible to call this function from the library. This function is called from porting layer of the driver and send created queue to the driver. OSPL does not have finalizing function (it is portable with CMSIS). An object specified "in_QueueDefine" argument can be specified to the create function 1 times only. Some OS does not have this limitation. The address of a variable as "r_ospl_queue_id_t*" type is set at "out_self" argument. Internal variables of the queue are stored in the variable specified with "in_QueueDefine" argument.	
Arguments	r_ospl_queue_id_t** out_self	Output: Initialized queue object
	r_ospl_queue_def_t* in_QueueDefine	Initial attributes of queue and work area.
Return value	Error code. If there is no error, the return value is 0	

(4) R_OSPL_QUEUE_GetStatus

Outline	Gets status of the queue
Header	r_ospl.h
Declaration	errnum_t R_OSPL_QUEUE_GetStatus(r_ospl_queue_id_t* self, const r_ospl_queue_status_t** out_Status);
Description	Got status are the information at calling moment. If other threads were run, the status will be changed. See "R_DRIVER_GetAsyncStatus" function about pointer type of "out_Status" argument

Arguments	r_ospl_queue_id_t* self	A queue object
	const r_ospl_queue_status_t** out_Status	Output: Pointer to the status structure
Return value	Error code. If there is no error, the return value is 0	

(5) R_OSPL_QUEUE_Allocate

Outline	Allocates an element from the queue object
Header	r_ospl.h
Declaration	errnum_t R_OSPL_QUEUE_Allocate(r_ospl_queue_id_t* self, void* out_Address, uint32_t in_Timeout_msec);
Description	An error will be raised, if "in_Timeout_msec != 0" in interrupt context. It becomes "*out_Address = NULL", when it was timeout and "in_Timeout_msec = 0". E_TIME_OUT error is raised, when it was timeout and "in_Timeout_msec != 0". In OS less environment, "R_OSPL_QUEUE_Allocate" supports pseudo multi-threading. See "R_OSPL_THREAD_GetIsWaiting" function.

Arguments	r_ospl_queue_id_t* self	A queue object
	void* out_Address	Input: Address of pointer variable. Output: Address of allocated element
	uint32_t in_Timeout_msec	Timeout (msec) or R_OSPL_INFINITE
Return value	Error code. If there is no error, the return value is 0	

(6) R_OSPL_QUEUE_Put

Outline	Sends the element to the queue
Header	r_ospl.h
Declaration	errnum_t R_OSPL_QUEUE_Put(r_ospl_queue_id_t* self, void* in_Address);
Description	It is correct, even if other thread put to the queue or get from the queue from calling "R_OSPL_QUEUE_Allocate" to calling "R_OSPL_QUEUE_Put". The message put to the queue by this function receives the thread calling "R_OSPL_QUEUE_Get" function.

Arguments	r_ospl_queue_id_t* self	A queue object
	void* in_Address	Address of element to put
Return value	Error code. If there is no error, the return value is 0	

(7) R_OSPL_QUEUE_Get

Outline	Receives the element from the queue
Header	r_ospl.h
Declaration	errnum_t R_OSPL_QUEUE_Get(r_ospl_queue_id_t* self, void* out_Address, uint32_t in_Timeout_msec);
Description	Call "R_OSPL_QUEUE_Free" function after finishing to access to the element. Don't access the memory area of the element after calling "R_OSPL_QUEUE_Free". "E_NOT_THREAD" error is raised, if "in_Timeout_msec = 0" was specified from interrupt context. It is not possible to wait for put data to the queue in interrupt context. "*out_Address" is NULL and returns 0 (no error), if it became to timeout and "in_Timeout_msec = 0". "E_TIME_OUT" is raised, if "in_Timeout_msec != 0".

Specify "in_Timeout_msec = 0", call the following functions by the following order and use an event for preventing to block to receive other events by the thread having waited for the queue.

Sending Side:
R_OSPL_QUEUE_Allocate
R_OSPL_QUEUE_Put
R_OSPL_EVENT_Set

Receiving Side:
R_OSPL_EVENT_Wait
R_OSPL_QUEUE_Get
R_OSPL_QUEUE_Free

In OS less environment, "R_OSPL_QUEUE_Get" supports pseudo multi-threading. See "R_OSPL_THREAD_GetIsWaiting" function.

Arguments	r_ospl_queue_id_t* self	A queue object
	void* out_Address	Input: Address of pointer variable. Output: Address of received element
	uint32_t in_Timeout_msec	Timeout (msec) or R_OSPL_INFINITE
Return value	Error code. If there is no error, the return value is 0	

(8) R_OSPL_QUEUE_Free

Outline	Returns received memory area to the queue
Header	r_ospl.h
Declaration	errnum_t R_OSPL_QUEUE_Free(r_ospl_queue_id_t* self, void* in_Address);
Description	It is correct, even if other thread put to the queue or get from the queue from calling "R_OSPL_QUEUE_Get" to calling "R_OSPL_QUEUE_Free".

This function makes a space of element in the queue.

Arguments	r_ospl_queue_id_t* self	A queue object
	void* in_Address	Address of element which are got from "R_OSPL_QUEUE_Get"
Return value	Error code. If there is no error, the return value is 0	

(9) R_OSPL_GetQueueAsSingletonLock

Outline	Returns a T-lock object that is locked when singleton was created and deleted.
Header	r_ospl.h
Declaration	r_ospl_queue_id_t R_OSPL_GetQueueAsSingletonLock(void);

Description	Returned lock object (queue) is one in the whole system. For the first call, this function disables all interrupts and creates the lock object and enables all interrupts. The lock object has maximum resource count is 1, therefore it can do exclusive control. Don't call "R_OSPL_QUEUE_Put" and "R_OSPL_QUEUE_Get" with the lock object (queue). OSPL does not define special API function for T-lock.
	
	It is necessary to T-lock, while a singleton object (class object) is initializing and finalizing.
	
	Example:
	`errnum_t CreateSingletonObject()`
	`{`
	`r_ospl_queue_id_t signleton_lock_ID;`

```
    void*           signleton_lock = NULL;

    signleton_lock_ID = R_OSPL_GetQueueAsSingletonLock();
    e= R_OSPL_QUEUE_Allocate( signleton_lock_ID,
           &signleton_lock,  R_OSPL_INFINITE );
       IF(e){goto fin;}  /* Start of T-Lock */
    if ( gs_SingletonObject == NULL ) {


      gs_SingletonObject = ...;

    }
    e=0;
fin:
    if ( signleton_lock != NULL ) {
        ee= R_OSPL_QUEUE_Free( signleton_lock_ID,
               signleton_lock );  /* End of T-Lock */
           e= R_OSPL_MergeErrNum( e, ee );
    }
    return  e;
}
```

Arguments	None	
Return value	Queue as lock object.	

4.6.9. Functions for the area disabled all interrupts

(1) R_OSPL_EnableAllInterrupt

Outline	Releases all disabled interrupts
Header	r_ospl.h
Declaration	void R_OSPL_EnableAllInterrupt();
Description	Driver user should not call this function. Call this function at the end of area of all interrupts disabled. Do not release, if all interrupts were already disabled by caller function. This function does not release disabled NMI.

Arguments	None	
Return value	None	

(2) R_OSPL_DisableAllInterrupt

Outline	Disables all interrupts
Header	r_ospl.h
Declaration	bool_t R_OSPL_DisableAllInterrupt();
Description	Driver user should not call this function. Call this function at begin of area of all interrupts disabled. This function does not disable NMI. Example:

```
  void  Func()
  {
      bool_t  was_all_enabled = false;

      was_all_enabled = R_OSPL_DisableAllInterrupt();
```

```
        /* All interrupt disabled */

        if ( was_all_enabled )
            { R_OSPL_EnableAllInterrupt(); }
    }
```

Arguments	None	
Return value	Whether all interrupts were enabled	

(3) R_OSPL_GetIsAllInterruptEnabled

Outline	Returns whether all interrupts are enabled	
Header	r_ospl.h	
Declaration	bool_t R_OSPL_GetIsAllInterruptEnabled();	
Description		
Arguments	None	
Return value	whether all interrupts are enabled	

4.6.10. Functions for interrupt handling

(1) R_BSP_InterruptWrite

Outline	Registers an interrupt handler	
Header	platform.h or mcu_interrupts.h	
Declaration	bsp_int_err_t R_BSP_InterruptWrite(bsp_int_src_t in_IRQ_Num, bsp_int_cb_t in_Callback);	
Description	-	
Arguments	bsp_int_src_t in_IRQ_Num	Interrupt request number
	bsp_int_cb_t in_Callback	The function as interrupt handler
Return value	Error code. If there is no error, the return value is BSP_INT_SUCCESS.	

(2) R_BSP_InterruptRead

Outline	Returns registered interrupt handler	
Header	platform.h or mcu_interrupts.h	
Declaration	bsp_int_err_t R_BSP_InterruptRead(bsp_int_src_t in_IRQ_Num, bsp_int_cb_t* out_Callback);	
Description	-	
Arguments	bsp_int_src_t in_IRQ_Num	Interrupt request number
	bsp_int_cb_t* out_Callback	Output: the function as interrupt handler
Return value	Error code. If there is no error, the return value is BSP_INT_SUCCESS.	

(3) R_BSP_InterruptControl

Outline	Controls related to the interrupt	
Header	platform.h or mcu_interrupts.h	
Declaration	bsp_int_err_t R_BSP_InterruptControl(bsp_int_src_t in_IRQ_Num, bsp_int_cmd_t in_Command, void* in_Parameter);	
Description	See (4.4.15.) bsp_int_cmd_t.	
Arguments	bsp_int_src_t in_IRQ_Num	Interrupt request number
	bsp_int_cmd_t in_Command	Control command
	void* in_Parameter	See (4.4.15.) bsp_int_cmd_t.
Return value	Error code. If there is no error, the return value is BSP_INT_SUCCESS.	

(4) R_OSPL_SetInterruptPriority

Outline	Sets the priority of the interrupt line.	
Header	r_ospl.h	
Declaration	errnum_t R_OSPL_SetInterruptPriority(bsp_int_src_t in_IRQ_Num, int_fast32_t in_Priority);	
Description	Interrupt of priority less than specified interrupt priority is disabled in interrupt context. Set priority to most high level, if nested interrupt was not supported.	
Arguments	bsp_int_src_t in_IRQ_Num	Interrupt request number
	int_fast32_t in_Priority	Priority. The less the prior.
Return value	Error code. If there is no error, the return value is 0	

(5) R_OSPL_END_OF_INTERRUPT

Outline	Macro that is called at last of interrupt handler	
Header	r_ospl.h	
Declaration	#define R_OSPL_END_OF_INTERRUPT()	

Description	Depending on the define of "R_OSPL_INTERRUPT_ARGUMENTS" this calls "R_UNREFERENCED_VARIABLE".
	
	R_OSPL_INTERRUPT_ARGUMENTS is macro written as common code, if arguments of interrupt handler are different depending on environment (interrupt API).
	
	Example:
	`static void R_DRIVER_IRQ_Handler2(`
	`R_OSPL_INTERRUPT_ARGUMENTS )`
	`{`
	`R_DRIVER_IRQ_HandlerN( 2 ); /* 2 = channel */`
	`R_OSPL_END_OF_INTERRUPT();`
	`}`	

Arguments	None	
Return value	None	

4.6.11. Functions for BSP_CFG_USER_LOCKING_TYPE type

(1) R_OSPL_LockChannel

Outline	Locks by channel number.
Header	r_ospl.h
Declaration	errnum_t R_OSPL_LockChannel(int_fast32_t in_ChannelNum, int_fast32_t* out_ChannelNum, mcu_lock_t in_HardwareIndexCh0, mcu_lock_t in_HardwareIndexChMax);
Description	This function is called from the internal of "R_DRIVER_Initialize" function or "R_DRIVER_LockChannel" function. This function calls "R_BSP_HardwareLock". If "in_Channel" argument was specified to "R_OSPL_UNLOCKED_CHANNEL" (=0xFEE), this function locks unlocked channel. In this case, "out_ChannelNum" argument was specified to "NULL", "E_OTHERS" error is raised, when "R_OSPL_NDEBUG" = 1. If channel number started from except for 0, "in_ChannelNum" argument must be specified to channel number minus minimum channel number. Locked channel number is "out_ChannelNum" + minimum channel number.

This function does not initialize the specified channel of peripheral.

"E_ACCESS_DENIED" error is raised, if specified channel was already locked. "E_FEW_ARRAY" error is raised, if specified channel number was out of range and "R_OSPL_DEBUG" = 1. "E_FEW_ARRAY" error is raised, if "in_ChannelNum" = "R_OSPL_UNLOCKED_CHANNEL" and all channels was locked.

Arguments	int_fast32_t in_ChannelNum	Locking channel number or "R_OSPL_UNLOCKED_CHANNEL"
	int_fast32_t* out_ChannelNum	Output: Locked channel number, (in) NULL is permitted
	mcu_lock_t in_HardwareIndexCh0	Hardware index of channel number = 0
	mcu_lock_t in_HardwareIndexChMax	Hardware index of max channel number
Return value	Error code. If there is no error, the return value is 0.	

(2) R_OSPL_UnlockChannel

Outline	Unlocks by channel number.
Header	r_ospl.h
Declaration	errnum_t R_OSPL_UnlockChannel(int_fast32_t in_ChannelNum, errnum_t e, mcu_lock_t in_HardwareIndexCh0, mcu_lock_t in_HardwareIndexMax);
Description	This function is called from the internal of "R_DRIVER_Finalize" function or "R_DRIVER_UnlockChannel" function. This function calls "R_BSP_HardwareUnlock". If channel number started from except for 0, "in_ChannelNum" argument must be specified to channel number minus minimum channel number. This function does not finalize the specified channel of peripheral. "E_ACCESS_DENIED" error is raised, if the specified channel was already unlocked. This function does nothing and returns 0 (no error) (if argument e=0), if "in_ChannelNum" argument was specified "R_OSPL_UNLOCKED_CHANNEL", under 0, count of channel or more.

Arguments	int_fast32_t in_ChannelNum	Channel number
	errnum_t e	Raising error code, if there is no error, 0
	mcu_lock_t in_HardwareIndexCh0	Hardware index of channel number = 0
	mcu_lock_t in_HardwareIndexMax	Hardware index of max channel number
Return value	Error code or e 0 = successful and e = 0	

(3) R_BSP_HardwareLock

Outline	Locks by hardware index
Header	r_ospl.h, locking.h or platform.h
Declaration	bool_t R_BSP_HardwareLock(mcu_lock_t in_HardwareIndex);
Description	It is usable to call "R_DRIVER_LockChannel" function, if there was 2 and more channels.

This function calls functions by the following order, if "BSP_CFG_USER_LOCKING_ENABLED" is 0.
R_OSPL_Start_T_Lock (Only "R_OSPL_IS_PREEMPTION" = 1)
R_OSPL_C_LOCK_Lock
R_OSPL_End_T_Lock (Only "R_OSPL_IS_PREEMPTION" = 1)
But this function calls "DMA_Alloc" function of RZ/A1H RTX BSP, if BSP environment was used, "BSP_CFG_USER_LOCKING_ENABLED" was 0 and DMAC index was specified.

This function calls "BSP_CFG_USER_LOCKING_HW_LOCK_FUNCTION", if "BSP_CFG_USER_LOCKING_ENABLED" is 1.

Arguments	mcu_lock_t in_HardwareIndex	Hardware index
Return value	Whether the operation was successes or not	

(4) R_BSP_HardwareUnlock

Outline	Unlocks by hardware index
Header	r_ospl.h, locking.h or platform.h
Declaration	bool_t R_BSP_HardwareUnlock(mcu_lock_t in_HardwareIndex);
Description	It is usable to call "R_DRIVER_UnlockChannel" function, if there was 2 and more channels.

"E_ACCESS_DENIED" error is raised, if the specified channel was already unlocked.

This function calls functions by the following order, if "BSP_CFG_USER_LOCKING_ENABLED" is 0.
R_OSPL_Start_T_Lock (Only "R_OSPL_IS_PREEMPTION" = 1)
R_OSPL_C_LOCK_Unlock
R_OSPL_End_T_Lock (Only "R_OSPL_IS_PREEMPTION" = 1)
But this function calls "DMA_Free" function of RZ/A1H RTX BSP, if BSP environment was used, "BSP_CFG_USER_LOCKING_ENABLED" was 0 and DMAC index was specified.

This function calls "R_OSPL_RaiseUnrecoverable" function, if the specified hardware index is out of range.

This function calls "BSP_CFG_USER_LOCKING_HW_UNLOCK_FUNCTION", if "BSP_CFG_USER_LOCKING_ENABLED" is 1.

Arguments	mcu_lock_t in_HardwareIndex	Hardware index
Return value	Whether the operation was success or not	

(5) R_BSP_SoftwareLock

Outline	Locks by lock object.
Header	r_ospl.h, locking.h or platform.h
Declaration	bool_t R_BSP_SoftwareLock(BSP_CFG_USER_LOCKING_TYPE* LockObject);
Description	This function calls functions by the following order, if "BSP_CFG_USER_LOCKING_ENABLED" was 0. R_OSPL_Start_T_Lock (Only "R_OSPL_IS_PREEMPTION" = 1) R_OSPL_C_LOCK_Lock R_OSPL_End_T_Lock (Only "R_OSPL_IS_PREEMPTION" = 1)

If it was 1, this function calls the function defined by "BSP_CFG_USER_LOCKING_SW_LOCK_FUNCTION" macro.

