Trampoline (OSEK/VDX OS) Test Procedure - Version 1.0

Florent PAVIN

October 13, 2015

Contents

1 Introduction

This document describes the test procedure for the conformance test of an OSEK/VDX operating system. The test procedure contains the definition of test sequences.
Chapter 2 contains the definition of the used test sequences. The test sequences determine how the test cases will be tested. This contains the sequence of actions that must be taken by the test program, and their expected reactions. The definition of the test sequences is based on the test cases defined in the Trampoline Test Plan [5].
Some specifications generates errors. If it's an API service call, it should return a status indicating that an error occurs and call the errorhook if defined. In some cases (ex:alarm set an event to a basic task), it could happen that the error comes from the oil file, GOIL has to tell the user that an error occurs during the compilation and don't generate the executable file. Because the executable file is not generated, these tests are different than API service calls and thus, are described in Chapter 3.
This test procedure is defined from OSEK/VDX OS v2.2.3 [1] unlike OSEK/VDX Test Procedure which is defined from OSEK/VDX OS v2.0 [2].

2 Test sequences

This chapter contains the specification of the test sequences that will be run during the conformance tests. The test sequences define the sequence of actions that will be done during the execution of the test program, i. e. the sequence of instructions executed by each task. Each test sequence fulfils the test for one ore more of the test cases defined in Trampoline Test Plan.

2.1 Task management

Test Sequence 1 :

```
TEST CASES:          1, 10, 15, 20, 21, 23, 24, 25, 28, 33, 34, 35, 37
RETURN STATUS:       EXTENDED
SCHEDULING POLICY:   NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t2 {
  AUTOSTART =  FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(INVALID_TASK)	E_OS_ID	1
t1	GetTaskState(INVALID_TASK, &TaskState)	E_OS_ID	37
t1	ChainTask(INVALID_TASK)	E_OS_ID	23
t1	ActivateTask(t2)	E_OK	
t1	*force scheduling*		
t2	ActivateTask(t1)	E_OS_LIMIT	10
t2	ActivateTask(t2)	E_OS_LIMIT	15
t2	ChainTask(t1)	E_OS_LIMIT	28
t2	TerminateTask()		

Running task	Called OS service	Return Status	Test case
t1	GetResource(RES_SCHEDULER)	E_OK	
t1	TerminateTask()	E_OS_RESOURCE	21
t1	ChainTask(t2)	E_OS_RESOURCE	25
t1	Schedule()	E_OS_RESOURCE	33
t1	ReleaseResource(RES_SCHEDULER)	E_OK	
t1	*trigger interrupt isr1*		
isr1	TerminateTask()	E_OS_CALLEVEL	20
isr1	ChainTask(t2)	E_OS_CALLEVEL	24
isr1	Schedule()	E_OS_CALLEVEL	34
isr1	GetTaskID()	E_OK, TaskID=INVALID_TASK	35
t1	TerminateTask()		

Test Sequence 2 :

```
TEST CASES:      2, 32
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:    NON-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON;
};

TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OK	2
t1	ActivateTask(t3)	E_OK	2
t1	Schedule()	E_OK	32
t3	TerminateTask()		

Running task	Called OS service	Return Status	Test case
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 3 :

```
TEST CASES:      3, 4
RETURN STATUS:      STANDARD, EXTENDED
SCHEDULING POLICY:     FULL–PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t3)	E_OK	3
t3	ActivateTask(t2)	E_OK	4
t3	TerminateTask()		
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 4 :

```
TEST CASES:      6
RETURN STATUS:      STANDARD, EXTENDED
SCHEDULING POLICY:     NON–PREEMPTIVE

TASK t1 {
  AUTOSTART =  TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
  EVENT = Event1;
};
TASK t2 {
```

```
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON;
  EVENT = Event2;
};
EVENT Event1 { MASK=AUTO; };
EVENT Event2 { MASK=AUTO; };
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OK	6
t1	GetEvent(t1, &EventMask)	E_OK, EventMask=0x0	
t1	GetEvent(t2, &EventMask)	E_OK, EventMask=0x0	
t1	Schedule()	E_OK	
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 5 :

```
TEST CASES:     7, 8
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:     FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event2;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event3;
};
EVENT Event2 { MASK=AUTO; };
EVENT Event3 { MASK=AUTO; };
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t3)	E_OK	7
t3	GetEvent(t3, &EventMask)	E_OK, EventMask=0x0	
t3	ActivateTask(t2)	E_OK	8
t3	TerminateTask()		
t2	GetEvent(t2, &EventMask)	E_OK, EventMask=0x0	
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 6 :

```
TEST CASES:      11, 16, 19, 29, 31
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART =  TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t2 {
  AUTOSTART =  FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event2;
};
EVENT Event2 { MASK = AUTO; };
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OK	
t1	*force scheduling*		
t2	ActivateTask(t1)	E_OS_LIMIT	11
t2	ActivateTask(t2)	E_OS_LIMIT	16
t2	WaitEvent(Event2)	E_OK	
t1	GetTaskState(t2,&TaskState)	E_OK, TaskState=WAITING	
t1	ActivateTask(t2)	E_OS_LIMIT	19
t1	ChainTask(t2)	E_OS_LIMIT	31
t1	SetEvent(t2, Event2)	E_OK	
t1	*force scheduling*		
t2	ChainTask(t1)	E_OS_LIMIT	29
t2	TerminateTask()		
t1	TerminateTask()		

Running task	Called OS service	Return Status	Test case

Test Sequence 7 :

```
TEST CASES:      12, 17, 30
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:     NON–PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 2;
  SCHEDULE = NON;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 2;
  SCHEDULE = NON;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 2;
  SCHEDULE = NON;
};
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OK	
t1	ActivateTask(t2)	E_OK	12
t1	Schedule()	E_OK	
t2	TerminateTask()		
t2	TerminateTask()		
t1	ActivateTask(t1)	E_OK	17
t1	ActivateTask(t3)	E_OK	
t1	ChainTask(t3)		30
t3	TerminateTask()		
t3	TerminateTask()		
t1	ActivateTask(t1)	E_OK	
t1	TerminateTask()		
t1	TerminateTask()		

Test Sequence 8 :

```
TEST CASES:      5, 13, 14, 18
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:     FULL–PREEMPTIVE
```

```
TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 2;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OK	5
t2	ActivateTask(t1)	E_OK	13
t2	ActivateTask(t3)	E_OK	14
t2	TerminateTask()		
t3	TerminateTask()		
t1	TerminateTask()		
t1	ActivateTask(t1)	E_OK	18
t1	TerminateTask()		
t1	TerminateTask()		

Test Sequence 9 :

```
TEST CASES:      22, 26, 27, 36, 38
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t3 {
```

```
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
```