Arguments	BSP_CFG_USER_LOCKING_TYPE* LockObject	Lock object
Return value	Whether the operation was success or not	

Example:

```
/* (*1) If owner is a thread */
/* (*2) If owner is a context */
BSP_CFG_USER_LOCKING_TYPE  lock;  /* 0 initialized global variable */
r_ospl_thread_id_t        lock_owner_thread = R_OSPL_THREAD_NULL;  /* (*1) */
r_ospl_c_lock_t*          lock_owner_object = NULL;  /* (*2) */


    /* In initialize or open function. */
    b= R_BSP_SoftwareLock( &lock ); IF(!b){e=E_ACCESS_DENIED; goto fin;}
    lock_owner_thread = R_OSPL_THREAD_GetCurrentId();  /* (*1) */
    lock_owner_object = &lock;                        /* (*2) */


    /* Checks locking (with right) at start in each API function. */
    IF ( lock_owner_thread != R_OSPL_THREAD_GetCurrentId() )  /* (*1) */
       { e=E_ACCESS_DENIED; goto fin; }
    IF ( lock_owner_object != &lock )                      /* (*2) */
       { e=E_ACCESS_DENIED; goto fin; }

fin:
    /* In finalize or close function. */
    if ( lock_owner_thread == R_OSPL_THREAD_GetCurrentId() )  /* (*1) */
    /* if ( lock_owner_object == &lock ) */                  /* (*2) */
    {
        lock_owner_thread = R_OSPL_THREAD_NULL;  /* (*1) */
        lock_owner_object = NULL;  /* (*2) */
        ee= R_BSP_SoftwareUnlock( &lock ); e=R_OSPL_MergeErrNum( e, ee );
    }
```

(6) R_BSP_SoftwareUnlock

Outline	Unlocks by lock object.
Header	r_ospl.h, locking.h or platform.h
Declaration	bool_t R_BSP_SoftwareUnlock(BSP_CFG_USER_LOCKING_TYPE* LockObject);
Description	"E_ACCESS_DENIED" error is raised, if the specified channel was already unlocked. This function calls functions by the following order, if "BSP_CFG_USER_LOCKING_ENABLED" was 0. R_OSPL_Start_T_Lock (Only "R_OSPL_IS_PREEMPTION" = 1) R_OSPL_C_LOCK_Unlock R_OSPL_End_T_Lock (Only "R_OSPL_IS_PREEMPTION" = 1) If it was 1, this function calls the function defined by "BSP_CFG_USER_LOCKING_SW_UNLOCK_FUNCTION" macro.

Arguments	BSP_CFG_USER_LOCKING_TYPE* LockObject	Lock object
Return value	Whether the operation was success or not	

4.6.12. Functions for r_ospl_c_lock_t type

(1) R_OSPL_C_LOCK_InitConst

Outline	Initializes the C-lock object
Header	r_ospl.h
Declaration	void R_OSPL_C_LOCK_InitConst(r_ospl_c_lock_t* self);
Description	If *self is global variable or static variable initialized 0, this function does not have to be called.

Arguments		
	r_ospl_c_lock_t* self	C-lock object
Return value	None	

(2) R_OSPL_C_LOCK_Lock

Outline	Locks the target, if lockable state.
Header	r_ospl.h
Declaration	errnum_t R_OSPL_C_LOCK_Lock(r_ospl_c_lock_t* self);
Description	Even if lock owner called this function, if lock object was already locked, E_ACCESS_DENIED error is raised. In OS-using environment, call this function in lock area having the code changing the owner information variable. "R_OSPL_C_LOCK_Lock" does not do exclusive control. E_NOT_THREAD error is raised, if this function was called from the interrupt context.

Arguments		
	r_ospl_c_lock_t* self	C-lock object
Return value	Error code. If there is no error, the return value is 0.	

(3) R_OSPL_C_LOCK_Unlock

Outline	Unlocks the target.
Header	r_ospl.h
Declaration	errnum_t R_OSPL_C_LOCK_Unlock(r_ospl_c_lock_t* self);
Description	If this function was called with unlocked object, this function does nothing and raises "E_ACCESS_DENIED" error. If self == NULL, this function does nothing and raises no error. E_NOT_THREAD error is raised, if this function was called from the interrupt context. "R_OSPL_C_LOCK_Unlock" does not do exclusive control. In OS-using environment, call this function in T-lock area having the code changing the owner information variable.

Arguments		
	r_ospl_c_lock_t* self	C-lock object
Return value	Error code. If there is no error, the return value is 0.	

4.6.13. Functions for array index table - r_ospl_table_t type

(1) R_OSPL_TABLE_DEF

Outline	Declares work area and initial value of array index table.
Header	r_ospl.h
Declaration	#define R_OSPL_TABLE_DEF(Name, MaxCount, Flags)
Description	It is not possible to use this macro in the library. If implement of OSPL was changed, the library must be recompiled.

This macro can be written in global scope.
The array index table declared by this macro can be gotten by "R_OSPL_TABLE".

Arguments	Name	Name of array index table. Don't enclose by " "
	MaxCount	Maximum count of element in the table
	Flags	See 4.4.11. r_ospl_table_flags_t
Return value	None	

(2) R_OSPL_TABLE

Outline	Returns work area and initial value of array index table.
Header	r_ospl.h
Declaration	#define R_OSPL_TABLE(Name)
Description	It is not possible to use this macro in the library. If implement of OSPL was changed, the library must be recompiled.

Arguments	Name	Name of array index table. Don't enclose by " "
Return value	Work area and initial value of array index table.	

(3) R_OSPL_TABLE_InitConst

Outline	Initializes an array index table.
Header	r_ospl.h
Declaration	void R_OSPL_TABLE_InitConst(r_ospl_table_t* self, void* in_Area, uint32_t in_AreaSize, r_ospl_table_flags_t in_Flags);
Description	It is not necessary to call this function with the variable declared by "R_OSPL_TABLE_DEF". It is necessary to call this function with the variable not declared by "R_OSPL_TABLE_DEF".

Arguments	r_ospl_table_t* self	Array index table
	void* in_Area	Address of area for the array index table
	uint32_t in_AreaSize	Size of area for the array index table (byte) See "R_OSPL_TABLE_SIZE".
	r_ospl_table_flags_t in_Flags	Option. 0 or bitwise OR of 4.4.11. r_ospl_table_flags_t.
Return value	None	

(4) R_OSPL_TABLE_SIZE

Outline	Calculate size of area for the array index table.
Header	r_ospl.h
Declaration	#define R_OSPL_TABLE_SIZE(int_fast32_t in_MaxCount)
Description	Array index numbers from 0 to "in_MaxCount - 1" will be managed.
Arguments	int_fast32_t in_MaxCount Count of elements in the table
Return value	Size of area for the array index table (byte)

(5) R_OSPL_TABLE_GetIndex

Outline	Returns array index number from specified key.
Header	r_ospl.h
Declaration	errnum_t R_OSPL_TABLE_GetIndex(r_ospl_table_t* self, void* in_Key, int_fast32_t* out_Index, r_ospl_if_not_t in_TypeOfIfNot);
Description	User can know that specified key is not registered by calling after setting "*out_Index = R_OSPL_NO_INDEX", if "in_TypeOfIfNot = R_OSPL_DO_NOTHING_IF_NOT". User can know that specified key is already registered by calling after setting "*out_Index = R_OSPL_NO_INDEX", if "in_TypeOfIfNot = R_OSPL_OUTPUT_IF_NOT".

Symbol	Value	Description
R_OSPL_NO_INDEX	-1	There is not corresponding index number

Arguments	r_ospl_table_t* self	Array index table
	void* in_Key	Key as specified value (address)
	int_fast32_t* out_Index	Output: array index number
	r_ospl_if_not_t in_TypeOfIfNot	Constant value of behavior, when searching key was not found See 4.4.12. r_ospl_if_not_t
Return value	Error code. If there is no error, the return value is 0.	

(6) R_OSPL_TABLE_Free

Outline	Remove from array index table and separate index from key. (key specified)
Header	r_ospl.h
Declaration	void R_OSPL_TABLE_Free(r_ospl_table_t* self, void* in_Key);
Description	This function returns 0 (no error), even if specified key was already removed. This function calls "R_OSPL_RaiseUnrecoverable(E_STATE)" after getting array index number by calling with "R_OSPL_ALLOCATE_IF_EXIST_OR_IF_NOT".

Arguments	r_ospl_table_t* self	Array index table
	void* in_Key	Key
Return value	None	

(7) R_OSPL_TABLE_FreeByIndex

Outline	Remove from array index table and separate index from key. (index specified)
Header	r_ospl.h
Declaration	void R_OSPL_TABLE_FreeByIndex(r_ospl_table_t* self, int_fast32_t in_Index);
Description	This function calls "R_OSPL_RaiseUnrecoverable(E_STATE)", when array index number is never gotten by calling with "R_OSPL_ALLOCATE_IF_EXIST_OR_IF_NOT". This function calls "R_OSPL_RaiseUnrecoverable(E_NOT_FOUND_SYMBOL)", if specified key was already removed.

Arguments	r_ospl_table_t* self	Array index table
	nt_fast32_t in_Index	Array index number
Return value	None	

(8) R_OSPL_TABLE_GetStatus

Outline	Gets (pointer to structure that describes) status of array index table.
Header	r_ospl.h
Declaration	errnum_t R_OSPL_TABLE_GetStatus(r_ospl_table_t* self, const r_ospl_table_status_t** out_Status);
Description	Pointer variable specified at "out_Status" argument is necessary const qualifier. The value of "*out_Status" is updated, even if this function was not called.

Arguments	r_ospl_table_t* self	Array index table
	const r_ospl_table_status_t** out_Status	Output: Pointer to structure that describes status of array index table. See 4.5.14. r_ospl_table_status _t
Return value	None	

4.6.14. Functions for the memory

(1) R_OSPL_MEMORY_Flush

Outline	Flushes cache memory
Header	r_ospl.h
Declaration	void R_OSPL_MEMORY_Flush(r_ospl_flush_t in_FlushType);
Description	Call the function of the driver after flushing input output buffer in the cache memory, If the data area accessing by the hardware is on cache and the driver did not manage the cache memory. Whether the driver manages the cache memory is depend on the driver specification. For RZ/A1H, "in_FlushType" argument can be set only "R_OSPL_FLUSH_WRITEBACK_INVALIDATE". But "R_OSPL_FLUSH_WRITEBACK_INVALIDATE" is little used.
Arguments	r_ospl_flush_t in_FlushType — The operation of flush
Return value	None

(2) R_OSPL_MEMORY_RangeFlush

Outline	Flushes cache memory with the range of virtual address.
Header	r_ospl.h
Declaration	errnum_t R_OSPL_MEMORY_RangeFlush(r_ospl_flush_t in_FlushType, void* in_StartAddress, size_t in_Length);
Description	Align "in_StartAddress" argument and "in_Length" argument to cache line size. If not aligned, E_OTHERS error is raised. Refer to : R_OSPL_MEMORY_GetSpecification If the data area written by the hardware and read from CPU was in cached area, when the hardware started without invalidate ("R_OSPL_FLUSH_WRITEBACK_INVALIDATE" or "R_OSPL_FLUSH_INVALIDATE"), invalidate the data area and read it after finished to write by hardware. (If the driver does not manage the cache memory.) For RZ/A1H, "in_FlushType" argument can be set only "R_OSPL_FLUSH_INVALIDATE".
Arguments	r_ospl_flush_t in_FlushType — The operation of flush void* in_StartAddress — The first virtual address of the range of flushing size_t in_Length — The size of the range of flushing (byte)
Return value	Error code. If there is no error, the return value is 0.

(3) R_OSPL_MEMORY_GetLevelOfFlush

Outline	Gets the level of cache flush for the memory
Header	r_ospl.h
Declaration	errnum_t R_OSPL_MEMORY_GetLevelOfFlush(void* in_Address, int_fast32_t* out_Level);
Description	
Arguments	void* in_Address — The address in flushing memory int_fast32_t* out_Level — Output: 0=Not need to flush, 1=L1 cache only, 2=both of L1 and L2 cache
Return value	Error code. If there is no error, the return value is 0.

(4) R_OSPL_MEMORY_GetMaxLevelOfFlush

Outline	Gets the level of all cache flush for the memory	
Header	r_ospl.h	
Declaration	int_fast32_t R_OSPL_MEMORY_GetMaxLevelOfFlush();	
Description	This function returns 1, if it is not necessary to flush L2 cache.	
Arguments	None	
Return value	Level of cache. 1=L1 cache, 2=L2 cache	

(5) R_OSPL_MEMORY_GetSpecification

Outline	Gets the specification about memory and cache memory.	
Header	r_ospl.h	
Declaration	void R_OSPL_MEMORY_GetSpecification(r_ospl_memory_spec_t* out_MemorySpec);	
Description		
Arguments	r_ospl_memory_spec_t* out_MemorySpec	The specification about memory and cache memory
Return value	None	

(6) R_OSPL_ToPhysicalAddress

Outline	Changes to physical address	
Header	r_ospl.h	
Declaration	errnum_t R_OSPL_ToPhysicalAddress(void* in_Address, uintptr_t* out_PhysicalAddress);	

Description	Example: Case that physical address is integer type
	`uintptr_t physical_address;`
	
	`e= R_OSPL_ToPhysicalAddress( address, &physical_address );`
	`IF(e){goto fin;}`
	
	Example: Case that physical address is pointer type
	`uintptr_t physical_address;`
	`void* pointer;`
	
	`e= R_OSPL_ToPhysicalAddress( address, &physical_address );`
	`IF(e){goto fin;}`
	`pointer = (void*) physical_address;`
	
	See 4.8.9. Pattern of mapping cached area and uncached area (RZ/A1)	

Arguments	void* in_Address	Virtual address
	uintptr_t* out_PhysicalAddress	Output: Physical address
Return value	Error code. If there is no error, the return value is 0.	

(7) R_OSPL_ToCachedAddress

Outline	Changes to the address in the cached area
Header	r_ospl.h
Declaration	errnum_t R_OSPL_ToCachedAddress(void* in_Address, void* out_CachedAddress);
Description	"out_CachedAddress" argument is passed the address of pointer. If the target environment did not have mirror area and "in_Address" argument was passed uncached address, "E_ACCESS_DENIED" error is raised. If "E_ACCESS_DENIED" error was raised, you may know the variable by looking at value of "in_Address" argument and map file.

If the environment did not have any mirror area, it is not necessary to call the function to change the address. But, the function can be used by checking whether passed address is in cached area or uncached area.

See 4.8.9. Pattern of mapping cached area and uncached area (RZ/A1)

Arguments	void* in_Address	Virtual address
	void* out_CachedAddress	(Output) Virtual address for cached area
Return value	Error code. If there is no error, the return value is 0.	

(8) R_OSPL_ToUncachedAddress

Outline	Changes to the address in the uncached area
Header	r_ospl.h
Declaration	errnum_t R_OSPL_ToUncachedAddress(void* in_Address, void* out_UncachedAddress);
Description	"out_UncachedAddress" argument is passed the address of pointer. If the target environment did not have mirror area and "in_Address" argument was passed cached address, "E_ACCESS_DENIED" error is raised. If "E_ACCESS_DENIED" error was raised, you may know the variable by looking at value of "in_Address" argument and map file. If the environment did not have any mirror area, it is not necessary to call the function to change the address. But, the function can be used by checking whether passed address is in cached area or uncached area.

See 4.8.9. Pattern of mapping cached area and uncached area (RZ/A1)

Arguments	void* in_Address	Virtual address
	void* out_UncachedAddress	(Output) Virtual address for uncached area
Return value	Error code. If there is no error, the return value is 0.	

(9) R_OSPL_MEMORY_Barrier

Outline	Set data memory barrier	
Header	r_ospl.h	
Declaration	void R_OSPL_MEMORY_Barrier(void);	
Description	DSB assembly operation of ARM.	
Arguments	None	
Return value	None	

(10) R_OSPL_InstructionSyncBarrier

Outline	Set instruction synchronization barrier	
Header	r_ospl.h	
Declaration	void R_OSPL_InstructionSyncBarrier(void);	
Description	ISB assembly operation of ARM.	
Arguments	None	
Return value	None	

(11) R_OSPL_AXI_Get2ndCacheAttribute

Outline	Gets L2 cache attribute of AXI bus from the address
Header	r_ospl.h
Declaration	errnum_t R_OSPL_AXI_Get2ndCacheAttribute(uintptr_t in_PhysicalAddress, r_ospl_axi_cache_attribute_t* out_CacheAttribute);
Description	This function is used, when L2 cache attribute is different by the kind of the memory.

Arguments	uintptr_t in_PhysicalAddress	The physical address in the memory area
	r_ospl_axi_cache_attribute_t* out_CacheAttribute	Output: The L2 cache attribute of AXI bus
Return value	Error code. If there is no error, the return value is 0.	

(12) R_OSPL_AXI_GetProtection

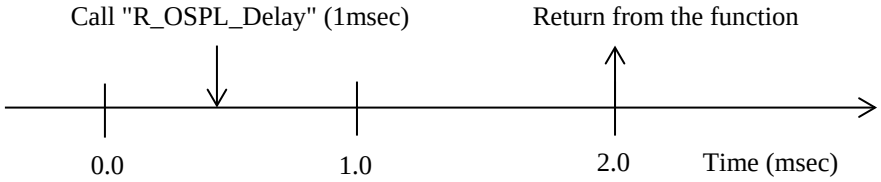
Outline	Gets protection attribute of AXI bus from the address
Header	r_ospl.h
Declaration	errnum_t R_OSPL_AXI_GetProtection(uintptr_t in_PhysicalAddress, r_ospl_axi_protection_t* out_Protection);
Description	This function is used, when protection attribute is different by the kind of the memory.

Arguments	uintptr_t in_PhysicalAddress	The physical address in the memory area
	r_ospl_axi_protection_t* out_Protection	Output: The protection attribute of AXI bus
Return value	Error code. If there is no error, the return value is 0.	

4.6.15. Functions for time

(1) R_OSPL_Delay

Outline	Waits for a while until passed time
Header	r_ospl.h
Declaration	errnum_t R_OSPL_Delay(uint32_t in_DelayTime_msec);
Description	In OS less environment, this function uses free running timer. In OS-using environment, this function uses OS function. This function sometimes waits long time than specified time by timer precision. For example, If the timer precision was 1millisecond, the argument was passed 1millisecond and this function was called at 0.4millisecond, this function returns at 2.0millisecond that it is 1.6millisecond after started to call this function.



If "in_DelayTime_msec" argument was set the value over "R_OSPL_MAX_TIME_OUT",
this function waits for "R_OSPL_MAX_TIME_OUT" millisecond and "E_TIME_OUT" error is raised. In debug configuration (not defined "R_OSPL_NDEBUG"). "ASSERT_D" notifies in this function before waiting. If you want to wait for over "R_OSPL_MAX_TIME_OUT" millisecond, for OS-using environment, use periodic timer. For OS less, use "R_OSPL_FTIMER_IsPast" function with care overflow.

If "R_OSPL_Delay" function was called from the interrupt context, "R_OSPL_RaiseUnrecoverable (E_NOT_THREAD)" is called. If "R_OSPL_RaiseUnrecoverable" returned, "R_OSPL_Delay" function returns "E_NOT_THREAD" error. In interrupt handler, wait by calling "R_OSPL_FTIMER_IsPast" function with care that the timer count does not overflow.

Attention: if waiting process run in the interrupt handler, even most high priority thread cannot run.

In OS less, this function callbacks the function of "r_ospl_idle_callback_t" type, while this function is waiting. Use "R_OSPL_FTIMER_IsPast" function instead of this function, if the system has pseudo multithreading.
OS function like this function sometimes raises an error, when OS function was called from the interrupt handler. In that environment, "R_OSPL_Delay" does the polling.

"R_OSPL_Delay" function returns soon even if waiting state after called "R_OSPL_THREAD_SetOnWait" function with "R_OSPL_WAIT_PM_THREAD" on "R_OSPL_IS_PREEMPTION" = 0. In this case, the return value of "R_OSPL_Delay" function is 0. The return value of "R_OSPL_THREAD_GetIsWaiting" function shows whether waiting state or not.

Arguments	uint32_t in_DelayTime_msec	Time of waiting (millisecond, maximum value is "R_OSPL_MAX_TIME_OUT" (=65533))
Return value	Error code. If there is no error, the return value is 0.	

(2) R_OSPL_FTIMER_InitializeIfNot

Outline	Set up the free running timer
Header	r_ospl.h
Declaration	errnum_t R_OSPL_FTIMER_InitializeIfNot(r_ospl_ftimer_spec_t* out_Specification);
Description	The free running timer does not stop. If the counter of the free running timer was overflow, the counter returns to 0. Even in interrupt handler, the counter does count up. OSPL free running timer does not use any interrupt. Using timer can be selected by "R_OSPL_FTIMER_IS" macro. If the free running timer was already set up, this function does not set up it, outputs to "out_Specification" argument and returns 0 (no error). When OSPL API function with timeout or "R_OSPL_Delay" function was called, "R_OSPL_FTIMER_InitializeIfNot" function is callbacked from these functions. There is all interrupt disabled area inside.

Arguments	r_ospl_ftimer_spec_t* out_Specification	NULL is permitted. Output: The precision of the free run timer
Return value	Error code. If there is no error, the return value is 0.	

(3) R_OSPL_FTIMER_Get

Outline	Get current time of free running timer
Header	r_ospl.h
Declaration	uint32_t R_OSPL_FTIMER_Get(void);

Description	Call "R_OSPL_FTIMER_InitializeIfNot" function before calling this function.
	Call "R_OSPL_FTIMER_IsPast" function, when it is determined whether time passed.
	
	Example:
	`errnum_t e;`
	`r_ospl_ftimer_spec_t ts;`
	`uint32_t start;`
	`uint32_t end;`

```
e= R_OSPL_FTIMER_InitializeIfNot( &ts ); IF(e){goto fin;}
start = R_OSPL_FTIMER_Get();

/* The section of measuring */

end = R_OSPL_FTIMER_Get();
printf( "%d msec\n", R_OSPL_FTIMER_CountToTime(
    &ts, end - start ) );
```

Arguments	None	
Return value	The current clock count of free run timer	

(4) R_OSPL_FTIMER_IsPast

Outline	Returns whether specified time was passed
Header	r_ospl.h
Declaration	errnum_t R_OSPL_FTIMER_IsPast(r_ospl_ftimer_spec_t* out_Specification, uint32_t Now, uint32_t TargetTime, bool_t* out_IsPast);
Description	If the time of free running timer was same as the time of "TargetTime" argument, this function returns false. If past, this function returns true. (If the specification is returning true at same time, waiting time is sometimes short from the target time by the timer precision.) "Now" argument does not have to real current time. Because operations in modules at the same time must use with same time value. If "Now" argument was set more than the target time plus extension time (r_ospl_ftimer_spec_t::ExtensionOfCount), E_TIME_OUT error is raised. Then the interval of calling "R_OSPL_FTIMER_IsPast" must be shorter than extension time. The target time cannot set longer than the time that counter is overflow. For these errors can be detected, E_TIME_OUT error is raised, if the target time was set longer than current timer count plus maximum count / 2. The extension time in "out_Specification" argument can be change by the application. However, the more long extension time was set, the less potential the error is detected. For details, refer to following "About boundary value". While the polling is doing, less priority thread cannot run. If the polling is doing in the interrupt handler, even most high priority thread cannot run. In OS-using environment, while "R_OSPL_Delay" was calling, other thread can run. In OS less environment, other thread cannot run. (Interrupts can run.) Example: Judge whether 300 milliseconds was past.

```
errnum_t            e;
r_ospl_ftimer_spec_t  ts;
uint32_t            target;
bool_t              is_past;

e= R_OSPL_FTIMER_InitializeIfNot( &ts ); IF(e){goto fin;}

target = R_OSPL_FTIMER_Get() +
    R_OSPL_FTIMER_TimeToCount( &ts, 300 );
   /* or  previous_target +
       R_OSPL_FTIMER_TimeToCount( &ts, 300 ); */
```

```
    /* ... */

    e= R_OSPL_FTIMER_IsPast( &ts, R_OSPL_FTIMER_Get(),
       target, &is_past );
       IF(e){goto fin;}
```

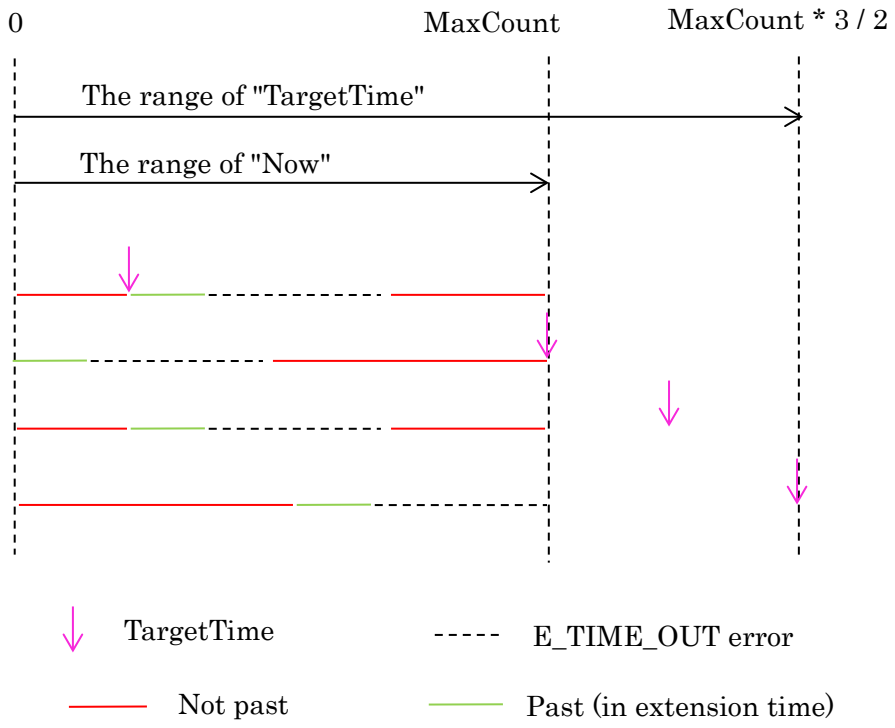
Example: Judge whether 1 hour was past.

```
    errnum_t            e;
    r_ospl_ftimer_spec_t  ts;
    uint32_t            target;
    bool_t              is_past;
    const uint32_t       interval_time = 60 * 1000;  /* msec
 */
    uint32_t            interval_count;
    uint32_t            elapsed_time;

    e= R_OSPL_FTIMER_InitializeIfNot( &ts ); IF(e){goto fin;}

    if ( ts.msec_Numerator >= ts.msec_Denominator ) {
       printf( "Max time is UINT32_MAX/2 msec\n" );
    } else {
       printf( "Max time is %d msec\n",
         R_OSPL_FTIMER_CountToTime( &ts, ts.MaxCount / 2 ) );
    }

    ASSERT_R( ts.msec_Numerator >= ts.msec_Denominator  ||
       R_OSPL_FTIMER_CountToTime( &ts, ts.MaxCount / 2 ) >=
       interval_time,
       e=E_LIMITATION; goto fin );

    interval_count = R_OSPL_FTIMER_TimeToCount( &ts,
        interval_time );
    elapsed_time = 0;
    target = R_OSPL_FTIMER_Get() + interval_count;

    /* ... */

    e= R_OSPL_FTIMER_IsPast( &ts, R_OSPL_FTIMER_Get(),
       target,  &is_past );
       IF(e){goto fin;}
       /* The interval of calling this function must */
       /* be less than ts.ExtensionOfCount. */
    if ( is_past ) {
       elapsed_time += 1;
       if ( elapsed_time >= 60 ) {
          ...
       } else {
          target += interval_count;
       }
    }
```

About boundary value:

If "r_ospl_ftimer_spec_t::MaxCount" is maximum of "uint32_t" and "TargetTime" is more than "r_ospl_ftimer_spec_t::MaxCount", it judges that whether free running timer was over than "TargetTime - (MaxCount + 1)". Because even when the counter was round to 0, current judgement is done,

If "r_ospl_ftimer_spec_t::MaxCount" is not maximum of "uint32_t", "TargetTime" argument can be set the value more than "r_ospl_ftimer_spec_t::MaxCount". However, current time plus "r_ospl_ftimer_spec_t ::MaxCount / 2" raises E_TIME_OUT error.



Arguments	r_ospl_ftimer_spec_t* out_Specification	Precision of the free running timer
	uint32_t Now	Count of current time
	uint32_t TargetTime	Count of target time
	bool_t* out_IsPast	Output: Whether the target time was past or not
Return value	Error code. If there is no error, the return value is 0.	

(5) R_OSPL_FTIMER_TimeToCount

Outline	Change from millisecond unit to free running timer unit
Header	r_ospl.h
Declaration	uint32_t R_OSPL_FTIMER_TimeToCount(r_ospl_ftimer_spec_t* ts, uint32_t msec);
Description	The fractional part is rounded up. (For waiting time must be more than specified time.) This function calculates like the following formula. `( msec * ts->msec_Denominator + ts->msec_Numerator - 1 ) / ts->msec_Numerator` Attention: If "ts->msec_Denominator" was more than "ts->msec_Numerator", take care of overflow.

Arguments	r_ospl_ftimer_spec_t* ts	Precision of the free running timer
	uint32_t msec	The value of millisecond unit
Return value	The value of free running timer unit	

(6) R_OSPL_FTIMER_CountToTime

Outline	Change from free running timer unit to millisecond unit
Header	r_ospl.h
Declaration	uint32_t R_OSPL_FTIMER_CountToTime(r_ospl_ftimer_spec_t* ts, uint32_t Count);
Description	The fractional part is rounded down. (Because overflow does not occur, when "Count = r_ospl_ftimer_spec_t::MaxCount") This function calculates like the following formula. `( Count * ts->msec_Numerator ) / ts->msec_Denominator`

Arguments	r_ospl_ftimer_spec_t* ts	Precision of the free running timer
	uint32_t Count	The value of free running timer unit
Return value	The value of millisecond unit	

(7) R_OSPL_FTIMER_GetSpecification

Outline	Get the specification of the free running timer
Header	r_ospl.h
Declaration	void R_OSPL_FTIMER_GetSpecification(r_ospl_ftimer_spec_t* out_Specification);
Description	Review the definition of this function, when the value of "R_OSPL_FTIMER_IS" macro was added or changed.

Arguments	r_ospl_ftimer_spec_t* out_Specification	NULL is permitted. (Output) The precision of the free running timer
Return value	None	

4.6.16. Functions for the idle state

(1) R_OSPL_IDLE_Start_CPU_Load

Outline	Starts to measure CPU load (for OS less)
Header	r_ospl_os_less.h
Declaration	void R_OSPL_IDLE_Start_CPU_Load(int32_t Interval_msec);
Description	This function sets OSPL to call printf with CPU load from "R_OSPL_OnIdleDefault" function during periodic intervals. This function is enabled, if OSPL was compiled with setting "R_OSPL_CPU_LOAD" to 1. Call "R_OSPL_IDLE_Print_CPU_Load" function, when CPU load is shown soon. "R_OSPL_WAIT_POLLING" must be set.

Arguments	int32_t Interval_msec	The periodic intervals to show (msec), 0=Not show
Return value	None	

(2) R_OSPL_IDLE_Print_CPU_Load

Outline	Show the CPU load by printf (for OS less)
Header	r_ospl.h
Declaration	void R_OSPL_IDLE_Print_CPU_Load(bool_t IsPrintNow);
Description	This function is enabled, if OSPL was compiled with setting "R_OSPL_CPU_LOAD" to 1. This function shows CPU load from calling "R_OSPL_IDLE_Start_CPU_Load" function or the time of "Interval_msec" argument of its function until calling "R_OSPL_IDLE_Print_CPU_Load" function. However, it is necessary to call "R_OSPL_IDLE_Start_CPU_Load" function once at least.

If "IsPrintNow = false", this function shows, when the time of "Interval_msec" argument was passed only.
This function shows CPU load from previous showing to now, if "Interval_msec" argument of "R_OSPL_IDLE_Start_CPU_Load" function was 0.
Developer of application can customize this function.

CPU load keeps 100%, if "R_OSPL_WAIT_POLLING" was not set.

Arguments	bool_t IsPrintNow	Whether shows immediately unless the timing of periodic to show
Return value	None	

4.6.17. Functions for interrupt callback functions - r_ospl_caller_t type

(1) R_OSPL_CallInterruptCallback

Outline	Calls the interrupt callback function. It is called from OS porting layer in the driver	
Header	r_ospl.h	
Declaration	void R_OSPL_CallInterruptCallback(r_ospl_caller_t* self, r_ospl_interrupt_t* InterruptSource);	
Description	This function calls the function registered in "self" as "bsp_int_cb_t" type. Call this function from the interrupt handler in OS porting layer of the driver. The value of "self" argument is passed from the driver unit to OS porting layer of the driver before the interrupt.	
Arguments	r_ospl_caller_t* self	The internal parameters about interrupt operations
	r_ospl_interrupt_t* InterruptSource	The source of the interrupt
Return value	None	

(2) r_ospl_callback_t

Outline	The function type of interrupt callback
Header	r_ospl.h
Declaration	typedef errnum_t (* r_ospl_callback_t)(const r_ospl_interrupt_t* InterruptSource, const r_ospl_caller_t* Caller);
Description	This is type of the interrupt callback function running in the interrupt context and called from the interrupt handler. It is possible to replace to application defined interrupt callback function by setting to "r_ospl_async_t::InterruptCallback". But it is usually not necessary to replace. As interrupt callback function, the default interrupt callback function provided from the driver is used. It is unusual to use application defined interrupt callback function. Write the response code of the interrupt (event driven code) next to the code calling "R_OSPL_EVENT_Wait" function Whether the asynchronous operation was ended is possible to know whether the variable of "r_ospl_async_state_t" type referred from "R_DRIVER_GetAsyncStatus" function is set to "R_OSPL_RUNNABLE" value. It is not necessary to write the code of interrupt return (IRET) in the interrupt callback function. The interrupt handlers calling interrupt callback function calls IRET, if necessary.

It is not possible to divide interrupt callback functions by the kind of interrupt. Alternatively, it is possible to write operations in interrupt handlers calling interrupt callback function in porting layer under the driver. There are interrupt handlers by each interrupt numbers.

It is possible to signal any event from application defined interrupt callback function. But it is necessary to do following operations:

Necessary operations in the interrupt callback function:

If "r_ospl_async_t::I_Thread == R_OSPL_THREAD_NULL", call "R_DRIVER_OnInterrupted" function after calling "R_DRIVER_OnInterrupting" function.
(If there was not "R_DRIVER_OnInterrupted" function, do not call it.)

If "r_ospl_async_t::I_Thread != R_OSPL_THREAD_NULL", signal the I-event after calling "R_DRIVER_OnInterrupting" function. The thread receiving the I-event must call "R_DRIVER_OnInterrupted" function.
(If there was not "R_DRIVER_OnInterrupted" function, do not signal the I-event.)

Arguments	r_ospl_interrupt_t* InterruptSource	Source of interrupt
	r_ospl_caller_t* Caller	Driver's internal parameters about interrupt operations
Return value	Error code. If there is no error, the return value is 0. The value set to "r_ospl_async_t::ReturnValue"	

4.6.18. Functions for error handling and debugging

(1) CHK

Outline	Enters infinite loop, if error was raised. It is for debugging
Header	r_ospl.h
Declaration	void CHK(errnum_t e);
Description	This returns without doing anything, if e == 0. Otherwise this disables all interrupts and enters infinite loop.

Example A:

```
e= FunctionX(); CHK(e);
```

Example B:

```
CHK( FunctionX() );
```

Arguments	errnum_t e	Checking error code
Return value	None	

(2) R_OSPL_RaiseUnrecoverable

Outline	Raises the error of system unrecoverable
Header	r_ospl.h
Declaration	void R_OSPL_RaiseUnrecoverable(errnum_t e);
Description	The error of system unrecoverable is the error of impossible to self-recover by process or main system. Example, the heap area was broken or there are not any responses from hardware. This error can be recoverable by OS or the system controller(e.g. Software reset)

Example, when an error of recovery process was raised,
"R_OSPL_RaiseUnrecoverable" function must be called.

"R_OSPL_RaiseUnrecoverable" function can be customized by the application. By default, it calls "R_DebugBreak" function and falls into the infinite loop.

Arguments	errnum_t e	Error code
Return value	None	

(3) R_DEBUG_BREAK

Outline	Breaks here
Header	r_ospl.h
Declaration	#define R_DEBUG_BREAK()
Description	Does break by calling "R_DebugBreak" function. This macro is not influenced the setting of "R_OSPL_ERROR_BREAK" macro.