Running task	Called OS service	Return Status	Test case
t1	GetTaskID()	E_OK, TaskID=t1	36
t1	GetTaskState(t1,&TaskState)	E_OK, TaskState=RUNNING	38
t1	GetTaskState(t2,&TaskState)	E_OK, TaskState=SUSPENDED	38
t1	ActivateTask(t2)	E_OK	
t1	*force scheduling*		
t2	GetTaskState(t1,&TaskState)	E_OK, TaskState=READY	38
t2	TerminateTask()		
t1	ChainTask(t3)		26
t3	ChainTask(t3)		27
t3	TerminateTask()		22

Test Sequence 10 :

```
TEST CASES:      9
RETURN STATUS:      STANDARD, EXTENDED
SCHEDULING POLICY:      FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event2;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event3;
};
EVENT Event2 { MASK = AUTO; };
EVENT Event3 { MASK = AUTO; };
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OK	
t2	GetEvent(t2, &EventMask)	E_OK, EventMask=0x0	
t2	ActivateTask(t3)	E_OK	9
t2	TerminateTask()		
t3	GetEvent(t3, &EventMask)	E_OK, EventMask=0x0	
t3	TerminateTask()		
t1	TerminateTask()		

Test Sequence 11 :

```
TEST CASES:     A task beeing released from the waiting state is treated like
     the newest task in the ready queue of its priority
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:     NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t2 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event2;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t4 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
EVENT Event2 { MASK = AUTO; };
```

Running task	Called OS service	Return Status	Test case
t2	ActivateTask(t1)	E_OK	
t2	WaitEvent(Event2)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t3)	E_OK	
t1	*force scheduling*		
t3	ActivateTask(t4)	E_OK	
t3	*force scheduling*		
t4	SetEvent(t2, Event2)	E_OK	
t4	TerminateTask()		
t3	TerminateTask()		
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 12 :

```
TEST CASES:     A preempted task is considered to be the first task in the
    ready queue of its current priority
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:     NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 3;
  SCHEDULE = NON,FULL;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 3;
  SCHEDULE = NON,FULL;
};
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OK	
t1	ActivateTask(t3)	E_OK	
t1	ActivateTask(t2)	E_OK	
t1	ActivateTask(t2)	E_OK	
t1	ActivateTask(t3)	E_OK	
t1	ActivateTask(t3)	E_OK	
t1	TerminateTask()		

Running task	Called OS service	Return Status	Test case
t2	TerminateTask()		
t3	TerminateTask()		
t2	TerminateTask()		
t2	TerminateTask()		
t3	TerminateTask()		
t3	TerminateTask()		

Test Sequence 13 :

```
TEST CASES:     Number of tasks which are not in the suspended state >= 8
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:     NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 8;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 7;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 6;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t4 {
  AUTOSTART = FALSE;
  PRIORITY = 5;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t5 {
  AUTOSTART = FALSE;
  PRIORITY = 4;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t6 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
```

```
TASK t7 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t8 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OK	
t1	ActivateTask(t3)	E_OK	
t1	ActivateTask(t4)	E_OK	
t1	ActivateTask(t5)	E_OK	
t1	ActivateTask(t6)	E_OK	
t1	ActivateTask(t7)	E_OK	
t1	ActivateTask(t8)	E_OK	
t1	TerminateTask()		
t2	TerminateTask()		
t3	TerminateTask()		
t4	TerminateTask()		
t5	TerminateTask()		
t6	TerminateTask()		
t7	TerminateTask()		
t8	TerminateTask()		

Test Sequence 14 :

```
TEST CASES:     Number of tasks which are not in the suspended state >= 16,
    number of events per task >= 8.
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:     NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 16;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task1_Event1;
  EVENT = Task1_Event2;
  EVENT = Task1_Event3;
  EVENT = Task1_Event4;
  EVENT = Task1_Event5;
  EVENT = Task1_Event6;
  EVENT = Task1_Event7;
```

```
  EVENT = Task1_Event8;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 15;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task2_Event1;
  EVENT = Task2_Event2;
  EVENT = Task2_Event3;
  EVENT = Task2_Event4;
  EVENT = Task2_Event5;
  EVENT = Task2_Event6;
  EVENT = Task2_Event7;
  EVENT = Task2_Event8;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 14;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task3_Event1;
  EVENT = Task3_Event2;
  EVENT = Task3_Event3;
  EVENT = Task3_Event4;
  EVENT = Task3_Event5;
  EVENT = Task3_Event6;
  EVENT = Task3_Event7;
  EVENT = Task3_Event8;
};
TASK t4 {
  AUTOSTART = FALSE;
  PRIORITY = 13;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task4_Event1;
  EVENT = Task4_Event2;
  EVENT = Task4_Event3;
  EVENT = Task4_Event4;
  EVENT = Task4_Event5;
  EVENT = Task4_Event6;
  EVENT = Task4_Event7;
  EVENT = Task4_Event8;
};
TASK t5 {
  AUTOSTART = FALSE;
  PRIORITY = 12;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task5_Event1;
```

```
  EVENT = Task5_Event2;
  EVENT = Task5_Event3;
  EVENT = Task5_Event4;
  EVENT = Task5_Event5;
  EVENT = Task5_Event6;
  EVENT = Task5_Event7;
  EVENT = Task5_Event8;
};
TASK t6 {
  AUTOSTART = FALSE;
  PRIORITY = 11;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task6_Event1;
  EVENT = Task6_Event2;
  EVENT = Task6_Event3;
  EVENT = Task6_Event4;
  EVENT = Task6_Event5;
  EVENT = Task6_Event6;
  EVENT = Task6_Event7;
  EVENT = Task6_Event8;
};
TASK t7 {
  AUTOSTART = FALSE;
  PRIORITY = 10;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task7_Event1;
  EVENT = Task7_Event2;
  EVENT = Task7_Event3;
  EVENT = Task7_Event4;
  EVENT = Task7_Event5;
  EVENT = Task7_Event6;
  EVENT = Task7_Event7;
  EVENT = Task7_Event8;
};
TASK t8 {
  AUTOSTART = FALSE;
  PRIORITY = 9;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task8_Event1;
  EVENT = Task8_Event2;
  EVENT = Task8_Event3;
  EVENT = Task8_Event4;
  EVENT = Task8_Event5;
  EVENT = Task8_Event6;
  EVENT = Task8_Event7;
  EVENT = Task8_Event8;
};
```

```
TASK t9 {
  AUTOSTART = FALSE;
  PRIORITY = 8;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task9_Event1;
  EVENT = Task9_Event2;
  EVENT = Task9_Event3;
  EVENT = Task9_Event4;
  EVENT = Task9_Event5;
  EVENT = Task9_Event6;
  EVENT = Task9_Event7;
  EVENT = Task9_Event8;
};
TASK t10 {
  AUTOSTART = FALSE;
  PRIORITY = 7;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task10_Event1;
  EVENT = Task10_Event2;
  EVENT = Task10_Event3;
  EVENT = Task10_Event4;
  EVENT = Task10_Event5;
  EVENT = Task10_Event6;
  EVENT = Task10_Event7;
  EVENT = Task10_Event8;
};
TASK t11{
  AUTOSTART = FALSE;
  PRIORITY = 6;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task11_Event1;
  EVENT = Task11_Event2;
  EVENT = Task11_Event3;
  EVENT = Task11_Event4;
  EVENT = Task11_Event5;
  EVENT = Task11_Event6;
  EVENT = Task11_Event7;
  EVENT = Task11_Event8;
};
TASK t12 {
  AUTOSTART = FALSE;
  PRIORITY = 5;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task12_Event1;
  EVENT = Task12_Event2;
  EVENT = Task12_Event3;
```