Arguments	None	
Return value	None	

(4) R_DEBUG_BREAK_IF_ERROR

Outline	Breaks here, if it is error state
Header	r_ospl.h
Declaration	#define R_DEBUG_BREAK_IF_ERROR()
Description	This function does nothing, if "R_OSPL_ERROR_BREAK" macro was defined to be 0. The following descriptions are available, if "R_OSPL_ERROR_BREAK" macro was defined to be 1. Checks the error state of the current thread. Call this macro from the last of each thread. Does break by calling "R_DebugBreak" function. If an error was raised, this function calls "printf" with following message. Set "error_ID" to "R_OSPL_SET_BREAK_ERROR_ID" `<ERROR error_ID="0x1" file="../src/api.c(336)"/>`

Arguments	None	
Return value	None	

(5) IF

Outline	Breaks and transits to error state, if condition expression is not 0
Header	r_ospl.h
Declaration	#define IF (Condition)
Description	"IF" is as same as general "if", if "R_OSPL_ERROR_BREAK" macro was defined to be 0. The following descriptions are available, if "R_OSPL_ERROR_BREAK" macro was defined to be 1. "IF" macro supports to find the code raising an error. Example: `e= TestFunction(); IF(e){goto fin;}` If the "if statement" that is frequently seen in guard condition and after calling functions was changed to "IF" macro, the CPU breaks at raising an error. Then the status (values of variables) can be looked immediately and the code (call stack) can be looked. Thus, debug work grows in efficiency.

"IF" macro promotes recognizing normal code and exceptional code. Reading speed will grow up by skipping exceptional code.

Call "R_OSPL_SET_BREAK_ERROR_ID" function, if set to break at the code raising an error.

Whether the state was error state or the error raised count is stored in the thread local storage. In Release configuration, the variable of error state and the error raised count is deleted. Manage the error code using auto variable and so on at out of OSPL.

The error state is resolved by calling "R_OSPL_CLEAR_ERROR" function. If "R_DEBUG_BREAK_IF_ERROR" macro was called with any error state, the process breaks at the macro.

"R_OSPL_CHECK_STACK_OVERFLOW" function is called from "IF" macro inside, if "R_OSPL_STACK_CHECK_CODE" macro is 1.

Arguments	Condition	Condition expression
Return value	None	

(6) IF_D

Outline	It is same as "IF" (for Debug configuration only)
Header	r_ospl.h
Declaration	#define IF_D (Condition)
Description	In Release configuration, the result of condition expression is always "false". The release configuration is the configuration defined "R_OSPL_NDEBUG".

Arguments	Condition	Condition expression
Return value	None	

(7) ASSERT_R

Outline	Assertion (Programming by Contract)
Header	r_ospl.h
Declaration	#define ASSERT_R(in_Condition, in_StatementsForError)
Description	Case of defined "R_OSPL_ERROR_BREAK" to be 0: If the result of condition expression is 0(false), do "in_StatementsForError". Case of defined "R_OSPL_ERROR_BREAK" to be 1: If the result of condition expression is 0(false), the error state will become active and the operation of "in_StatementsForError" argument will be done. "R_OSPL_CHECK_STACK_OVERFLOW" function is called from "ASSERT_R" macro inside, if "R_OSPL_STACK_CHECK_CODE" macro is 1. If operations did nothing, write "R_NOOP()" at "in_StatementsForError" argument. Call "R_OSPL_ReturnFalse" function, if code of "false" passed at "in_Condition" argument was warned. Example: `ASSERT_R( size <= sizeof( buffer ), goto fin );`

Arguments	in_Condition	The condition expression expected true

	in_StatementsForError	The operation doing at condition is false. It is possible to write complex sentence divided by ";".
Return value	None	

(8) ASSERT_D

Outline	Assertion (Programming by Contract) (for Debug configuration only)
Header	r_ospl.h
Declaration	#define ASSERT_D(Condition, StatementForError)
Description	This does nothing in Release configuration. Release configuration is the configuration defined "R_OSPL_NDEBUG" as same as standard library.

Arguments	Condition	The condition expression expected true
	StatementsForError	The operation doing at condition is false. It is possible to write complex sentence divided by ";".
Return value	None	

(9) R_STATIC_ASSERT

Outline	Assertion (Programming by Contract). It raises compiling error, if static condition was false
Header	r_ospl.h
Declaration	#define R_STATIC_ASSERT(ConstantExpression, StringLiteral)
Description	Use "R_STATIC_ASSERT_GLOBAL", if in global scope. "STATIC_ASSERT" is as same as "static_assert" of C++0x. But OSPL changes the symbol because it goes against GSCE naming rule. No assembly code is generated.

Arguments	ConstantExpression	The condition expression (The result must be given at the compile time)
	StringLiteral	Write "". This is ignored.
Return value	None	

(10) R_STATIC_ASSERT_GLOBAL

Outline	"R_STATIC_ASSERT" for global scope
Header	r_ospl.h
Declaration	#define R_STATIC_ASSERT_GLOBAL(ConstantExpression, StringLiteral)
Description	No assembly code is generated.

Arguments	ConstantExpression	The condition expression (The result must be given at the compile time)
	StringLiteral	Write "". This is ignored.
Return value	None	

(11) R_NOOP

Outline	The function doing nothing
Header	r_ospl.h
Declaration	void R_NOOP();
Description	This can be written at "StatementsForError" argument of "ASSERT_R"

Arguments	None	
Return value	None	

(12) R_OSPL_MergeErrNum

Outline	Merge the error code raised in the finalizing operation
Header	r_ospl.h

Declaration	errnum_t R_OSPL_MergeErrNum(errnum_t CurrentError, errnum_t AppendError);

Description	When the state was error state, if other new error was raised, new error code is ignored.
	If "CurrentError != 0", this function returns "CurrentError" argument.
	If "CurrentError == 0", this function returns "AppendError" argument.
	This function can be modified by user.
	
	Example:
	`ee= Sample();`
	`e= R_OSPL_MergeErrNum( e, ee );`
	`return e;`

Arguments	errnum_t CurrentError	Current error code
	errnum_t AppendError	New append error code
Return value	Merged error code	

(13) R_OSPL_SetErrNum

Outline	Sets an error code to TLS (Thread Local Storage).
Header	r_ospl.h
Declaration	void R_OSPL_SetErrNum(errnum_t e);
Description	Usually error code is returned. If API function cannot return any error code, API function can have the specification of setting error code by "R_OSPL_SetErrNum". There is this function, if "R_OSPL_TLS_ERROR_CODE" macro was defined to be 1. This function does nothing, if any error code was stored already in TLS. The state does not change to error state, if "R_OSPL_SetErrNum" function was called only. See "R_OSPL_GET_ERROR_ID".

Arguments	errnum_t e	Raising error code
Return value	None	

(14) R_OSPL_GetErrNum

Outline	Returns the error code from TLS (Thread Local Storage).
Header	r_ospl.h
Declaration	errnum_t R_OSPL_GetErrNum();
Description	Usually error code is returned. If API function cannot return any error code, API function may have the specification of getting error code by "R_OSPL_GetErrNum". There is this function, if "R_OSPL_TLS_ERROR_CODE" macro was defined to be 1. This function returns 0 after called "R_OSPL_CLEAR_ERROR" function.

Arguments	None	
Return value	Error code	

(15) R_OSPL_CLEAR_ERROR

Outline	Clears the error state
Header	r_ospl.h
Declaration	void R_OSPL_CLEAR_ERROR();
Description	This function does nothing, if "R_OSPL_ERROR_BREAK" macro and "R_OSPL_TLS_ERROR_CODE" macro were defined to be 0. But code of calling "R_OSPL_CLEAR_ERROR" function is not deleted, because library must clear the error raised from callback function in the application, when the application set with "R_OSPL_ERROR_BREAK = 1" linked with the library set with "R_OSPL_ERROR_BREAK = 0". The following descriptions are available, if "R_OSPL_ERROR_BREAK" macro was defined to be 1.

Whether the state is the error state is stored in thread local storage.
"R_OSPL_GetErrNum" function returns 0 after called this function.

If the error state was not cleared, the following descriptions were caused.

- Breaks at "R_DEBUG_BREAK_IF_ERROR" macro
- "R_OSPL_SET_BREAK_ERROR_ID" function behaves not expected behavior because the count of error is not counted up.

Arguments	None	
Return value	None	

(16) R_OSPL_NOTIFY_ERROR

Outline	Notifies an error to another thread and clears an error of current thread.
Header	r_ospl.h
Declaration	void R_OSPL_NOTIFY_ERROR(errnum_t* in_out_ErrorNumInOtherThread, errnum_t in_NewErrorNum);

Description	This function notifies by simple substitution like "*in_out_ErrorNumInOtherThread = in_NewErrorNum". This function does not access internal status of the thread.
	
	This function writes an error code in Int Log, if "in_NewErrorNum != 0", "R_OSPL_DEBUG_TOOL = 1" and "R_OSPL_ERROR_NOTIFICATION_WATCH = 1".
	
	Example:
	`r_ospl_async_t async;`
	
	`R_OSPL_NOTIFY_ERROR( /*Set*/ &async.ReturnValue, e );`
	
	This function does like the following code, if "R_OSPL_ERROR_NOTIFICATION_WATCH = 0".
	
	Example:
	`r_ospl_async_t async;`
	
	`async.ReturnValue = e;`
	`if ( e != 0 ) {`
	`  R_OSPL_CLEAR_ERROR();`
	`}`

Arguments	errnum_t* in_out_ErrorNumInOtherThread	Address of variable to get error code from current thread.
	errnum_t in_NewErrorNum	Error code of current thread
Return value	None	

(17) R_OSPL_SET_BREAK_ERROR_ID

Outline	Register to break at raising error at the moment
Header	r_ospl.h
Declaration	void R_OSPL_SET_BREAK_ERROR_ID(int_fast32_t ErrorID);
Description	This function does nothing, if "R_OSPL_ERROR_BREAK" macro was defined to be 0. You know whether error raising state or not by checking variable of "errnum_t" in this condition.

The following descriptions are available, if "R_OSPL_ERROR_BREAK" macro was defined to be 1.

Set a break point at "R_DebugBreak" function, when the process breaks at the error raised code.

The number of "ErrorID" argument can be known by "R_DEBUG_BREAK_IF_ERROR" macro or "R_OSPL_GET_ERROR_ID" macro.
In multi-threading environment, the number of "ErrorID" argument is the number of raised errors in all threads.

The following code breaks at first error.

```
R_OSPL_SET_BREAK_ERROR_ID( 1 );
```

The following code breaks at next error after resuming from many errors.

```
R_OSPL_SET_BREAK_ERROR_ID( R_OSPL_GET_ERROR_ID() + 1 );
```

Arguments	int_fast32_t ErrorID	Breaking number of error
Return value	None	

(18) R_OSPL_GET_ERROR_ID

Outline	Returns the number of current error
Header	r_ospl.h
Declaration	int_fast32_t R_OSPL_GET_ERROR_ID();
Description	This function does nothing, if "R_OSPL_ERROR_BREAK" macro was defined to be 0. The following descriptions are available, if "R_OSPL_ERROR_BREAK" macro was defined to be 1.

This function returns 0, if any errors were not raised.
This function returns 1, if first error was raised.
After that, this function returns 2, if second error was raised after calling "R_OSPL_CLEAR_ERROR" function.
This function does not return 0 after that the error was cleared by calling "R_OSPL_CLEAR_ERROR".
The number of current error is running number in the whole of system (all threads).

Error is raised by following macros.

```
IF, IF_D, ASSERT_R, ASSERT_D
```

The process breaks at a moment of error raised, if the number of current error was set to "R_OSPL_SET_BREAK_ERROR_ID" macro.

Arguments	None	
Return value	The number of current error	

(19) R_OSPL_DEBUG_WORK_SIZE

Outline	Calculates the size of debug work area
Header	r_ospl.h
Declaration	#define R_OSPL_DEBUG_WORK_SIZE(int_fast32_t ThreadMaxCount)
Description	"ThreadMaxCount" argument must be added for the interrupt context. For example, if thread max count was 2, "ThreadMaxCount" is 3.

Arguments	int_fast32_t ThreadMaxCount	The max count of threads + count of interrupt level
Return value	The size of work area (byte)	

(20) R_OSPL_GetCurrentThreadError

Outline	Returns debug information of current thread.	
Header	r_ospl.h	
Declaration	r_ospl_error_t* R_OSPL_GetCurrentThreadError(void);	
Description	Don't modify member variable in return value.	
Arguments	None	
Return value	Debug information	

(21) R_OSPL_FreeCurrentThreadError

Outline	Releases an area of error and debug information in current thread.	
Header	r_ospl.h	
Declaration	void R_OSPL_FreeCurrentThreadError(void);	
Description	"R_OSPL_GetCurrentThreadError" function raises an error, when the count of thread that calls error handling API was over than "R_OSPL_DEBUG_THREAD_COUNT". The count is always increased, if this function did not call. Please call this function at the last of thread function, This function deletes the checking information whether event was allocated by calling "R_OSPL_EVENT_Allocate" function and changes to the state of not allocating the area. This function does nothing, if current thread was not attached debug information.	
Arguments	None	
Return value	None	

(22) R_OSPL_CHANGE_THREAD_LOCKED_COUNT

Outline	Modifies value of counter that can be accessed by current thread only	
Header	r_ospl.h	
Declaration	void R_OSPL_CHANGE_THREAD_LOCKED_COUNT(r_ospl_thread_id_t ThreadID, int_fast32_t Plus);	
Description	Drivers calls this function. This function is not called from OSPL. This function does nothing, if "R_OSPL_ERROR_BREAK" macro is 0.	
Arguments	r_ospl_thread_id_t ThreadID	ID of thread
	int_fast32_t Plus	The value of adding to the counter. The counter is subtracted, if this argument was minus.
Return value	None	

(23) R_OSPL_GET_THREAD_LOCKED_COUNT

Outline	Returns value of counter that can be accessed by current thread only	
Header	r_ospl.h	
Declaration	int_fast32_t R_OSPL_GET_THREAD_LOCKED_COUNT(r_ospl_thread_id_t ThreadID);	
Description	This function returns 0, if "R_OSPL_ERROR_BREAK" macro is 0.	
Arguments	r_ospl_thread_id_t ThreadID	ID of thread
Return value	Count of objects that current thread has locked	

(24) R_OSPL_GET_STACK_POINTER

Outline	Returns value of current stack pointer for debugging.

Header	r_ospl.h	
Declaration	uint8_t* R_OSPL_GET_STACK_POINTER();	
Description	There is not this function, if "R_OSPL_STACK_CHECK_CODE = 0". This function is enabled, if "R_OSPL_STACK_CHECK_CODE = 1".	
Arguments	None	
Return value	Value of current stack pointer	

(25) R_OSPL_SET_END_OF_STACK

Outline	Sets the end of stack area of current thread for debugging.	
Header	r_ospl.h	
Declaration	void R_OSPL_SET_END_OF_STACK(void* in_EndOfStackAddress);	
Description	This function writes a canary's value "R_OSPL_STACK_CHECK_CANARY_VALUE" at the end of stack area. Stack check is disabled until calling current function. Stack check is disabled, if the argument is specified with NULL. Please, call "R_OSPL_RESET_MIN_FREE_STACK_SIZE" after calling this function, if it needs. A stack area is shared by all pseudo threads and interrupt context for OS less environment. Stack areas are split by each threads and interrupt context for OS-using environment. This function set a canary value at a stack area for current thread or current interrupt context. There is not this function, if "R_OSPL_STACK_CHECK_CODE = 0". This function is enabled, if "R_OSPL_STACK_CHECK_CODE = 1".	
Arguments	void* in_EndOfStackAddress	Address of the end of stack area or NULL.
Return value	None	

(26) R_OSPL_MOVE_END_OF_STACK

Outline	Moves end of stack area	
Header	r_ospl.h	
Declaration	void R_OSPL_MOVE_END_OF_STACK(void* in_EndOfStackAddress);	
Description	This moves the end of stack area of current thread and fills canary value at the expanded area for debugging. Some standard library changes the size of stack area by expanding heap area. This function supports the specification. The difference from "R_OSPL_SET_END_OF_STACK" is to fill the canary value "R_OSPL_STACK_CHECK_CANARY_VALUE" at the expanded area, when the end of stack area was moved. The filling "R_OSPL_GET_MIN_STACK_POINTER" works properly.	
Arguments	void* in_EndOfStackAddress	Address of new end of stack area
Return value	None	

(27) R_OSPL_CHECK_STACK_OVERFLOW

Outline	Checks that stack overflow occurred.
Header	r_ospl.h
Declaration	void R_OSPL_CHECK_STACK_OVERFLOW();
Description	This calls "R_OSPL_RaiseUnrecoverable" function, if stack overflow occurred.

This function checks whether the canary value is overwritten in the current stack.
This function is called from "IF" macro and "ASSERT_R" macro.

There is not this function, if "R_OSPL_STACK_CHECK_CODE = 0".
This function is enabled, if "R_OSPL_STACK_CHECK_CODE = 1".

This function does nothing until setting at the end of stack by "R_OSPL_SET_END_OF_STACK".

You know the code of changing stack size by breaking "R_OSPL_SET_END_OF_STACK", when stack overflow occurred.

This function does not check, if a call to OS API is prohibited.

Arguments	None	
Return value	None	

(28) R_OSPL_RESET_MIN_FREE_STACK_SIZE

Outline	Resets minimum free stack size in current thread.
Header	r_ospl.h
Declaration	errnum_t R_OSPL_RESET_MIN_FREE_STACK_SIZE();
Description	This function fills "R_OSPL_STACK_CHECK_CANARY_VALUE" from the end of stack specified by "R_OSPL_SET_END_OF_STACK" to the stack pointer for current thread or current interrupt context. There is not this function, if "R_OSPL_STACK_CHECK_CODE = 0". This function is enabled, if "R_OSPL_STACK_CHECK_CODE = 1".

Arguments	None	
Return value	Error code. If there is no error, the return value is 0.	

(29) R_OSPL_GET_MIN_FREE_STACK_SIZE

Outline	Counts minimum free stack size in current thread.
Header	r_ospl.h
Declaration	size_t R_OSPL_GET_MIN_FREE_STACK_SIZE();
Description	Minimum free stack size means minimum free space size in the stack area from calling "R_OSPL_RESET_MIN_FREE_STACK_SIZE" to calling "R_OSPL_GET_MIN_FREE_STACK_SIZE". This function investigates a size of area not changed from "R_OSPL_STACK_CHECK_CANARY_VALUE" in the stack area for current thread or current interrupt context. To measure the size of the stack used to perform an operation, calculate by using "R_OSPL_GET_MIN_STACK_POINTER". See There is not this function, if "R_OSPL_STACK_CHECK_CODE = 0". This function is enabled, if "R_OSPL_STACK_CHECK_CODE = 1".

Arguments	None	
Return value	Minimum free stack size	

(30) R_OSPL_GET_MIN_STACK_POINTER

Outline	Searches the position of stack pointer when largest stack area was used by now
Header	r_ospl.h

Declaration	void* R_OSPL_GET_MIN_STACK_POINTER();
Description	To search the position of the end of largest using range in the stack area of current thread or interrupt context, this function reads in it and returns the address of position that read value at the position was not same as "R_OSPL_STACK_CHECK_CANARY_VALUE". Size of stack usage is calculated by "before-address" - "after-min-address". "before-address" is value of stack pointer before target operation. "after-min-address" is position of the end of largest using range after target operation. The way is correct, even if the end of stack area was moved by "R_OSPL_MOVE_END_OF_STACK" for expanding heap area or others. There is not this function, if "R_OSPL_STACK_CHECK_CODE = 0". This function is enabled, if "R_OSPL_STACK_CHECK_CODE = 1". This function returns NULL, if the end of stack area did not set by "R_OSPL_SET_END_OF_STACK".

Arguments	None	
Return value	The position of stack pointer when largest stack area was used by now	

(31) R_D_Add

Outline	Registers watching integer variable or pointer variable
Header	r_ospl_debug.h
Declaration	void R_D_Add(int_fast32_t IndexNum, void* Address, uint32_t BreakValue, bool_t IsPrintf);
Description	"R_D_Add" and "R_D_Watch" are APIs related to watch function. This debug tool is available, if "R_OSPL_DEBUG_TOOL" was set to 1. Example: `R_D_Add( 0, &var, 0xB0, true ); // 0xB0 may be not hit` There is not this function, if "R_OSPL_DEBUG_TOOL" macro was defined to be 0. This function is available, if "R_OSPL_DEBUG_TOOL" macro was defined to be 1.

Arguments	int_fast32_t IndexNum	Watch Number, 0 or more
	void* Address	Address of watching integer variable or pointer variable
	uint32_t BreakValue	Breaking value of variable when "R_D_Watch" was called
	bool_t IsPrintf	Whether "printf" is called, when "R_D_Watch" was called
Return value	None	

(32) R_D_Watch

Outline	Show and Check watching variable's value
Header	r_ospl_debug.h
Declaration	void R_D_Watch(int_fast32_t IndexNum);

Description	"R_D_Add" and "R_D_Watch" are APIs related to watch function.
	This debug tool is available, if "R_OSPL_DEBUG_TOOL" was set to 1.
	This function calls "printf" with the value of variable registered by "R_D_Add" and breaks when watching variable becomes registered value.
	This function can be called from out of scope of registered variable. Then this function can be written at many places without care of the scope.
	Example:
	`R_D_Watch( 0 );`
	`printf( "Line: %d", __LINE__ );`
	`  // Show the value indicated this place`

There is not this function, if "R_OSPL_DEBUG_TOOL" macro was defined to be 0.
This function is available, if "R_OSPL_DEBUG_TOOL" macro was defined to be 1.

If internal data of the debug tool was broken, the global variable in "r_ospl_debug.o" file should be move to safe address (memory map).

Arguments	int_fast32_t IndexNum	Watch Number, 0 or more
Return value	None	

(33) R_D_AddToIntLog

Outline	Records to the log fast
Header	r_ospl_debug.h
Declaration	void R_D_AddToIntLog(int_fast32_t Value);

Description	"R_D_AddToIntLog", "g_IntLog" and "g_IntLogLength" are APIs related to Int Log function.
	This debug tool is available, if "R_OSPL_DEBUG_TOOL" was set to 1.
	This function overwrites from the first of the log, if max element value of "g_IntLog" was over.
	It is recommended to record not only the showing value of variable, but also the value of the identifier of the place and the value of current time.
	
	"g_IntLog", "g_IntLogLength" and "g_DebugVar" are external linkage global variable. "g_DebugVar" is not accessed by "R_D_AddToIntLog" function. This variable can be used for the conditional break referred the variable over the scope. And this variable can be used for logging the last position that CPU went through .
	`int_fast32_t g_IntLog[100];`
	`int_fast32_t g_IntLogLength;`
	`int_fast32_t g_DebugVar[10];`
	
	There is not this function, if "R_OSPL_DEBUG_TOOL" macro was defined to be 0.
	This function is available, if "R_OSPL_DEBUG_TOOL" macro was defined to be 1.
	
	If internal data of the debug tool was broken, the global variable in "r_ospl_debug.o" file should be moved to safe address (memory map).

Arguments	int_fast32_t Value	Recording value
Return value	None	

(34) R_D_Counter

Outline	Count the through count
Header	r_ospl_debug.h
Declaration	bool_t R_D_Counter(int_fast32_t* in_out_Counter, int_fast32_t TargetCount, char* Label);

Description	This debug tool is available, if "R_OSPL_DEBUG_TOOL" was set to 1.
	If this function was called with "TargetCount = 0", the count of through is output by "printf" for each calling. If "TargetCount" argument was set to the through count and restart the program, when the counter was counted up to "TargetCount", this function returns "true". If there were many "printf" output, set "Label = NULL". At first calling, the address of the counter is output by "printf". The counter can be look by the debugger.
	
	Example:
	`{ static int tc; if ( R_D_Counter( &tc, 0, "A" ) ) {`
	`R_DEBUG_BREAK(); }}`

Arguments	int_fast32_t* in_out_Counter	Input/Output: The through counter
	int_fast32_t TargetCount	The value comparing with the through counter

	char* Label	The label for "printf", NULL=printf : no output
Return value	Whether the through counter is counted up to "TargetCount"	

4.6.19. Functions for reviewed tags for the static code analyzer

(1) IS

Outline	Changes the code accepted with MISRA 13.2 to readable
Header	r_static_an_tag.h
Declaration	#define IS(bool_value)

Description	Avoid "not 0" as double negation.
	"IS" macro conforms to cast to boolean type specified in the language.
	Example:
	`bool_t is_condition = ( x > 0 );`
	`if ( IS( is_condition ) ) { ... }`
	`/* if ( is_condition != 0 ) ) { ... } */`
	
	Write this macro after being warned by static code analyzer.

Arguments	bool_value	The expression that evaluated result becomes boolean type
Return value	Evaluate result of "bool_value != 0".	

(2) R_OSPL_ReturnFalse

Outline	Countermeasure for the warning, when "if" block was disabled.
Header	r_ospl.h
Declaration	int_t R_OSPL_ReturnFalse(void);

Description	If a warning occurred by writing like:
	`#define IF_D( Condition ) if ( false )`
	Write like:
	`#define IF_D( Condition ) if ( R_OSPL_ReturnFalse() )`

Arguments	None	
Return value	0	

(3) R_UNREFERENCED_VARIABLE

Outline	Suppress the warning of not referenced variable
Header	r_static_an_tag.h
Declaration	#define R_UNREFERENCED_VARIABLE(Variable)
Description	Write this macro after warned.

Arguments	Variable	The variable suppressing the warning
Return value	None	

(4) R_UNREFERENCED_VARIABLE2

Outline	"R_UNREFERENCED_VARIABLE" with 2 arguments
Header	r_static_an_tag.h
Declaration	#define R_UNREFERENCED_VARIABLE2(Variable, Variable2)
Description	

Arguments	Variable	The variable suppressing the warning
	Variable2	The variable suppressing the warning
Return value	None	

(5) R_UNREFERENCED_VARIABLE3

Outline	"R_UNREFERENCED_VARIABLE" with 3 arguments	
Header	r_static_an_tag.h	
Declaration	#define R_UNREFERENCED_VARIABLE3(Variable, Variable2, Variable3)	
Description		
Arguments	Variable	The variable suppressing the warning
	Variable2	The variable suppressing the warning
	Variable3	The variable suppressing the warning
Return value	None	

(6) R_UNREFERENCED_VARIABLE4

Outline	"R_UNREFERENCED_VARIABLE" with 4 arguments	
Header	r_static_an_tag.h	
Declaration	#define R_UNREFERENCED_VARIABLE4(Variable, Variable2, Variable3, Variable4)	
Description		
Arguments	Variable	The variable suppressing the warning
	Variable2	The variable suppressing the warning
	Variable3	The variable suppressing the warning
	Variable4	The variable suppressing the warning
Return value	None	

4.6.20. Multi compiler support

(1) R_OSPL_SECTION

Outline: Names section name to function or variable

Header: r_ospl.h

Declaration:
#define R_OSPL_SECTION(SectionName, Declaration)
#define R_OSPL_SECTION_FOR_ZERO_INIT(SectionName, Declaration)
#define R_OSPL_SECTION_FOR_INLINE(SectionName, Declaration)

Description: This macro names section name to the function or the variable for setting where the function or the variable put at the area in the memory map.
Initializer is written out of "R_OSPL_SECTION" macro.
"R_OSPL_SECTION_FOR_ZERO_INIT" is for the variable without initializer.
"R_OSPL_SECTION_FOR_INLINE" is for inline function.

Example:

```
int  NormalFunction( int a )
{
   return  a + 1;
}

R_OSPL_SECTION( "CODE_BASIC_SECTION",
int  SectionFunction( int a )
)
{
   return  a + 1;
}

R_OSPL_SECTION( "DATA_BASIC_SECTION",
int  g_Variable_Data[100]
)
```

```
= { 1 };
```

Arguments	SectionName	Section name
	Declaration	Declaration of the function or the variable
Return value	Declaration of the function or the variable	

(2) R_OSPL_ALIGNMENT

Outline: Alignments first address of global variable
Header: r_ospl.h
Declaration: #define R_OSPL_ALIGNMENT(ByteCount, Declaration)
Description: Initializer is written out of "R_OSPL_ALIGNMENT" macro.

Example

```
R_OSPL_ALIGNMENT( 0x100,
extern const int  g_Variable_Const2[4] );

R_OSPL_ALIGNMENT( 0x100,
const int  g_Variable_Const2[4] ) = { 0x01, 0x02, 0x03,
0x04 };
```

Arguments	ByteCount	Value of alignment
	Declaration	Declaration of the variable.
Return value	Declaration of the variable	

(3) R_COUNT_OF

Outline: Returns element count of the array
Header: r_ospl.h
Declaration: #define R_COUNT_OF(Array)
Description: Example:

```
uint32_t  array[10];
R_COUNT_OF( array )  // = 10
```

Array argument must not be specified the pointer using like array.
Example:

```
uint32_t   array[10];
func( array );

void  func( uint32_t array[] )  /* "array" is a pointer */
{
    R_COUNT_OF( array )  // NG
}
```

Arguments	Array	Array
Return value	Count of element	

(4) INLINE

"INLINE" macro is same as C99 specification of inline without static and extern.

(5) STATIC_INLINE

"STATIC_INLINE" macro is same as C99 specification of static inline.

(6) R_ADDRESS_Add

Outline	Adds at a value of address (pointer) by byte unit.
Header	r_ospl.h
Declaration	void* R_ADDRESS_Add(void* in_BaseAddress, int_fast32_t in_Offset);

Description	The following code:
	`R_ADDRESS_Add( base_address, +offset )`
	calculate like the following code:
	`( ( (uint8_t*) base_address ) + offset )`
	"in_Offset" argument is by byte unit despite type of pointer.
	This macro prevents forgetting casting because of casting from "uint8_t*" to "uint8_t*" does not feel redundant.
	MISRA-C 2004-17.4 requests writing array, but it is not possible to add by byte unit.
	Checking accessible range before accessing at the address as return value in the similar way to array.

Arguments	void* in_BaseAddress	Augend. Address before adding
	int_fast32_t in_Offset	Addend. Adding value. Minus value is permitted
Return value	Added Address	

(7) R_OSPL_CountLeadingZeros

Outline	Counts bits which is set 0 from most significant bit (MSB).
Header	r_ospl.h
Declaration	int_fast32_t R_OSPL_CountLeadingZeros(bit_flags32_t in_BitFlags);

Description	Return value is 32, if "in_BitFlags" argument was 0.
	"R_OSPL_CountLeadingZeros" is macro, if there is in compiler-specific features.
	Example:
	`R_OSPL_CountLeadingZeros( 0x20000000 ) ==  2;`
	`R_OSPL_CountLeadingZeros( 0x02000000 ) ==  6;`
	`R_OSPL_CountLeadingZeros( 0x00000000 ) == 32;`
	`int_fast32_t  leading_bit_num = 31 -`
	`  R_OSPL_CountLeadingZeros( flags );`

Arguments	in_BitFlags	Investigating value of bit flags
Return value	Count of bits	

(8) R_OSPL_IsSetBitsCount1

Outline	Returns whether there is one bit which is set 1.
Header	r_ospl.h
Declaration	bool_t R_OSPL_IsSetBitsCount1(uint32_t in_BitFlags);

Description	Example:
	`R_OSPL_IsSetBitsCount1( 0x00000001 ) == true;`
	`R_OSPL_IsSetBitsCount1( 0x00000002 ) == true;`
	`R_OSPL_IsSetBitsCount1( 0x00000003 ) == false;`
	`R_OSPL_IsSetBitsCount1( 0x00000000 ) == false;`

Arguments	uint32_t in_BitFlags	Investigating value of bit flags
Return value	Whether there is one bit which is set 1	

4.6.21. Functions for the layer under OSPL

(1) R_DebugBreak

Outline	The function callbacked from OSPL for breaking
Header	r_ospl.h
Declaration	void R_DebugBreak(char_t* File, int_fast32_t Line);
Description	Set a break point at this function. In Release configuration, "File = NULL, Line = 0". If "File = NULL", "Line" argument is error code. This function can be customized by application developer. OSPL's error handling function must not be called in this function. If there was not debug environment and printf output serial communication, LED or oscilloscope watching GPIO shows data instead of serial output. The way is that "R_DebugBreak" function calls function controlling LED and function waiting for a given length of time. Then LED shows variables of "R_DebugBreak" function. For example, the following step shows data. ● Turn on in 1 second and turn off in 1 second at starting LED debug ● If data was 1, turn on in 0.5 second and turn off in 0.5 second ● If data was 0, turn on in 0.2 second and turn off in 0.8 second ● Show binary number by next digit 1 or 0 by shift operation

Arguments	char_t* File	The file name calling this function or NULL
	int_fast32_t Line	The line number calling this function or error code
Return value	None	

(2) R_OSPL_OnIdleDefault

Outline	The default callback function on idle state (for OS less)
Header	r_ospl.h
Declaration	errnum_t R_OSPL_OnIdleDefault(r_ospl_idle_event_t IdleEvent);
Description	Callbacks many times at idle state. This function shows CPU load by calling "R_OSPL_IDLE_Print_CPU_Load" function during periodic intervals, if "R_OSPL_CPU_LOAD" was set to be 1 and "R_OSPL_IDLE_Start_CPU_Load" function was called. However, this function does not show during periodic intervals, if waiting API was not called. This function can be customized by application developer.

Arguments	r_ospl_idle_event_t IdleEvent	The kind about idle state
Return value	Error code. If there is no error, the return value is 0.	

(3) R_OSPL_Start_T_Lock

Outline	The function callbacked from OSPL internal, when T-Lock started
Header	-
Declaration	errnum_t R_OSPL_Start_T_Lock(void);
Description	This function locks by one synchronous object in OSPL. This function must not be called from application. Functions for error handling and debugging must not be called from this function. This function is not callbacked from T-Lock area.

Arguments	None	
Return value	Error code. If there is no error, the return value is 0.	

(4) R_OSPL_End_T_Lock

Outline	The function callbacked from OSPL internal, when T-Lock ended	
Header	-	
Declaration	void R_OSPL_End_T_Lock(void);	
Description	This function must not be called from application. Functions for error handling and debugging must not be called from this function. This function is not callbacked from out of T-Lock area.	
Arguments	None	
Return value	None	

(5) R_OSPL_EVENT_GROUP_Create

Outline	Creates an event group	
Header	r_ospl.h	
Declaration	errnum_t R_OSPL_EVENT_GROUP_Create(volatile r_ospl_event_group_id_t* out_EventGroupId, r_ospl_thread_id_t in_ThreadId);	
Description	This function must not be called from application. There is this function, if "R_OSPL_EVENT_GROUP_CODE" macro was defined to be 1. See section 4.5.4. r_ospl_event_group_id_t.	
Arguments	r_ospl_event_group_id_t * out_EventGroupId	Output: ID of created event group
	r_ospl_thread_id_t in_ThreadId	Thread ID that waits created event group
Return value	Error code. If there is no error, the return value is 0.	

(6) R_OSPL_EVENT_GROUP_Delete

Outline	Delete an event group	
Header	r_ospl.h	
Declaration	errnum_t R_OSPL_EVENT_GROUP_Delete(volatile r_ospl_event_group_id_t in_EventGroupId, r_ospl_thread_id_t in_ThreadId);	
Description	This function must not be called from application. There is this function, if "R_OSPL_EVENT_GROUP_CODE" macro was defined to be 1.	
Arguments	r_ospl_event_group_id_t in_EventGroupId	ID of deleting event group
	r_ospl_thread_id_t in_ThreadId	ID of thread that waits an event group specified at "in_EventGroupId argument".
Return value	Error code. If there is no error, the return value is 0.	