```
  EVENT = Task12_Event4;
  EVENT = Task12_Event5;
  EVENT = Task12_Event6;
  EVENT = Task12_Event7;
  EVENT = Task12_Event8;
};
TASK t13 {
  AUTOSTART = FALSE;
  PRIORITY = 4;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task13_Event1;
  EVENT = Task13_Event2;
  EVENT = Task13_Event3;
  EVENT = Task13_Event4;
  EVENT = Task13_Event5;
  EVENT = Task13_Event6;
  EVENT = Task13_Event7;
  EVENT = Task13_Event8;
};
TASK t14 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task14_Event1;
  EVENT = Task14_Event2;
  EVENT = Task14_Event3;
  EVENT = Task14_Event4;
  EVENT = Task14_Event5;
  EVENT = Task14_Event6;
  EVENT = Task14_Event7;
  EVENT = Task14_Event8;
};
TASK t15 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Task15_Event1;
  EVENT = Task15_Event2;
  EVENT = Task15_Event3;
  EVENT = Task15_Event4;
  EVENT = Task15_Event5;
  EVENT = Task15_Event6;
  EVENT = Task15_Event7;
  EVENT = Task15_Event8;
};
TASK t16 {
  AUTOSTART = FALSE;
```

```
    PRIORITY = 1;
    ACTIVATION = 1;
    SCHEDULE = NON,FULL;
    EVENT = Task16_Event1;
    EVENT = Task16_Event2;
    EVENT = Task16_Event3;
    EVENT = Task16_Event4;
    EVENT = Task16_Event5;
    EVENT = Task16_Event6;
    EVENT = Task16_Event7;
    EVENT = Task16_Event8;
};
EVENT Task1_Event1 { MASK = AUTO; };
EVENT Task1_Event2 { MASK = AUTO; };
EVENT Task1_Event3 { MASK = AUTO; };
EVENT Task1_Event4 { MASK = AUTO; };
EVENT Task1_Event5 { MASK = AUTO; };
EVENT Task1_Event6 { MASK = AUTO; };
EVENT Task1_Event7 { MASK = AUTO; };
EVENT Task1_Event8 { MASK = AUTO; };
EVENT Task2_Event1 { MASK = AUTO; };
EVENT Task2_Event2 { MASK = AUTO; };
EVENT Task2_Event3 { MASK = AUTO; };
EVENT Task2_Event4 { MASK = AUTO; };
EVENT Task2_Event5 { MASK = AUTO; };
EVENT Task2_Event6 { MASK = AUTO; };
EVENT Task2_Event7 { MASK = AUTO; };
EVENT Task2_Event8 { MASK = AUTO; };
EVENT Task3_Event1 { MASK = AUTO; };
EVENT Task3_Event2 { MASK = AUTO; };
EVENT Task3_Event3 { MASK = AUTO; };
EVENT Task3_Event4 { MASK = AUTO; };
EVENT Task3_Event5 { MASK = AUTO; };
EVENT Task3_Event6 { MASK = AUTO; };
EVENT Task3_Event7 { MASK = AUTO; };
EVENT Task3_Event8 { MASK = AUTO; };
EVENT Task4_Event1 { MASK = AUTO; };
EVENT Task4_Event2 { MASK = AUTO; };
EVENT Task4_Event3 { MASK = AUTO; };
EVENT Task4_Event4 { MASK = AUTO; };
EVENT Task4_Event5 { MASK = AUTO; };
EVENT Task4_Event6 { MASK = AUTO; };
EVENT Task4_Event7 { MASK = AUTO; };
EVENT Task4_Event8 { MASK = AUTO; };
EVENT Task5_Event1 { MASK = AUTO; };
EVENT Task5_Event2 { MASK = AUTO; };
EVENT Task5_Event3 { MASK = AUTO; };
EVENT Task5_Event4 { MASK = AUTO; };
EVENT Task5_Event5 { MASK = AUTO; };
EVENT Task5_Event6 { MASK = AUTO; };
```

```
EVENT Task5_Event7 { MASK = AUTO; };
EVENT Task5_Event8 { MASK = AUTO; };
EVENT Task6_Event1 { MASK = AUTO; };
EVENT Task6_Event2 { MASK = AUTO; };
EVENT Task6_Event3 { MASK = AUTO; };
EVENT Task6_Event4 { MASK = AUTO; };
EVENT Task6_Event5 { MASK = AUTO; };
EVENT Task6_Event6 { MASK = AUTO; };
EVENT Task6_Event7 { MASK = AUTO; };
EVENT Task6_Event8 { MASK = AUTO; };
EVENT Task7_Event1 { MASK = AUTO; };
EVENT Task7_Event2 { MASK = AUTO; };
EVENT Task7_Event3 { MASK = AUTO; };
EVENT Task7_Event4 { MASK = AUTO; };
EVENT Task7_Event5 { MASK = AUTO; };
EVENT Task7_Event6 { MASK = AUTO; };
EVENT Task7_Event7 { MASK = AUTO; };
EVENT Task7_Event8 { MASK = AUTO; };
EVENT Task8_Event1 { MASK = AUTO; };
EVENT Task8_Event2 { MASK = AUTO; };
EVENT Task8_Event3 { MASK = AUTO; };
EVENT Task8_Event4 { MASK = AUTO; };
EVENT Task8_Event5 { MASK = AUTO; };
EVENT Task8_Event6 { MASK = AUTO; };
EVENT Task8_Event7 { MASK = AUTO; };
EVENT Task8_Event8 { MASK = AUTO; };
EVENT Task9_Event1 { MASK = AUTO; };
EVENT Task9_Event2 { MASK = AUTO; };
EVENT Task9_Event3 { MASK = AUTO; };
EVENT Task9_Event4 { MASK = AUTO; };
EVENT Task9_Event5 { MASK = AUTO; };
EVENT Task9_Event6 { MASK = AUTO; };
EVENT Task9_Event7 { MASK = AUTO; };
EVENT Task9_Event8 { MASK = AUTO; };
EVENT Task10_Event1 { MASK = AUTO; };
EVENT Task10_Event2 { MASK = AUTO; };
EVENT Task10_Event3 { MASK = AUTO; };
EVENT Task10_Event4 { MASK = AUTO; };
EVENT Task10_Event5 { MASK = AUTO; };
EVENT Task10_Event6 { MASK = AUTO; };
EVENT Task10_Event7 { MASK = AUTO; };
EVENT Task10_Event8 { MASK = AUTO; };
EVENT Task11_Event1 { MASK = AUTO; };
EVENT Task11_Event2 { MASK = AUTO; };
EVENT Task11_Event3 { MASK = AUTO; };
EVENT Task11_Event4 { MASK = AUTO; };
EVENT Task11_Event5 { MASK = AUTO; };
EVENT Task11_Event6 { MASK = AUTO; };
EVENT Task11_Event7 { MASK = AUTO; };
EVENT Task11_Event8 { MASK = AUTO; };
```

```
EVENT Task12_Event1 { MASK = AUTO; };
EVENT Task12_Event2 { MASK = AUTO; };
EVENT Task12_Event3 { MASK = AUTO; };
EVENT Task12_Event4 { MASK = AUTO; };
EVENT Task12_Event5 { MASK = AUTO; };
EVENT Task12_Event6 { MASK = AUTO; };
EVENT Task12_Event7 { MASK = AUTO; };
EVENT Task12_Event8 { MASK = AUTO; };
EVENT Task13_Event1 { MASK = AUTO; };
EVENT Task13_Event2 { MASK = AUTO; };
EVENT Task13_Event3 { MASK = AUTO; };
EVENT Task13_Event4 { MASK = AUTO; };
EVENT Task13_Event5 { MASK = AUTO; };
EVENT Task13_Event6 { MASK = AUTO; };
EVENT Task13_Event7 { MASK = AUTO; };
EVENT Task13_Event8 { MASK = AUTO; };
EVENT Task14_Event1 { MASK = AUTO; };
EVENT Task14_Event2 { MASK = AUTO; };
EVENT Task14_Event3 { MASK = AUTO; };
EVENT Task14_Event4 { MASK = AUTO; };
EVENT Task14_Event5 { MASK = AUTO; };
EVENT Task14_Event6 { MASK = AUTO; };
EVENT Task14_Event7 { MASK = AUTO; };
EVENT Task14_Event8 { MASK = AUTO; };
EVENT Task15_Event1 { MASK = AUTO; };
EVENT Task15_Event2 { MASK = AUTO; };
EVENT Task15_Event3 { MASK = AUTO; };
EVENT Task15_Event4 { MASK = AUTO; };
EVENT Task15_Event5 { MASK = AUTO; };
EVENT Task15_Event6 { MASK = AUTO; };
EVENT Task15_Event7 { MASK = AUTO; };
EVENT Task15_Event8 { MASK = AUTO; };
EVENT Task16_Event1 { MASK = AUTO; };
EVENT Task16_Event2 { MASK = AUTO; };
EVENT Task16_Event3 { MASK = AUTO; };
EVENT Task16_Event4 { MASK = AUTO; };
EVENT Task16_Event5 { MASK = AUTO; };
EVENT Task16_Event6 { MASK = AUTO; };
EVENT Task16_Event7 { MASK = AUTO; };
EVENT Task16_Event8 { MASK = AUTO; };
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OK	
t1	ActivateTask(t3)	E_OK	
t1	ActivateTask(t4)	E_OK	
t1	ActivateTask(t5)	E_OK	
t1	ActivateTask(t6)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t7)	E_OK	
t1	ActivateTask(t8)	E_OK	
t1	ActivateTask(Task9)	E_OK	
t1	ActivateTask(t10)	E_OK	
t1	ActivateTask(t11)	E_OK	
t1	ActivateTask(t12)	E_OK	
t1	ActivateTask(t13)	E_OK	
t1	ActivateTask(t14)	E_OK	
t1	ActivateTask(t15)	E_OK	
t1	ActivateTask(t16)	E_OK	
t1	ClearEvent(t1_Event1)	E_OK	
t1	ClearEvent(t1_Event2)	E_OK	
t1	ClearEvent(t1_Event3)	E_OK	
t1	ClearEvent(t1_Event4)	E_OK	
t1	ClearEvent(t1_Event5)	E_OK	
t1	ClearEvent(t1_Event6)	E_OK	
t1	ClearEvent(t1_Event7)	E_OK	
t1	ClearEvent(t1_Event8)	E_OK	
t1	TerminateTask()		
t2	ClearEvent(t2_Event1)	E_OK	
t2	ClearEvent(t2_Event2)	E_OK	
t2	ClearEvent(t2_Event3)	E_OK	
t2	ClearEvent(t2_Event4)	E_OK	
t2	ClearEvent(t2_Event5)	E_OK	
t2	ClearEvent(t2_Event6)	E_OK	
t2	ClearEvent(t2_Event7)	E_OK	
t2	ClearEvent(t2_Event8)	E_OK	
t2	TerminateTask()		
t3	ClearEvent(t3_Event1)	E_OK	
t3	ClearEvent(t3_Event2)	E_OK	
t3	ClearEvent(t3_Event3)	E_OK	
t3	ClearEvent(t3_Event4)	E_OK	
t3	ClearEvent(t3_Event5)	E_OK	
t3	ClearEvent(t3_Event6)	E_OK	
t3	ClearEvent(t3_Event7)	E_OK	
t3	ClearEvent(t3_Event8)	E_OK	
t3	TerminateTask()		
t4	ClearEvent(t4_Event1)	E_OK	
t4	ClearEvent(t4_Event2)	E_OK	
t4	ClearEvent(t4_Event3)	E_OK	

Running task	Called OS service	Return Status	Test case
t4	ClearEvent(t4_Event4)	E_OK	
t4	ClearEvent(t4_Event5)	E_OK	
t4	ClearEvent(t4_Event6)	E_OK	
t4	ClearEvent(t4_Event7)	E_OK	
t4	ClearEvent(t4_Event8)	E_OK	
t4	TerminateTask()		
t5	ClearEvent(t5_Event1)	E_OK	
t5	ClearEvent(t5_Event2)	E_OK	
t5	ClearEvent(t5_Event3)	E_OK	
t5	ClearEvent(t5_Event4)	E_OK	
t5	ClearEvent(t5_Event5)	E_OK	
t5	ClearEvent(t5_Event6)	E_OK	
t5	ClearEvent(t5_Event7)	E_OK	
t5	ClearEvent(t5_Event8)	E_OK	
t5	TerminateTask()		
t6	ClearEvent(t6_Event1)	E_OK	
t6	ClearEvent(t6_Event2)	E_OK	
t6	ClearEvent(t6_Event3)	E_OK	
t6	ClearEvent(t6_Event4)	E_OK	
t6	ClearEvent(t6_Event5)	E_OK	
t6	ClearEvent(t6_Event6)	E_OK	
t6	ClearEvent(t6_Event7)	E_OK	
t6	ClearEvent(t6_Event8)	E_OK	
t6	TerminateTask()		
t7	ClearEvent(t7_Event1)	E_OK	
t7	ClearEvent(t7_Event2)	E_OK	
t7	ClearEvent(t7_Event3)	E_OK	
t7	ClearEvent(t7_Event4)	E_OK	
t7	ClearEvent(t7_Event5)	E_OK	
t7	ClearEvent(t7_Event6)	E_OK	
t7	ClearEvent(t7_Event7)	E_OK	
t7	ClearEvent(t7_Event8)	E_OK	
t7	TerminateTask()		
t8	ClearEvent(t8_Event1)	E_OK	
t8	ClearEvent(t8_Event2)	E_OK	
t8	ClearEvent(t8_Event3)	E_OK	
t8	ClearEvent(t8_Event4)	E_OK	
t8	ClearEvent(t8_Event5)	E_OK	
t8	ClearEvent(t8_Event6)	E_OK	
t8	ClearEvent(t8_Event7)	E_OK	