4.6.22. Common functions for driver's APIs

(1) R_DRIVER_Transfer

Outline	Does synchronously the asynchronous operation of the peripheral function
Header	(Driver's header)
Declaration	errnum_t R_DRIVER_Transfer(int_fast32_t ChannelNum, ...);
Description	Call this function from A-thread. This function is synchronous (blocking) function of "R_DRIVER_TransferStart" function. This function waits inside. But this function can be called on OS-using environment or unnecessary other (pseudo) thread running on OS less.

"R_DRIVER_Transfer" is placeholder. Real function name is depending on the driver. e.g. R_FS_Read, R_VSYNC_Wait, R_TOUCH_Read.

This function starts asynchronous operation that runs the hardware or the communication and so on paralleled with CPU and waits at inside until the operation was ended or waits input data for writing next operation to calling "R_DRIVER_Transfer" function.

Internal event is allocated by calling "R_OSPL_EVENT_Allocate" function from inside. The function checks that there is not conflicted event with events for other parallel operation.

In OS less, "R_DRIVER_Transfer" function calls many times the function of "r_ospl_idle_callback_t" type while waiting.

In OS less or for reduce OS thread count (the total size of stack memory and so on), call "R_DRIVER_TransferStart" function and build pseudo multi-threading.

Arguments	int_fast32_t ChannelNum	The channel number. It can be changed by the API specification.
	...	The parameters. It can be changed by the API specification.
Return value	Error code. If there is no error, the return value is 0. It can be changed by the API specification.	

(2) R_DRIVER_TransferStart

Outline	Starts the asynchronous operation of the peripheral function
Header	(Driver's header)
Declaration	errnum_t R_DRIVER_TransferStart(int_fast32_t ChannelNum, ..., r_ospl_async_t* Async);
Description	Call this function from A-thread. This function is asynchronous (non-blocking) function of "R_DRIVER_Transfer" function. "R_DRIVER_TransferStart" is placeholder. Real function name is depending on the driver. e.g. R_FS_ReadStart, R_SOUND_Play, R_VSYNC_WaitStart, R_TOUCH_ReadStart Refer to "r_ospl_async_t" type structure how to receive the end of asynchronous operation. **Check the value of "r_ospl_async_t::ReturnValue" member variable, when the asynchronous operation was ended.** Keep the memory area of the structure passed "Async" argument until the asynchronous operation was ended. Don't pass the structure to "Async" argument of other asynchronous operation until the asynchronous operation was ended. Some driver may be able to start multi asynchronous operation by passed other "Async" structure to "Async" argument. If it is not possible, the function returns error code. Some drivers continue asynchronous input operation after A-event received for not lost continuous input from input hardware by starting to receive input event by "R_DRIVER_TransferStart" function. In that case, keep the memory area of the structure passed "Async" and not use bits used for A-event or I-event of other target until finished input.

(This function was the name called "R_DRIVER_TransferAsync".)

Refer to 4.7.1. 4.7.4. The figure of sequence

Arguments	int_fast32_t ChannelNum	The channel number. It can be changed by the API specification.
	...	The parameters. It can be changed by the API specification.
	r_ospl_async_t* Async	Input/Output: Setting of notifications. NULL is not permitted
Return value	Error code. If there is no error, the return value is 0. It can be changed by the API specification.	

(3) R_DRIVER_OnInterrupting

Outline	Receives the interrupt
Header	(Driver's header)
Declaration	errnum_t R_DRIVER_OnInterrupting(const r_ospl_interrupt_t* InterruptSource);
Description	This function sends the interrupt notification from the interrupt status register to "r_driver_async_status_t::InterruptFlags" variable and clears the interrupt status in the register. "R_DRIVER_OnInterrupting" is placeholder. Real function name is depending on the driver. Normally, this function was called automatically from the default interrupt callback function provided by the driver. The interrupt response operation is usually done in "R_DRIVER_OnInterrupted" function. But there is sometimes the specification that "R_DRIVER_OnInterrupting" function does the interrupt response operation.

Arguments	r_ospl_interrupt_t* InterruptSource	The source of interrupt. It can be changed by the API specification.
Return value	Error code. If there is no error, the return value is 0. It can be changed by the API specification.	

(4) R_DRIVER_OnInterrupted

Outline	Does the interrupt response operation
Header	(Driver's header)
Declaration	errnum_t R_DRIVER_OnInterrupted(int_fast32_t ChannelNum);
Description	This function clears bits of "r_driver_async_status_t::InterruptFlags" variable to 0 that bits were set by "R_DRIVER_OnInterrupting" function to 1 and responds the interrupt. "R_DRIVER_OnInterrupted" is placeholder. Real function name is depending on the driver. When the asynchronous operation was ended, the A-event is signaled. The value of "r_ospl_async_t::ReturnValue" variable must be checked, when the A-event was received, If the asynchronous operation was started with "r_ospl_async_t::I_Thread == R_OSPL_THREAD_NULL", this function is called automatically from the default interrupt callback function provided by the driver.

If the asynchronous operation was started with "r_ospl_async_t::I_Thread != R_OSPL_THREAD_NULL", the I-event is signaled by the default interrupt callback function. This function must be called after receiving the I-event.

When any interrupts were not signaled, this function does nothing and returns 0 (no error).

There is sometimes not provided "R_DRIVER_OnInterrupted" function, if the interrupt response operation almost did nothing.

Arguments	int_fast32_t ChannelNum	The channel number. It can be changed by the API specification.
Return value	Error code. If there is no error, the return value is 0. It can be changed by the API specification.	

(5) R_DRIVER_GetAsyncStatus

Outline	Get the pointer to the structure indicated the status of interrupts and the asynchronous operation
Header	(Driver's header)
Declaration	errnum_t R_DRIVER_GetAsyncStatus(int_fast32_t ChannelNum, const r_driver_async_status_t** out_Status);
Description	"R_DRIVER_GetAsyncStatus" is placeholder. Real function name is depending on the driver.

The pointer variable passed to "out_Status" argument must be with "const" qualifier. This function can be called from the thread and the interrupt callback function. This function can be called, if the driver had not been initialized yet.

The value of **out_Status cannot be modified from the application. The driver modifies it. It is depending on the member variable's specification whether it is modified by calling this function or not calling.

Arguments	int_fast32_t ChannelNum	The channel number. It can be changed by the API specification.
	r_driver_async_status_t** out_Status	Output: The pointer to the structure about the status of interrupt and asynchronous operation. This is defined by the driver.
Return value	Error code. If there is no error, the return value is 0 It can be changed by the API specification.	

(6) R_DRIVER_Initialize

Outline	Initializes the driver and changes driver's state to usable
Header	(Driver's header)
Declaration	errnum_t R_DRIVER_Initialize(int_fast32_t ChannelNum, r_driver_config_t in_out_Config);
Description	This section describes channel using management (lock) only.

"R_DRIVER_Initialize" function locks initialized channel. However, "R_DRIVER_LockChannel" function locks the channel without initialization. Other drivers can use a non-competitive channel that was locked without initialization. Because not used channel list is managed.

This function returns "E_ACCESS_DENIED" error or "E_FEW_ARRAY" error or others, if lock operation was failed.

The module must raise an error or change to limitation mode, if the module failed to lock a channel.

The following code initializes specified number channel:

```
e= R_DRIVER_Initialize( 1, NULL );
    IF(e){goto fin;}
```

The following code initializes not used channel and gets the number of channel:

```
    int_fast32_t      channel_num = R_OSPL_UNLOCKED_CHANNEL;
    r_driver_config_t  config;

    e= R_DRIVER_Initialize( channel_num, &config );
       IF(e){goto fin;}
    channel_num = config.ChannelNum;
```

Arguments	int_fast32_t ChannelNum	Locking and initializing channel number or "R_OSPL_UNLOCKED_CHANNEL" It can be changed by the API specification.
	r_driver_config_t in_out_Config	Input/Output: all setting for initialization. NULL is permitted. It can be changed by the API specification.
Return value	Error code. If there is no error, the return value is 0 It can be changed by the API specification.	

(7) R_DRIVER_Finalize

Outline	Finalizes the driver
Header	(Driver's header)
Declaration	errnum_t R_DRIVER_Finalize(int_fast32_t ChannelNum, errnum_t e);
Description	This section describes channel using management (lock) only.

This function finalizes the channel and unlocks it.

The channel is not finalized and not unlocked, if the owner (thread or context) not called "R_DRIVER_Initialize" called "R_DRIVER_Finalize".

Arguments	int_fast32_t ChannelNum	Finalizing and unlocking channel number. It can be changed by the API specification.
	errnum_t e	Errors that have occurred. No error = 0 It can be changed by the API specification.
Return value	Error code or e 0 = successful and "e = 0"	

(8) R_DRIVER_LockChannel

r	Locks a channel (Changes to used state in the driver)
Header	(Driver's header)
Declaration	errnum_t R_DRIVER_LockChannel(int_fast32_t const ChannelNum, int_fast32_t* out_ChannelNum);
Description	This function calls "R_OSPL_LockChannel" and makes the specified channel available to other drivers. If the driver having "R_OSPL_LockChannel" function was used, standard initialize function (e.g. "R_DRIVER_Initialize") must be called.

This function does not initialize the specified channel of the peripheral unit.

If "ChannelNum" argument was specified to "R_OSPL_UNLOCKED_CHANNEL" (=0xFEE), unlocked channel is locked. In this case, "out_ChannelNum" argument cannot be specified to "NULL".

Arguments	int_fast32_t ChannelNum	Locking channel number or "R_OSPL_UNLOCKED_CHANNEL" It can be changed by the API specification.
	int_fast32_t* out_ChannelNum	Output: Locked channel number, NULL is permitted.
Return value	Error code. If there is no error, the return value is 0 It can be changed by the API specification.	

(9) R_DRIVER_UnlockChannel

Outline	Unlocks a channel (Changes to not used state in the driver)
Header	(Driver's header)
Declaration	errnum_t R_DRIVER_UnlockChannel(int_fast32_t const ChannelNum, errnum_t e);
Description	This function calls "R_OSPL_UnlockChannel" and makes the specified channel available to the driver having "R_DRIVER_UnlockChannel" function. This function does not finalize the specified channel of the peripheral unit. "E_ACCESS_DENIED" error is raised, if the specified channel was already unlocked. This function does nothing and returns 0 (no error) (if argument e=0), if "ChannelNum" argument was "R_OSPL_UNLOCKED_CHANNEL", less than 0, channel count or more.

Arguments	int_fast32_t ChannelNum	Unlocking channel number, It can be changed by the API specification.
	errnum_t e	Errors that have occurred. No error = 0 It can be changed by the API specification.
Return value	Error code or e 0 = successful and "e = 0"	

4.6.23. Common functions under the driver

(1) R_DRIVER_SetDefaultAsync

Outline	Sets default value in "r_ospl_async_t" type	
Header	(Driver's internal header)	
Declaration	void R_DRIVER_SetDefaultAsync(r_ospl_async_t* Async, r_ospl_async_type_t AsyncType);	
Description	The default settings set by this function can be changed for target system. "R_DRIVER_SetDefaultAsync" is placeholder. Real function name is depending on the driver. This function calls "R_OSPL_ASYNC_SetDefaultPreset". e.g. R_FS_SetDefaultAsync.	
Arguments	r_ospl_async_t* Async	Input/Output: Setting of notifications. NULL is not permitted
	r_ospl_async_type_t AsyncType	Type of asynchronous operation
Return value	None	

(2) R_DRIVER_I_LOCK_Replace

Outline	Replaces I-lock object to the integrated driver's I-lock object.	
Header	(Driver's internal header)	
Declaration	bool_t R_DRIVER_I_LOCK_Replace(int_fast32_t ChannelNum, void* i_lock, r_ospl_i_lock_vtable_t* i_lock_v_table);	
Description	This is an API provided the driver which may be integrated by an integrated type driver. This is called from the integrated type driver except for the application. This aggregate the operation of enabling and disabling an interrupt by I-lock.	
Arguments	int_fast32_t ChannelNum	Channel number
	void* i_lock	Value of first argument with member functions in "i_lock_v_table" argument with this function or NULL
	r_ospl_i_lock_vtable_t* i_lock_v_table	VTable related I-lock or NULL
Return value	Whether I-lock object was replaced or not	

(3) R_DRIVER_DisableInterrupt

Outline	Disables an interrupt for staring I-lock.	
Header	(Driver's internal header)	
Declaration	bool_t R_DRIVER_DisableInterrupt(int_fast32_t ChannelNum);	
Description	This is an API provided the driver which may be integrated by an integrated type driver. This is called from the integrated type driver except for the application.	
Arguments	int_fast32_t ChannelNum	Channel number
Return value	Whether the interrupt was enabled or not	

(4) R_DRIVER_EnableInterrupt

Outline	Enables an interrupt for ending I-lock	
Header	(Driver's internal header)	
Declaration	void R_DRIVER_EnableInterrupt(int_fast32_t ChannelNum);	
Description	This is an API provided the driver which may be integrated by an integrated type driver. This is called from the integrated type driver except for the application.	
Arguments	int_fast32_t ChannelNum	Channel number
Return value	None	

4.6.24. Other functions

(1) Not recommended functions

The following functions are for ensuring portability with old OSPL specification or other specification. They are not recommended to be used.

R_OSPL_EVENT_Get, R_OSPL_EVENT_OBJECT_Create, R_BSP_InterruptsEnable, R_BSP_InterruptsDisable

(2) half internal functions

The following functions are just common operation, not ensured portability or allow to be called from driver for debugging.

R_OSPL_ASYNC_CopyExceptAThread, R_OSPL_I_LOCK_LockStub, R_OSPL_I_LOCK_UnlockStub, R_OSPL_I_LOCK_RequestFinalizeStub, R_OSPL_LockCurrentThreadError, R_OSPL_UnlockCurrentThreadError

(3) Print function

The following functions show internal status. It is not recommended to call in your product.

R_OSPL_TABLE_Print, R_OSPL_QUEUE_Print

4.7. Figure of sequence

4.7.1. The interrupt response operation - Synchronous type (Responds by interrupt context)

Figure 4-3 shows the interrupt response operation using a synchronous function (Responds by interrupt context). In OS less, the interrupt callback function calls "R_DRIVER_OnInterrupted" function.

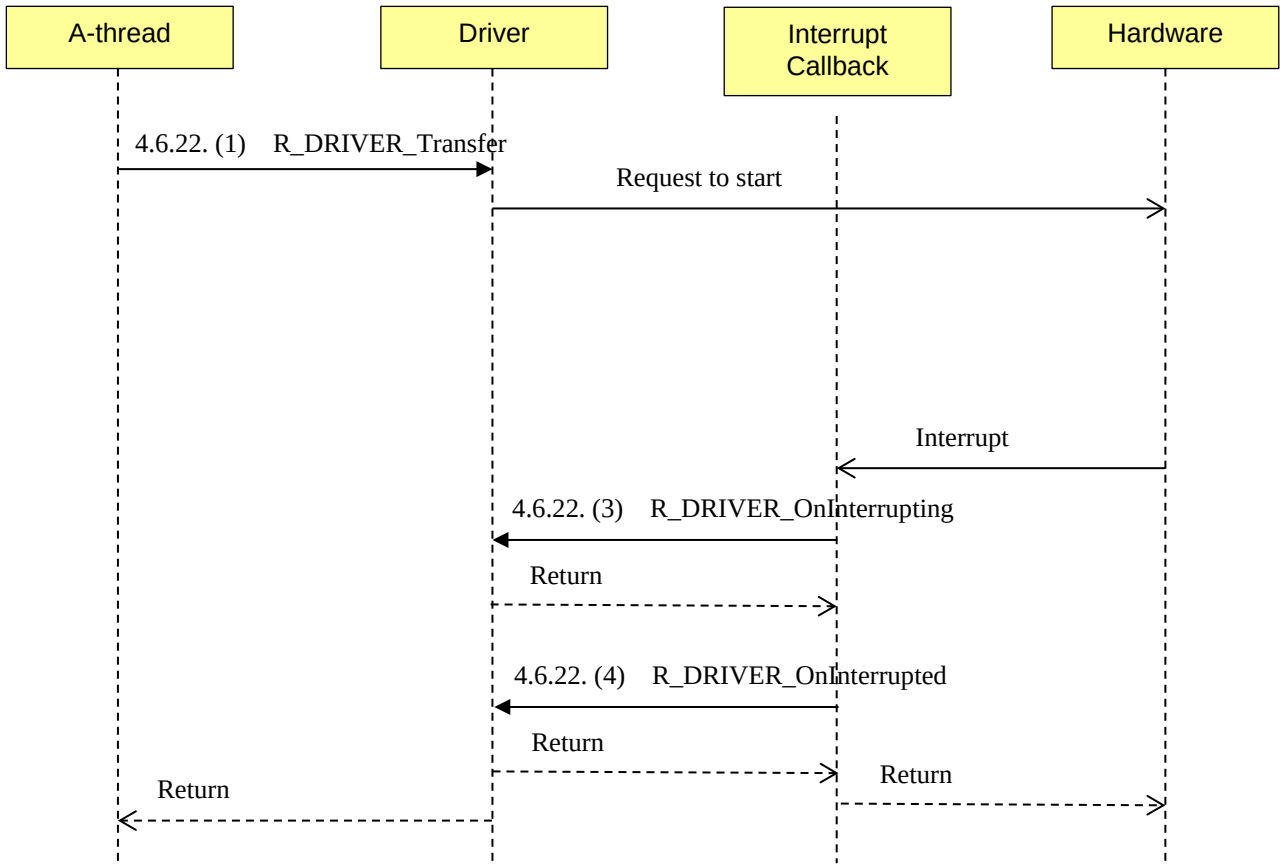


Figure 4-3 the interrupt response operation using a synchronous function (Responds by interrupt context)

4.7.2. The interrupt response operation - Synchronous type (Responds by A-thread)

Figure 4-4 shows the interrupt response operation using a synchronous function (Responds by A-thread). In OS-using environment, the driver usually calls "R_DRIVER_OnInterrupted" function from "R_DRIVER_Transfer" function running on A-thread not from the interrupt callback function (a interrupt context) for running high priority thread, if the interrupt response operation in the driver takes long time.

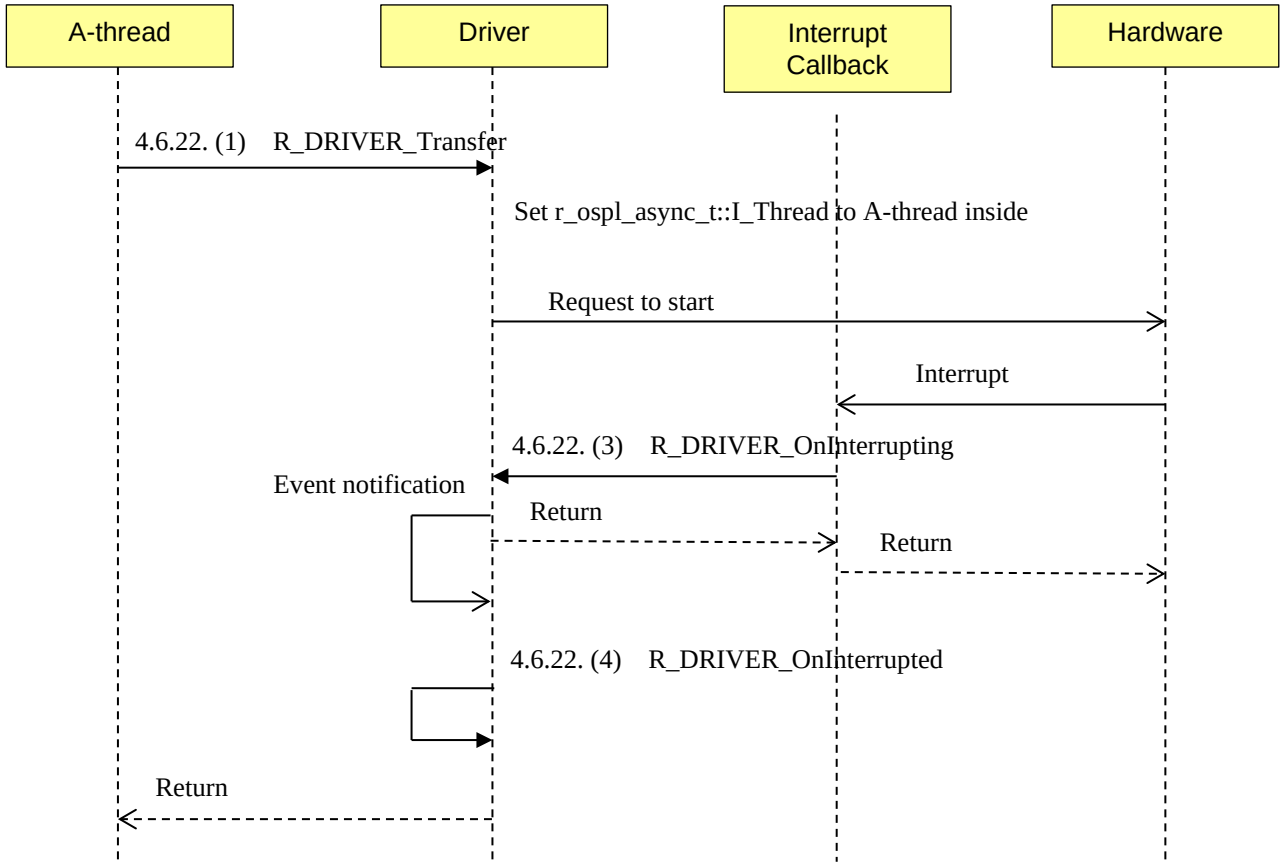


Figure 4-4 the interrupt response operation using a synchronous function (Responds by A-thread)

4.7.3. The interrupt response operation - Asynchronous type (Not I-thread)

Figure 4-5 shows the interrupt response operation using an asynchronous function (r_ospl_async_t::I_Thread==R_OSPL_THREAD_NULL). In OS-using environment, the driver may have same flow using an I-thread created in the driver inside as 4.7.4. The interrupt response operation - Asynchronous type (With I-thread).

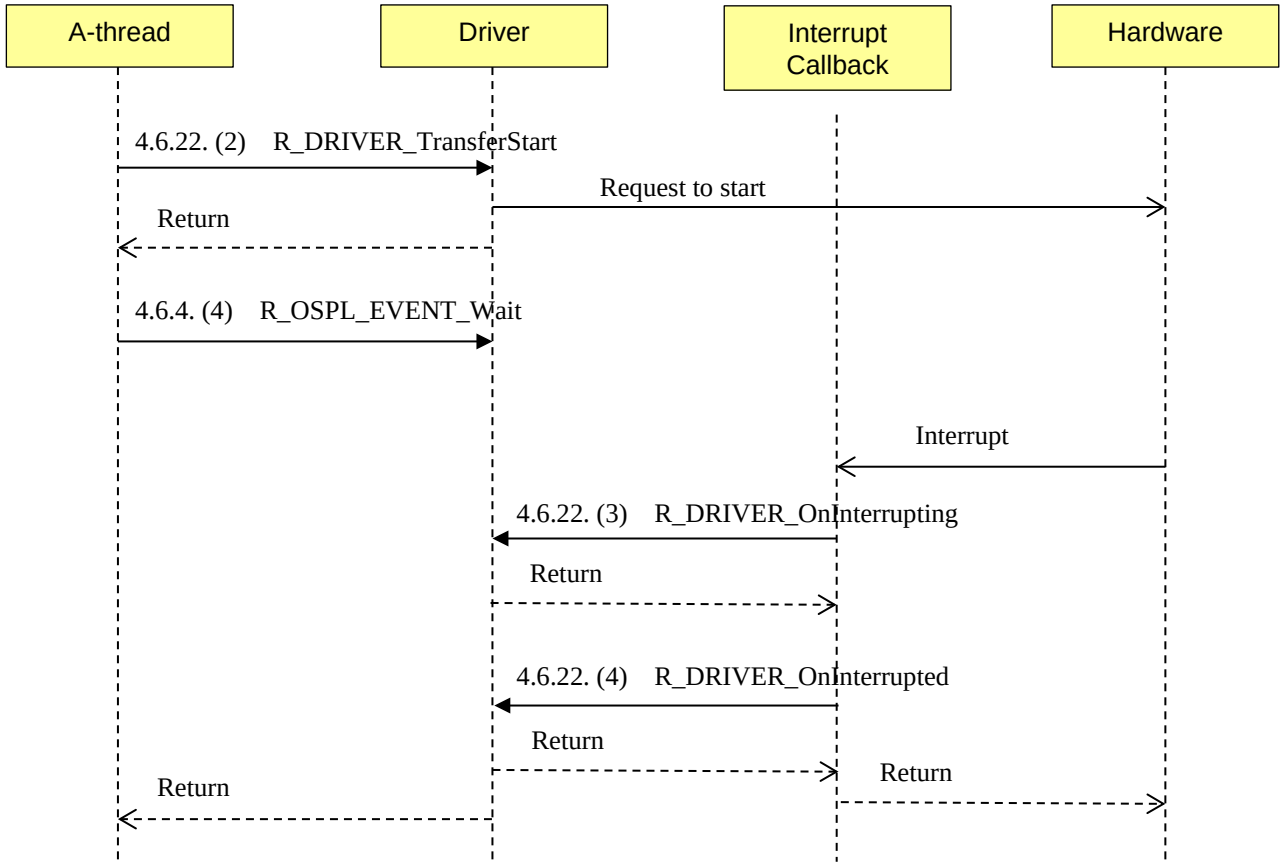


Figure 4-5 the interrupt response operation using an asynchronous function (r_ospl_async_t::I_Thread==R_OSPL_THREAD_NULL)

4.7.4. The interrupt response operation - Asynchronous type (With I-thread)

Figure 4-6 shows the interrupt response operation using an asynchronous function (r_ospl_async_t::I_Thread!=R_OSPL_THREAD_NULL). In OS-using environment, the high priority thread should be set to "r_ospl_async_t::I_Thread" explicitly, if the I-thread created in the driver inside runs, even if "r_ospl_async_t::I_Thread==R_OSPL_THREAD_NULL".

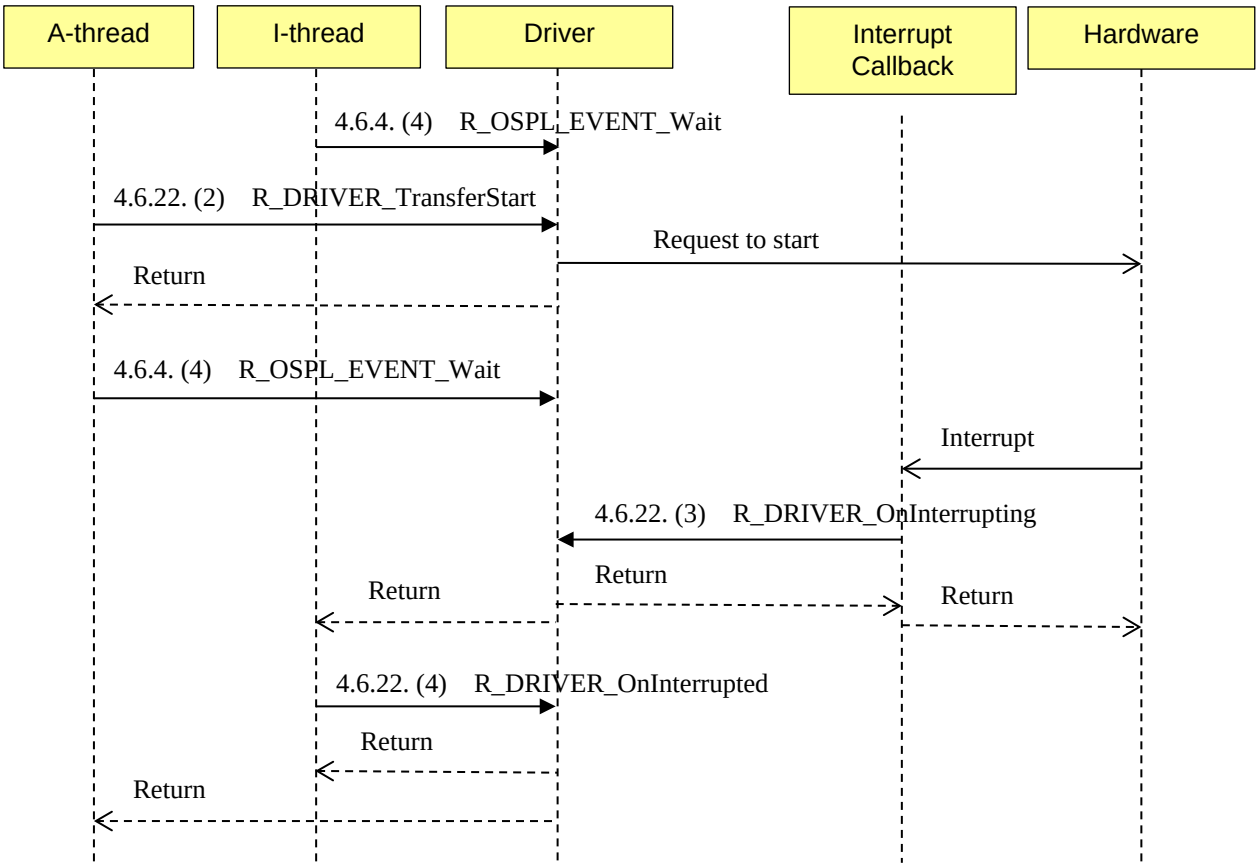


Figure 4-6 the interrupt response operation using an asynchronous function (r_ospl_async_t::I_Thread!=R_OSPL_THREAD_NULL)

4.8. Supplementary Information

4.8.1. Selecting the target (use_list.h, mcu_board_select.h)

OSPL supports some OSs and base software (e.g. BSP, board support package). Please, change between to define or not to define macro in "use_list.h" file and change the value of define in "mcu_board_select.h" file.

But OSPL does not support all pattern of OS and base software. Test pattern are the following sets. Enumerated symbols in patterns are defined without value or value of "#define".

- TARGET_RZA1H, USE_LIST_RZA1H_OS_LESS
- TARGET_RZA1H, USE_LIST_RZA1H_BSP, USE_LIST_RTX

4.8.2. Flagged structure parameters

Flags member variables in the structure are used as bit flags, and if a bit is 1, the corresponding member variable is enabled according to the coding pattern. If a bit is 0, the value of the member variable is assumed to be the default value. Even if the version is upgraded so that its structure contains additional members, the old and new versions can be binary compatible.

```
FuncA_ConfigClass  config;

config.Flags  = F_FuncA_Param1 | F_FuncA_Param2;
config.Param1 = 10;
config.Param2 = 2;
FuncA( &config );
```

Because there is not Flags |= F_FuncA_Param3, config.Param3 is the default value.

4.8.3. Nested Interrupt

Any interrupts are not signaled in area of disabling all interrupt, even if nested interrupt was supported.

Only a timer has potential to be signaled, but it is not effect to code driver and application.

It is possible to signal high priority interrupt of driver B as nested interrupt from I-lock area in driver A or interrupt callback function. But driver B cannot call API of driver A from interrupt context even if indirectly.

"R_OSPL_DisableAllInterrupt" function maximizes interrupt mask level after saving current interrupt mask level. "R_OSPL_EnableAllInterrupt" function returns interrupt mask level to the saved level.

4.8.4. OS porting guide

When you support OSPL for OS to new OS, you should port based on already exist OSPL for OS.

When you support OSPL for OS less to new board, you should port based on already exist OSPL for OS less.

Ported module can be identified as OSPL, even if not all API was supported.

Please, port attached test program, if your test used it.

This document does not explain the guide of developing the driver.

(1) Internal functions in OSPL and drivers

When you supported new OS or new board for OS less, you should port following functions in OSPL and the porting layer in drivers.

- R_BSP_InterruptWrite
- R_BSP_InterruptRead
- R_BSP_InterruptControl

When you support OSPL to new OS, you should port following functions.

- "R_OSPL_Start_T_Lock" function
- "R_OSPL_End_T_Lock" function

(2) Thread

About functions for threads (4.6.1. (2)), only "R_OSPL_THREAD_GetCurrentId" function is common API for different OS. The other functions are not provided as common API for different OS.

When you support OSPL to new OS, you should port "R_OSPL_THREAD_GetCurrentId" function.

When you port from with OS to without OS, functions that run parallel should be support pseudo multi-threading.

(3) Event

About functions for thread attached events (4.6.1. (3)), all functions are common API for different OS.

When you support OSPL to new OS, you should port all OSPL functions.

(4) Disabled all interrupts area

About functions for the area disabled all interrupts (4.6.1. (8)), all functions are common API for different OS.

When you support OSPL to new OS or new compiler, you should port all OSPL functions.

(5) Memory

About functions for the memory (4.6.1. (13)), all functions are common API for different OS.

When you support OSPL to new OS or new board, you should port all OSPL functions.

When memory map was changed, you should change the implement of functions, too. Cached area and uncached area is described in section 4.8.9.

Regarding the detail of AXI bus, refer to: RZ/A1H Group User's Manual: Hardware (5.8) AXI Protocol Control Signals.

(6) Time

About functions for time (0), all functions are common API for different OS.

When you support OSPL to new OS, new board, you should port all OSPL functions.

API must implement using timer hardware. It is not possible to implement by software only.

OSPL must use different timer that is used by OS and application. OSPL will not return from API function referring the timer inside, if the timer was shared.

This free running timer can be implemented by OS timer, only when there was no timer resource by using OS. In this case, define "R_OSPL_FTIMER_IS" macro "R_OSPL_FTIMER_IS_OS_TIMER_INTERRUPT".

(7) Regarding idle state

About functions for the idle state (0), all functions are API for OS less only.

Even if you supported software to new board, there is no need to port OSPL API functions.

In OS-using environment, the lowest priority thread is considered the idle state thread.

(8) Interrupt callback functions

About functions for interrupt callback functions (0), all functions are common API for different OS.

When you support OSPL to new OS or new board, you should port the code of calling the interrupt callback function from the interrupt hander, because the interrupt handler under the driver is not same.

(9) Regarding queue

Functions related with queue 4.6.1. (7) is common API, even if OS was supported to other.

Change implement of API function defined by OSPL, when OSPL is supported to new OS.

(10) Multi compiler support

Multi compiler support 4.6.1. (19) takes in each function and syntax between compilers.

Change implement, when OSPL is supported to new compiler.

4.8.5. Application Porting Guide

There is RTOS related function as a part of OSPL. This section describes related API as a reference of porting application of RTOS function.

The following table describes almost same functions between OSPL and CMSIS-RTOS and describes related µITRON functions.

CMSIS	OSPL	µITRON
Signal	Thread attached event *1	Event flag and task event flags (implementation specific)
Mail Queue	r_ospl_queue_id_t	Fix length memory pool + (data queue or mail box)
Generic Wait	R_OSPL_Delay	dly_tsk

*1 Added allocation and free API for detected conflicting with event.

The following table describes functions of OSPL closing to functions of CMSIS. It is necessary to port application or develop wrapper.

CMSIS	OSPL
Thread	To create and delete thread + r_ospl_thread_id_t
Timer	To create and delete thread + free running timer + R_OSPL_Delay
Mutex	r_ospl_c_lock_t or r_ospl_queue_id_t *2
Semaphore	r_ospl_c_lock_t or r_ospl_queue_id_t *2
Message Queue	r_ospl_queue_id_t
Memory Pool	r_ospl_table_t

*2 Split accessible data between threads which send to a queue and receive from the queue like exclusive control.

4.8.6. Body of inline function (inline_body.c)

When an inline function is not expanded, there is a compiler that automatically generates the entity of the function and the compiler that the user must explicitly generate.

Depending on the compiler options, they may not all be expanded. Without the entity of the function when it was not expanded, the linker prints an error message that is similar to the following:

```
undefined reference to `R_OSPL_THREAD_ExitWaiting'
```

You can explicitly generate a function entity by writing including header file code at below `#include "r_ospl.h"` in ospl\porting\inline_body.c file. `R_OSPL_MAKE_INLINE_BODY` is defined at the beginning of the inline_body.c. By that the definition of the INLINE macro (4.6.20. (4)) changes to always generates the entity of the function.