Running task	Called OS service	Return Status	Test case
t8	ClearEvent(t8_Event8)	E_OK	
t8	TerminateTask()		
t9	ClearEvent(t9_Event1)	E_OK	
t9	ClearEvent(t9_Event2)	E_OK	
t9	ClearEvent(t9_Event3)	E_OK	
t9	ClearEvent(t9_Event4)	E_OK	
t9	ClearEvent(t9_Event5)	E_OK	
t9	ClearEvent(t9_Event6)	E_OK	
t9	ClearEvent(t9_Event7)	E_OK	
t9	ClearEvent(t9_Event8)	E_OK	
t9	TerminateTask()		
t10	ClearEvent(t10_Event1)	E_OK	
t10	ClearEvent(t10_Event2)	E_OK	
t10	ClearEvent(t10_Event3)	E_OK	
t10	ClearEvent(t10_Event4)	E_OK	
t10	ClearEvent(t10_Event5)	E_OK	
t10	ClearEvent(t10_Event6)	E_OK	
t10	ClearEvent(t10_Event7)	E_OK	
t10	ClearEvent(t10_Event8)	E_OK	
t10	TerminateTask()		
t11	ClearEvent(t11_Event1)	E_OK	
t11	ClearEvent(t11_Event2)	E_OK	
t11	ClearEvent(t11_Event3)	E_OK	
t11	ClearEvent(t11_Event4)	E_OK	
t11	ClearEvent(t11_Event5)	E_OK	
t11	ClearEvent(t11_Event6)	E_OK	
t11	ClearEvent(t11_Event7)	E_OK	
t11	ClearEvent(t11_Event8)	E_OK	
t11	TerminateTask()		
t12	ClearEvent(t12_Event1)	E_OK	
t12	ClearEvent(t12_Event2)	E_OK	
t12	ClearEvent(t12_Event3)	E_OK	
t12	ClearEvent(t12_Event4)	E_OK	
t12	ClearEvent(t12_Event5)	E_OK	
t12	ClearEvent(t12_Event6)	E_OK	
t12	ClearEvent(t12_Event7)	E_OK	
t12	ClearEvent(t12_Event8)	E_OK	
t12	TerminateTask()		
t13	ClearEvent(t13_Event1)	E_OK	
t13	ClearEvent(t13_Event2)	E_OK	

Running task	Called OS service	Return Status	Test case
t13	ClearEvent(t13_Event3)	E_OK	
t13	ClearEvent(t13_Event4)	E_OK	
t13	ClearEvent(t13_Event5)	E_OK	
t13	ClearEvent(t13_Event6)	E_OK	
t13	ClearEvent(t13_Event7)	E_OK	
t13	ClearEvent(t13_Event8)	E_OK	
t13	TerminateTask()		
t14	ClearEvent(t14_Event1)	E_OK	
t14	ClearEvent(t14_Event2)	E_OK	
t14	ClearEvent(t14_Event3)	E_OK	
t14	ClearEvent(t14_Event4)	E_OK	
t14	ClearEvent(t14_Event5)	E_OK	
t14	ClearEvent(t14_Event6)	E_OK	
t14	ClearEvent(t14_Event7)	E_OK	
t14	ClearEvent(t14_Event8)	E_OK	
t14	TerminateTask()		
t15	ClearEvent(t15_Event1)	E_OK	
t15	ClearEvent(t15_Event2)	E_OK	
t15	ClearEvent(t15_Event3)	E_OK	
t15	ClearEvent(t15_Event4)	E_OK	
t15	ClearEvent(t15_Event5)	E_OK	
t15	ClearEvent(t15_Event6)	E_OK	
t15	ClearEvent(t15_Event7)	E_OK	
t15	ClearEvent(t15_Event8)	E_OK	
t15	TerminateTask()		
t16	ClearEvent(t16_Event1)	E_OK	
t16	ClearEvent(t16_Event2)	E_OK	
t16	ClearEvent(t16_Event3)	E_OK	
t16	ClearEvent(t16_Event4)	E_OK	
t16	ClearEvent(t16_Event5)	E_OK	
t16	ClearEvent(t16_Event6)	E_OK	
t16	ClearEvent(t16_Event7)	E_OK	
t16	ClearEvent(t16_Event8)	E_OK	
t16	TerminateTask()		

Test Sequence 15 :

```
TEST CASES:      35, 39 to 56
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
```

```
  SCHEDULE = NON,FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 2;
  SCHEDULE = NON,FULL;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 2;
  SCHEDULE = NON,FULL;
};
TASK t4 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 2;
  SCHEDULE = NON,FULL;
};
TASK t5 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event1;
};
TASK t6 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event1;
};

TASK t7 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event1;
};

TASK t8 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event1;
};
```

```
ISR isr1 {
  STACKSIZE = 32768;
  CATEGORY = 2;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};

EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t8	WaitEvent(Event1)	E_OK	
t1	*trigger interrupt*	E_OK	
isr1	ActivateTask(INVALID_TASK)	E_OS_ID	41
isr1	ActivateTask(t2)	E_OK	44, 42
isr1	ActivateTask(t3)	E_OK	45
isr1	ActivateTask(t4)	E_OK	43
isr1	ActivateTask(t5)	E_OK	48, 46
isr1	ActivateTask(t6)	E_OK	49
isr1	ActivateTask(t7)	E_OK	47
isr1	ActivateTask(t2)	E_OK	54, 52
isr1	ActivateTask(t3)	E_OK	55
isr1	ActivateTask(t4)	E_OK	53
isr1	ActivateTask(t1)	E_OS_LIMIT	50
isr1	ActivateTask(t5)	E_OS_LIMIT	51
isr1	ActivateTask(t8)	E_OS_LIMIT	56
isr1	GetTaskState(INVALID_TASK, &TaskState)	E_OS_ID	39
isr1	GetTaskState(t8, &TaskState)	E_OK, TaskState=WAITING	40
isr1	GetTaskID()	E_OK, TaskID=INVALID_TASK	35
{NON}t1	TerminateTask()		
t4	TerminateTask()		
t7	TerminateTask()		
t4	TerminateTask()		
t8	TerminateTask()		
{FULL}t1	TerminateTask()		
t3	TerminateTask()		
t6	TerminateTask()		
t3	TerminateTask()		
t2	TerminateTask()		
t5	TerminateTask()		

Running task	Called OS service	Return Status	Test case
t2	TerminateTask()		

2.2 Interrupt processing

Test cases 4 and 5 are respectively the same as 7 and 8 because we can't test ISRs of category 1 (they're OS independant).
Maximum number of activation for ISRs is 1, so if an interrupt is called several times when interrupts are disabled or suspended, only one will be executed when respectively enabling or resuming the interrupts.
Every counter ticks, a signal is send to Trampoline in order it checks if an alarm expires or not. The signal sent is SIGUSR2 so, during alarm test procedure, we will be careful not to use SIGUSR2 when we use interrupt at the same time of an alarm (see Test sequence 4).
Since no API service calls are allowed between SuspendAllInterrupts() and ResumeAllInterrupts(), test sequence 5 appears.

Test Sequence 1 :