Depending on the compiler, __STDC_VERSION__> = 199901 L becomes true even though it does not conform to the inline function specification of C99, and #if above the INLINE macro definition may not be determined as expected. In that case, it is better to define the INLINE macro without determining the compiler type with #if.

4.8.7. Reducing footprint

Memory size (footprint) can be reduced by disabling debugging function of OSPL.

Set the following symbols, remove *.c file of OSPL temporarily, remove application sources except one application's source and copy functions and variables which was raised link error at rebuilding from removed *.c file.

- R_OSPL_NDEBUG = 1
- R_OSPL_ERROR_BREAK = 0
- R_OSPL_TLS_ERROR_CODE = 0 (Set 1, if "R_OSPL_GetErrNum" function is used)
- R_OSPL_DEBUG_TOOL = 0
- R_OSPL_EVENT_OBJECT_CODE = 0
- R_OSPL_DETECT_BAD_EVENT = 0
- R_OSPL_TLS_EVENT_CODE = 0
- R_OSPL_CPU_LOAD = 0 (For OS less)

4.8.8. How to use the driver with interrupt handler

There are sometimes drivers that the specification of the driver is that the application makes the interrupt handler. OSPL calls the driver "driver with interrupt handler". This driver may have the specification that the application does exclusive control.

The APIs written in this document (e.g. 0R_OSPL_SetInterruptPriority) does not depend on various OS or with/without OS by abstract OS layer. The driver with interrupt handler does not depend on various OS or with/without OS by placing the module depending on the OS out of the driver.

If the application or the middle-ware used the driver with interrupt handler, the following point must be checked.

- With OS and without OS, some global variables must be used for receiving data from the interrupt handler. In the case, "volatile" qualifier must be written with global variables. If receiving code was written in the interrupt handler, it is necessary to make "volatile" global variable related to threads, events and arguments or register's value that is not able to get after clearing the interrupt (end of interrupt handler) at least.

- With OS and without OS, the code of I-Lock's exclusive control must be written, if necessary. (See glossary)

- If the application supported OS, too, it is necessary to reduce the code in the interrupt handler, because the highest priority thread can response soon. Specifically, only the code of set event is written in the interrupt handler. The code of response the interrupt is written in the function receiving the event. Also, the event must be clear before starting the asynchronous operation. (The application with OS for exclusive use and unit test are not necessary to support it.)

- When the application was ported with OS, codes of calling driver's API should be written in T-Lock's exclusive control. Otherwise, codes of calling driver's API should be written after checking that the current thread is expected only one thread.

Example: the code of supported OS and without OS using the driver with interrupt handler

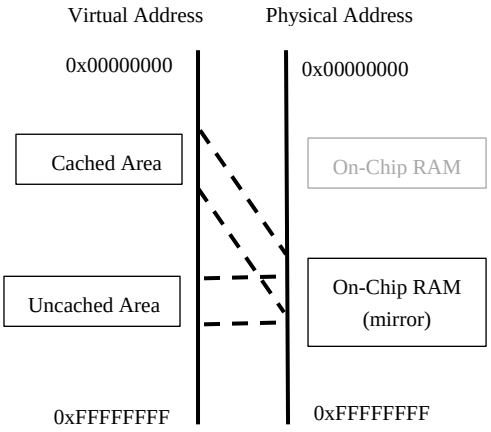
```
enum { R_DRIVER_EVENT_A = 0x0010 };
volatile r_ospl_thread_id_t    g_App_Thread;
volatile r_driver_interrupt_t  g_App_InterruptParameter;

int  main()
{
   g_App_Thread = R_OSPL_THREAD_GetCurrentId();
   R_OSPL_EVENT_Clear( g_App_Thread, R_DRIVER_EVENT_A | R_DRIVER_EVENT_B );
   R_DRIVER_RegisterISR( R_DRIVER_INTERRUPT, App_InterruptCallback );
   R_DRIVER_RegisterISR( R_DRIVER_ERROR_INTERRUPT,
      App_ErrorInterruptCallback );

   for (;;) {
      e= R_OSPL_EVENT_Wait( R_OSPL_ANY_FLAG, &got_flags,
         R_OSPL_INFINITE ); IF(e){goto fin;}

      /* Start I-Lock for R_DRIVER */

      #if R_OSPL_IS_PREEMPTION
         /* Start T-Lock for R_DRIVER */
      #endif

      if ( IS_BIT_SET( got_flags, R_DRIVER_EVENT_A ) ) {
```

```
            e = R_DRIVER_GetAsyncResult(); IF(e){goto fin;}
              :

            /* Write the code written in the interrupt handler here */
        }
        if ( IS_BIT_SET( got_flags, R_DRIVER_EVENT_B ) ) {
            e = R_DRIVER_GetAsyncResult(); IF(e){goto fin;}
              :
        }

        #if R_OSPL_IS_PREEMPTION
            /* End T-Lock and/or I-Lock  for R_DRIVER */
        #endif

        /* End I-Lock for R_DRIVER */
    }
}

void  App_InterruptCallback( int Argument )
{
    R_DRIVER_GetParameterOnInterrupting( Argument,
        &g_App_InterruptParameter );
    R_OSPL_EVENT_Set( g_App_Thread, R_DRIVER_EVENT_A );
}

void  App_ErrorInterruptCallback()
{
    R_OSPL_RaiseUnrecoverable( ERROR_CODE );
}
```

4.8.9. Pattern of mapping cached area and uncached area (RZ/A1)

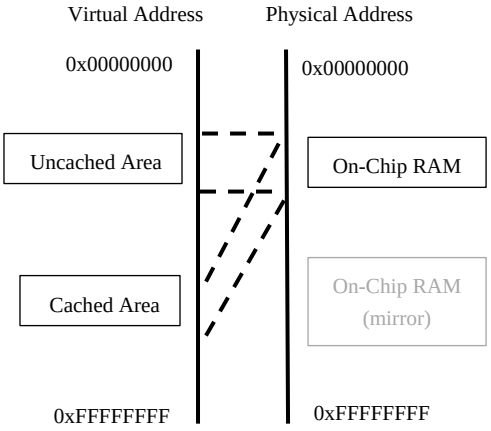
Please, change the function to define memory map in "r_ospl_memory.c" file, if you changed memory map of cached area and uncached area set by TTB (MMU) in CPU. Also, you can change some patterns of the memory map by value of "GS_RZ_A1_MMU_TYPE". Supported memory map are described in the following list.

1. GS_RZ_A1_MMU_TYPE_IS_SHIFTED_MIRROR: The virtual address (pointer in the software) of cached area and uncached area is mapped at mirror area of On-Chip RAM. The virtual address of uncached area is same as the physical address. This memory map cannot be selected in RZ/A1M, because it does not have mirror area of On-Chip RAM.



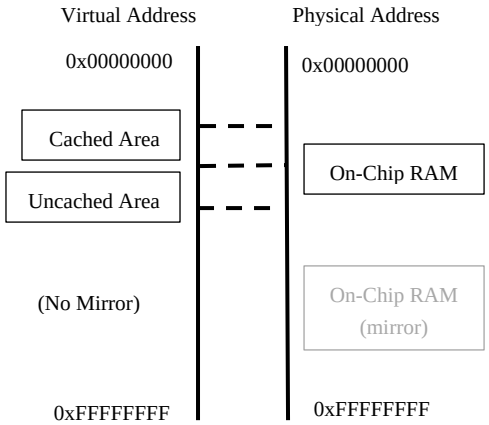
1. GS_RZ_A1_MMU_TYPE_IS_SHIFTED_MIRROR

2. GS_RZ_A1_MMU_TYPE_IS_ REVERSED _MIRROR: The virtual address (pointer in the software) of cached area and uncached area is mapped at On-Chip RAM (not mirror). The virtual address of uncached area is same as the physical address.



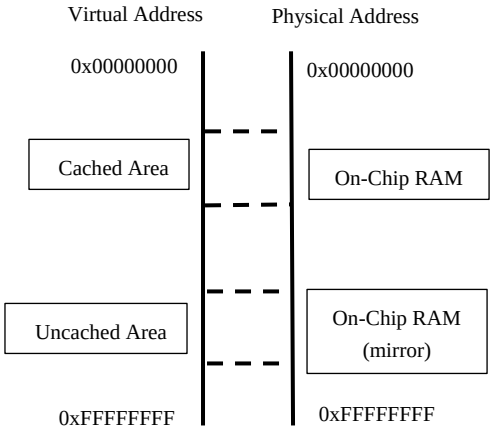
2. GS_RZ_A1_MMU_TYPE_IS_REVERSED_MIRROR

3. GS_RZ_A1_MMU_TYPE_IS_CACHE_UNCACHE: The virtual address (pointer in the software) of cached area is mapped at low address of On-Chip RAM (not mirror) and uncached area is mapped at high address. The virtual address is same as the physical address. It is not possible to change between virtual address and physical address.



3. GS_RZ_A1_MMU_TYPE_IS_CACHE_UNCACHE

4. GS_RZ_A1_MMU_TYPE_IS_CROSSED_MIRROR: The virtual address (pointer in the software) of cached area is mapped at On-Chip RAM (not mirror) and uncached area is mapped at mirror area of On-Chip RAM. The virtual address is same as the physical address. It is possible to change between virtual address and physical address at the first 1MB and last 1MB of On-Chip RAM in RZ/A1H. This memory map cannot be selected in RZ/A1M, because it does not have mirror area of On-Chip RAM.



4. GS_RZ_A1_MMU_TYPE_IS_CROSSED_MIRROR

4.9. Glossary

A-event	Event receiving the end of asynchronous operation and waking up A-thread.
A-thread	Thread in the application. E.g. main thread. Thread set to "r_ospl_async_t::A_Thread".
C-lock	What a thread holds exclusively a context (the current status of driver, middleware or hardware). If the competition was occurred, an error is raised without waiting. Lock and unlock operations must do exclusive control with internal lock state variable at inside, because two or more threads have potential to start to lock or unlock at the same time. T-lock usually locks in each function. C-lock usually locks from initializing to finalizing or from opening to closing. Even if in multi-thread environment (and pseudo multi-thread environment), the context set by current thread is not changed by other thread without exclusive control. Refer to: (4.5.12.) r_ospl_c_lock_t, (4.6.22. (6)) R_DRIVER_Initialize, T-lock, I-lock
Channel	1 or more numbers channel in a unit of the peripheral.
Context	The area that contains current setting values. In OSPL, context means context related a thread. Interrupt context means context when CPU is interrupt mode. A context is attached by a thread or an interrupt. The context is not always attached the thread only or the interrupt only. Even on one channel, when a series of operations are performed in parallel on the ont channel, context handling code is necessary. Note that parallel processing is not only between threads, but also at each stage of interrupt (multiple interrupt stage).
Event	Refer to 4.6.1. (3) Functions for thread attached events .
I-event	Event receiving the notification need to do the interrupt response operation and waking up I-thread.
I-thread	Thread to do the interrupt response operation. Thread set to "r_ospl_async_t::I_Thread". I-thread calls "R_DRIVER_OnInterrupted" function. In OS-using environment, I-thread may be the thread created in the driver. In OS less, A-thread does pseudo multi-threading. In "R_DRIVER_Transfer" function, A-thread does the interrupt response operations instead of I-thread. The role of I-thread is almost same as IST (Interrupt Service Thread).
I-Lock	Exclusive control between operations on the thread and operations on the interrupt context (the interrupt handler) by disabling the interrupt managed by the driver. Example: It may be necessary to I-Lock for keeping consistent between internal status variables in the driver or registers and status of the peripheral function, when both the operation of stopping the peripheral function from a thread and the operation of restart the peripheral function from an interrupt handler were done at the same time.

	When an interrupt signaled with masked the interrupt (in I-lock area), normally the interrupt is pended. In the state, when the interrupt was unmasked, CPU jumps to the interrupt vector soon. Refer to T-Lock, C-lock.
Interrupt callback function	Function callbacked from the interrupt handler.
Interrupt handler	Function callbacked when the interrupt was signaled. This is registered by calling OS or API of the target board. This calls the interrupt callback function with adapting different from OS or API for target board. This is sometimes called ISR (Interrupt Service Routines). See (4.8.8.) How to use the driver with interrupt handler
Pseudo multi-threading	Emulation of multi-threading doing operations for each thread context by calling from 1 message loop. Even if no time sharing, OS that can preempt high priority thread is not pseudo multi-threading. The module supported pseudo multi-threading cannot wait in API function. Operations for other thread context must be executed while waiting in the synchronous function, it is necessary to return to the message loop (main function) by doing following operation. ・ Separate a step function before waiting (polling) and a step function the waiting (polling) operation and later operations. ・ Change to checking without polling or waiting with time out 0 ・ Save local variables in the function and the function number (or function pointer) to the context of a pseudo multi-threading and register the context by "R_OSPL_THREAD_SetDelegate" function. OSPL recommends that the driver provides both synchronous API functions having wait and asynchronous API functions not having wait. Refer to 4.6.3. (13) R_OSPL_THREAD_GetIsWaiting
Setting of notifications	Refer to 4.5.6. r_ospl_async_t
Thread local storage (TLS)	Variable having different memory area by each thread accessing same global variable. It is sometimes implemented by the array indexed by thread number. This is sometimes not called TLS. See 4.5.13. r_ospl_table_t.
T-Lock	Exclusive control of resources or variables between operations on the thread and operations on the other thread by using the mutex and so on. In OS less, it is not necessary to T-Lock. Even if OS-using environment, the target was locked by C-lock, the target does not have to be locked by T-lock. Refer to I-Lock, C-lock.

5. Sample Codes

The sample codes can be downloaded from the Renesas Electronics website.

6. Documents for Reference

User's Manual: Hardware

RZ/A1H Group User's Manual: Hardware

The latest version can be downloaded from the Renesas Electronics website.

R7S72100 RTK772100BC00000BR (GENMAI) User's Manual

The latest version can be downloaded from the Renesas Electronics website.

R7S72100 CPU (GENMAI) Optional Board RTK7721000B00000BR User's Manual

The latest version can be downloaded from the Renesas Electronics website.

ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C

The latest version can be downloaded from the ARM website.

ARM Generic Interrupt Controller Architecture Specification Architecture version 1.0

The latest version can be downloaded from the ARM website.

Technical Update/Technical News

The latest information can be downloaded from the Renesas Electronics website.

User's Manual: Development Tools

ARM Software Development Tools (ARM Compiler toolchain, ARM DS-5 etc.) can be downloaded from the ARM website.

The latest version can be downloaded from the ARM website.

Website and Support

Renesas Electronics website
http://www.renesas.com

Inquiries
http://www.renesas.com/contact/

Revision History

Doc. Rev.	Version	Date	Description
1.60	1.60	Oct 26, 2017	Added: ● use_list.h, mcu_board_select.h : selecting environment ● R_OSPL_FOR_* : version of environment ● Supported FreeRTOS ● R_OSPL_INTERRUPT_HANDLER_IS ● R_OSPL_END_OF_INTERRUPT ● R_OSPL_THREAD_Destroy ● R_OSPL_THREAD_SetDelegate ● R_OSPL_THREAD_GetDelegate ● R_OSPL_NOTIFY_ERROR ● R_OSPL_FreeCurrentThreadError ● R_OSPL_INFINITE_PSEUDO ● R_OSPL_ALL_EVENT_ALLOCATE ● r_ospl_event_group_id_t ● R_OSPL_EVENT_GROUP_Create ● R_OSPL_EVENT_GROUP_Delete ● R_OSPL_QUEUE_NULL ● R_OSPL_GetQueueAsSingletonLock ● R_OSPL_TABLE_GetStatus ● R_OSPL_MOVE_END_OF_STACK ● R_OSPL_GET_MIN_STACK_POINTER ● CHK : tiny error check ● 4.8.6. Article of inline function ● 4.8.9. Article of cached area and uncached area Removed: ● Event object Fixed: ● Warning of each compiler ● Modify word of symbol (from sentinel to canary) (from modify thread to change thread) ● Fixed not intended I-lock (the issue is only when R_OSPL_NDEBUG is not defined)
1.01	0.96	Feb. 29, 2016	Added: ● R_OSPL_EVENT_Allocate ● R_OSPL_EVENT_Free ● R_OSPL_UNUSED_FLAG ● R_OSPL_DETECT_BAD_EVENT

			• R_OSPL_ASYNC_SetDefaultPreset • Event object • Array index table • Stack check • R_ADDRESS_Add • R_OSPL_MEMORY_GetMaxLevelOfFlush • R_OSPL_CountLeadingZeros Added "R_OSPL_DEBUG_THREAD_COUNT" and removed "R_OSPL_SET_DEBUG_WORK". Added "R_OSPL_THREAD_NULL" and removed "R_OSPL_THREAD_INTERRUPT". Modified to "r_ospl_queue_id_t" from "r_ospl_queue_t".
	0.90	Jul. 31, 2015	Added r_ospl_queue_t, r_ospl_axi_cache_attribute_t
1.00	0.89	Jul. 30, 2014	Added R_OSPL_SetInterruptPriority
0.88	0.88	Jun. 20, 2014	Added OSPL initializing API. Separated from NDEBUG to R_OSPL_NDEBUG. Defined the detail of timer functions. Added the behavior of waiting for OS less (e.g. R_OSPL_THREAD_SetOnWait). Added E_NOT_THREAD error. Added R_OSPL_FLUSH_INVALIDATE. Added C-lock functions.
	0.87	Mar. 18, 2014	The meaning of bits of clear argument was reversed. Output value was changed, when "R_OSPL_EVENT_Wait" become time out. Member variables in "r_ospl_interrupt_t" structure were added. Specifications were added: R_OSPL_TLS_ERROR_CODE, R_OSPL_DEBUG_TOOL, R_OSPL_THREAD_INTERRUPT, R_OSPL_FINAL_A_FLAG Explanations were added: R_DRIVER_SetDefaultAsync, NDEBUG
	0.80	Feb. 25, 2014	First edition issued

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

- The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

- The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
  In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

- The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

- When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

- The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other disputes involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawing, chart, program, algorithm, application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics products.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (space and undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of failure or malfunction of Renesas Electronics products, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.3.0-1 November 2016)

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 6.0