```
TEST CASES:     1, 2, 4, 5, 6, 7, 9, 10, 11, 12, 14, 15, 16, 20, 21, 22, 23,
    24, 25, 26, 27, 28, 29, 30, 31, 34
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};

ISR isr2 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 2;
  SOURCE = SIGUSR2;
};

ISR isr3 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 3;
  SOURCE = SIGPIPE;
```

};

Running task	Called OS service	Return Status	Test case
t1	DisableAllInterrupts()		4
t1	EnableAllInterrupts()		2
t1	DisableAllInterrupts()		4
t1	*trigger interrupt isr2*		15
t1	*trigger interrupt isr2*		16
t1	*trigger interrupt isr2*		16
t1	EnableAllInterrupts()		1
isr2			
t1	*trigger interrupt isr2*		
isr2			
t1	SuspendedAllInterrupts()		9
t1	ResumeAllInterrupts()		6
t1	SuspendedAllInterrupts()		
t1	SuspendedAllInterrupts()		
t1	SuspendedAllInterrupts()		
t1	*trigger interrupt isr2*		
t1	ResumeAllInterrupts()		
t1	ResumeAllInterrupts()		
t1	ResumeAllInterrupts()		5, 7
isr2			
t1	*trigger interrupt isr2*		
isr2			
t1	SuspendedOSInterrupts()		14
t1	ResumeOSInterrupts()		11
t1	SuspendedOSInterrupts()		
t1	SuspendedOSInterrupts()		
t1	SuspendedOSInterrupts()		
t1	*trigger interrupt isr2*		
t1	ResumeOSInterrupts()		
t1	ResumeOSInterrupts()		
t1	ResumeOSInterrupts()		10, 12
isr2			
t1	*trigger interrupt isr1*		
isr1	DisableAllInterrupts()		22
isr1	EnableAllInterrupts()		21
isr1	DisableAllInterrupts()		22
isr1	*trigger interrupt isr2*		
isr1	*trigger interrupt isr2*		

Running task	Called OS service	Return Status	Test case
isr1	*trigger interrupt isr2*		
isr1	EnableAllInterrupts()		20
isr2			
isr1	*trigger interrupt isr2*		31
isr2			
isr1	SuspendedAllInterrupts()		26
isr1	ResumeAllInterrupts()		24
isr1	SuspendedAllInterrupts()		
isr1	SuspendedAllInterrupts()		
isr1	SuspendedAllInterrupts()		
isr1	*trigger interrupt isr2*		
isr1	ResumeAllInterrupts()		
isr1	ResumeAllInterrupts()		
isr1	ResumeAllInterrupts()		23, 25
isr2			
isr1	*trigger interrupt isr2*		
isr2			
isr1	SuspendedOSInterrupts()		30
isr1	ResumeOSInterrupts()		28
isr1	SuspendedOSInterrupts()		
isr1	SuspendedOSInterrupts()		
isr1	SuspendedOSInterrupts()		
isr1	*trigger interrupt isr2*		
isr1	ResumeOSInterrupts()		
isr1	ResumeOSInterrupts()		
isr1	ResumeOSInterrupts()		27, 29
isr2			
isr1	*trigger interrupt isr3*		
isr3	*trigger interrupt isr2*		
isr2			34
isr1			
t1	TerminateTask()		

Test Sequence 2 :

```
TEST CASES:      17, 32
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:    NON-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
```

```
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON;
};

ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};

ISR isr2 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 2;
  SOURCE = SIGUSR2;
};
```

Running task	Called OS service	Return Status	Test case
t1	*trigger interrupt isr2*		
isr2	*trigger interrupt isr1*		32
isr2	ActivateTask(t2)	E_OK	
isr1			17
t1	TerminateTask()		
t2	TerminateTask()		

Test Sequence 3 :

```
TEST CASES:      18, 33
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:    FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
```

```
ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};

ISR isr2 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGUSR2;
};
```

Running task	Called OS service	Return Status	Test case
t1	*trigger interrupt isr2*		
isr2	*trigger interrupt isr1*		33
isr2	ActivateTask(t2)	E_OK	
isr1			18
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 4 :

```
TEST CASES:     35, 36, 37, 38, 39
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:     NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 2;
  SOURCE = SIGTERM;
};

COUNTER Counter1{
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};

ALARM Alarm1{
  COUNTER = Counter1;
```

```
  ACTION = ALARMCALLBACK {
    ALARMCALLBACKNAME = "CallBack1";
  };
  AUTOSTART = FALSE;
};

ALARM Alarm2{
  COUNTER = Counter1;
  ACTION = ALARMCALLBACK {
    ALARMCALLBACKNAME = "CallBack2";
  };
  AUTOSTART = FALSE;
};
```

Running task	Called OS service	Return Status	Test case
t1	SetAbsAlarm(Alarm1, 2, 0)	E_OK	
t1	*Wait alarm expires & Force scheduling*		
CallBack1	SuspendAllInterrupts()		38
CallBack1	ResumeAllInterrupts()		35
CallBack1	*trigger interrupt isr1*		39
isr1			
t1	SetAbsAlarm(Alarm2, 2, 0)	E_OK	
t1	*Wait alarm expires & Force scheduling*		
CallBack2	SuspendAllInterrupts()		
CallBack2	SuspendAllInterrupts()		
CallBack2	SuspendAllInterrupts()		
CallBack2	*trigger interrupt isr1*		39
CallBack2	ResumeAllInterrupts()		
CallBack2	ResumeAllInterrupts()		
CallBack2	ResumeAllInterrupts()		36, 37
isr1			
t1	TerminateTask()		

Test Sequence 5 :

Since no service call are allowed between disabled/enabled interrupts, It should return an error. Since AUTOSAR OS should return E_OS_DISABLEDINT in this case (OS093), it is implemented in Trampoline "OSEK" OS. Moreover, Suspending/Resuming interrupts work by pair, it means you can't resume "OS" interrupts after disabling "All" interrupts, the service is not done (see OSEK OS p26 and AUTOSAR OS Requirements OS092).

```
TEST CASES:      3, 8, 13, 19
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:    FULL_PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
```

```
  SCHEDULE = FULL;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
ISR isr1{
  CATEGORY = 2;
  PRIORITY = 1;
  STACKSIZE = 32768;
  SOURCE = SIGTERM;
};
```

Running task	Called OS service	Return Status	Test case
t1	SuspendAllInterrupts()		
t1	ActivateTask(t2)	E_OS_DISABLEDINT	
t1	*trigger interrupt isr1*		
t1	ResumeOSInterrupts()		13
t1	EnableAllInterrupts()		3
t1	ResumeAllInterrupts()		
isr1			
t1	SuspendOSInterrupts()		
t1	ActivateTask(t2)	E_OS_DISABLEDINT	
t1	*trigger interrupt isr1*		
t1	ResumeAllInterrupts()		8
t1	EnableAllInterrupts()		3
t1	DisableAllInterrupts()		19
t1	EnableAllInterrupts()		3
t1	ResumeOSInterrupts()		
isr1			
t1	DisableAllInterrupts()		
t1	ActivateTask(t2)	E_OS_DISABLEDINT	
t1	*trigger interrupt isr1*		
t1	ResumeAllInterrupts()		8
t1	ResumeOSInterrupts()		13
t1	EnableAllInterrupts()		
isr1			
t1	TerminateTask()		

Test Sequence 6 :

```
TEST CASES:     19
RETURN STATUS:     EXTENDED
```

```
SCHEDULING POLICY:    FULL_PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = Resource1;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
RESOURCE Resource1{
  RESOURCEPROPERTY = STANDARD;
};
EVENT Event1{
  MASK = AUTO;
};
COUNTER Counter1{
  MAXALLOWEDVALUE = 10;
  MINCYCLE = 1;
  TICKSPERBASE = 1;
};
ALARM Alarm1{
  COUNTER = Counter1;
  ACTION = ACTIVATETASK {
    TASK = t2;
  };
  AUTOSTART = FALSE;
};
```

Running task	Called OS service	Return Status	Test case
t1	GetTaskState(t3, &State)	E_OK, State=READY	
t1	SuspendAllInterrupts()		
t1	ActivateTask(t2)	E_OS_DISABLEDINT	19
t1	TerminateTask()	E_OS_DISABLEDINT	19
t1	ChainTask(t2)	E_OS_DISABLEDINT	19
t1	Schedule()	E_OS_DISABLEDINT	19
t1	GetTaskID(&TaskType)	E_OS_DISABLEDINT	19
t1	GetTaskState(t2, &TaskStateType)	E_OS_DISABLEDINT	19
t1	GetResource(Resource1)	E_OS_DISABLEDINT	19
t1	ReleaseResource(Resource1)	E_OS_DISABLEDINT	19

Running task	Called OS service	Return Status	Test case
t1	SetEvent(t3, Event1)	E_OS_DISABLEDINT	19
t1	ClearEvent(Event1)	E_OS_DISABLEDINT	19
t1	GetEvent(t2, &EventMaskType)	E_OS_DISABLEDINT	19
t1	WaitEvent(Event1)	E_OS_DISABLEDINT	19
t1	GetAlarmBase(Alarm1, &AlarmBaseType)	E_OS_DISABLEDINT	19
t1	GetAlarm(Alarm1, &TickType)	E_OS_DISABLEDINT	19
t1	SetRelAlarm(Alarm1, 1, 0)	E_OS_DISABLEDINT	19
t1	SetAbsAlarm(Alarm1, 1, 0)	E_OS_DISABLEDINT	19
t1	CancelAlarm(Alarm1)	E_OS_DISABLEDINT	19
t1	GetActiveApplicationMode()	E_OS_DISABLEDINT	19
t1	ResumeAllInterrupts()		
t1	GetEvent(t3, &EventMaskType)	E_OK, Event=0	
t1	TerminateTask()		

2.3 Event mechanism

Test Sequence 1 :

```
TEST CASES:     1, 2, 3, 11, 12, 14, 15, 16, 20, 21, 22
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t2 {
  AUTOSTART =  FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  RESOURCE = Resource1;
  EVENT = Event1;
};
ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};

EVENT Event1 {
  MASK = AUTO;
};
```

```
RESOURCE Resource1 {
  RESOURCEPROPERTY = STANDARD;
};
```

Running task	Called OS service	Return Status	Test case
t1	SetEvent(INVALID_TASK, Event1)	E_OS_ID	1
t1	SetEvent(t1,Event1)	E_OS_ACCESS	2
t1	SetEvent(t2, Event1)	E_OS_STATE	3
t1	ClearEvent(Event1)	E_OS_ACCESS	11
t1	*trigger interrupt isr1*		
isr1	ClearEvent(Event1)	E_OS_CALLEVEL	12
isr1	WaitEvent(Event1)	E_OS_CALLEVEL	22
t1	GetEvent(INVALID_TASK, &EventMask)	E_OS_ID	14
t1	GetEvent(t1, &EventMask)	E_OS_ACCESS	15
t1	GetEvent(t2, &EventMask)	E_OS_STATE	16
t1	WaitEvent(Event1)	E_OS_ACCESS	20
t1	ChainTask(t2)		
t2	GetResource(Resource1)	E_OK	
t2	WaitEvent(Event1)	E_OS_RESOURCE	21
t2	ReleaseResource(Resource1)	E_OK	
t2	TerminateTask()		

Test Sequence 2 :

```
TEST CASES:      13, 17, 18, 19, 23, 24
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event1;
};
TASK t2 {
  AUTOSTART =  TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event2;
};

EVENT Event1 {
  MASK = AUTO;
};
EVENT Event2 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t2	ActivateTask(t1)	E_OK	
t2	SetEvent(t1, Event1)	E_OK	9/10
t2	GetEvent(t1, &EventMask)	E_OK, EventMask=Event1	18
t2	WaitEvent(Event2)	E_OK	23
t1	WaitEvent(Event1)	E_OK	24
t1	GetEvent(t2, &EventMask)	E_OK, EventMask=0x0	19
t1	SetEvent(t1, Event1)	E_OK	
t1	GetEvent(t1, &EventMask)	E_OK, EventMask=Event1	17
t1	WaitEvent(Event1)	E_OK	24
t1	ClearEvent(Event1)	E_OK	13
t1	GetEvent(t1, &EventMask)	E_OK, EventMask=0x0	
t1	SetEvent(t2, Event2)	E_OK	
t1	*force scheduling*		
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 3 :

```
TEST CASES:      4, 5, 9
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:    NON-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
TASK t2 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON;
  EVENT = Event1;
  EVENT = Event2;
  EVENT = Event3;
};

TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
  EVENT = Event3;
};

EVENT Event1 {
  MASK = AUTO;
```

```
};

EVENT Event2 {
  MASK = AUTO;
};

EVENT Event3 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t2	WaitEvent(Event1)	E_OK	
t1	GetTaskState(t2, &TaskState)	E_OK, TaskState=WAITING	
t1	GetEvent(t2, &EventMask)	E_OK, EventMask=0x0	
t1	SetEvent(t2, Event2)	E_OK	5
t1	GetTaskState(t2, &TaskState)	E_OK, TaskState=WAITING	
t1	GetEvent(t2, &EventMask)	E_OK, EventMask=Event2	
t1	SetEvent(t2, Event1)	E_OK	4
t1	GetTaskState(t2, &TaskState)	E_OK, TaskState=READY	
t1	GetEvent(t2, &EventMask)	E_OK, EventMask= Event1|Event2	
t1	SetEvent(t2, Event3)	E_OK	9
t1	GetEvent(t2, &EventMask)	E_OK, EventMask= Event1|Event2|Event3	
t1	ActivateTask(t3)	E_OK	
t1	SetEvent(t2, Event3)	E_OK	9[1]
t1	TerminateTask()		
t2	TerminateTask()		
t3	WaitEvent(Event3)	E_OK	
t3	TerminateTask()		

Test Sequence 4 :

```
TEST CASES:      6, 7, 8, 10
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:    FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
TASK t2 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
```

```
  EVENT = Event1;
  EVENT = Event2;
};

TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event3;
};

TASK t4 {
  AUTOSTART = FALSE;
  PRIORITY = 4;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

EVENT Event1 {
  MASK = AUTO;
};

EVENT Event2 {
  MASK = AUTO;
};

EVENT Event3 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t2	WaitEvent(Event1)	E_OK	
t1	SetEvent(t2, Event2)	E_OK	8
t1	GetTaskState(t2, &TaskState)	E_OK, TaskState=WAITING	
t1	GetEvent(t2, &EventMask)	E_OK, EventMask= Event2	
t1	ActivateTask(t3)	E_OK	
t1	GetTaskState(t3, &TaskState)	E_OK, TaskState=READY	
t1	SetEvent(t3, Event3)	E_OK	10[1]
t1	GetEvent(t3, &EventMask)	E_OK, EventMask= Event3	
t1	SetEvent(t2, Event1)	E_OK	6
t2	ClearEvent(Event1)	E_OK	
t2	SetEvent(t1, Event1)	E_OK	10
t2	WaitEvent(Event1)	E_OK	
t1	ActivateTask(t4)	E_OK	

Running task	Called OS service	Return Status	Test case
t4	SetEvent(t2, Event1)	E_OK	7
t4	GetTaskState(t2, &TaskState)	E_OK, TaskState=READY	
t4	TerminateTask()		
t2	TerminateTask()		
t1	GetEvent(t1, &EventMask)	E_OK, EventMask= Event1	
t1	TerminateTask()		
t3	WaitEvent(Event3)	E_OK	
t3	TerminateTask()		

Test Sequence 5 :

```
TEST CASES:     25, 26, 27, 28, 29, 30, 34, 36, 37, 38, 39, 40
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:     NON-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 4;
  ACTIVATION = 1;
  SCHEDULE = NON;
  EVENT = Event1;
  EVENT = Event2;
};
TASK t2 {
  AUTOSTART =  TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON;
  EVENT = Event1;
};

TASK t3 {
  AUTOSTART =  FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = NON;
};

TASK t4 {
  AUTOSTART =  FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
  EVENT = Event1;
};

TASK t5 {
  AUTOSTART =  FALSE;
```

```
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
  EVENT = Event1;
};
ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};

EVENT Event1 {
  MASK = AUTO;
};
EVENT Event2 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	WaitEvent(Event1)	E_OK	
t2	ActivateTask(t3)	E_OK	
t2	ActivateTask(t5)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t3	*trigger interrupt isr1*		
isr1	GetEvent(INVALID_TASK, &EventMask)	E_OS_ID	36
isr1	GetEvent(t3, &EventMask)	E_OS_ACCESS	37
isr1	GetEvent(t4, &EventMask)	E_OS_STATE	38
isr1	GetEvent(t5, &EventMask)	E_OK	39
isr1	GetEvent(t2, &EventMask)	E_OK	40
isr1	SetEvent(INVALID_TASK, Event1)	E_OS_ID	25
isr1	SetEvent(t3, Event1)	E_OS_ACCESS	26
isr1	SetEvent(t4, Event1)	E_OS_STATE	27
isr1	SetEvent(t1, Event2)	E_OK	30
isr1	SetEvent(t1, Event1)	E_OK	28
isr1	SetEvent(t2, Event1)	E_OK	29
isr1	SetEvent(t5, Event1)	E_OK	34
t3	TerminateTask()		
t1	TerminateTask()		
t2	TerminateTask()		
t5	TerminateTask()		

Test Sequence 6 :

```
TEST CASES:      25, 26, 27, 31, 32, 33, 35, 36, 37, 38, 39, 40
RETURN STATUS:      EXTENDED
SCHEDULING POLICY:      FULL–PREEMPTIVE
```

```
TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 4;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
  EVENT = Event2;
};
TASK t2 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};

TASK t3 {
  AUTOSTART =  FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t4 {
  AUTOSTART =  FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};

TASK t5 {
  AUTOSTART =  FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};

EVENT Event1 {
  MASK = AUTO;
};
EVENT Event2 {
  MASK = AUTO;
```

};

Running task	Called OS service	Return Status	Test case
t1	WaitEvent(Event1)	E_OK	
t2	ActivateTask(t5)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t5	ActivateTask(t3)	E_OK	
t3	*trigger interrupt isr1*		
isr1	GetEvent(INVALID_TASK, &EventMask)	E_OS_ID	36
isr1	GetEvent(t3, &EventMask)	E_OS_ACCESS	37
isr1	GetEvent(t4, &EventMask)	E_OS_STATE	38
isr1	GetEvent(t5, &EventMask)	E_OK	39
isr1	GetEvent(t2, &EventMask)	E_OK	40
isr1	SetEvent(INVALID_TASK, Event1)	E_OS_ID	25
isr1	SetEvent(t3, Event1)	E_OS_ACCESS	26
isr1	SetEvent(t4, Event1)	E_OS_STATE	27
isr1	SetEvent(t1, Event2)	E_OK	33
isr1	SetEvent(t1, Event1)	E_OK	31
isr1	SetEvent(t2, Event1)	E_OK	32
isr1	SetEvent(t5, Event1)	E_OK	35
t1	TerminateTask()		
t3	TerminateTask()		
t2	TerminateTask()		
t5	TerminateTask()		

2.4 Resource management

The non-preemptive mode of test sequence 4 (cf OSEK Test Procedure [3] p38) has to be extended because it's forbidden to call Schedule() from standard mode (because the resource used is not saved when the task releases RES_SCHEDULER).

Test Sequence 1 :

```
TEST CASES:      1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  RESOURCE = source1;
  RESOURCE = Resource2;
  RESOURCE = Resource3;
  RESOURCE = Resource4;
```

```
  RESOURCE = Resource5;
  RESOURCE = Resource6;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  RESOURCE = ResourceA;
};

RESOURCE source1 {
  RESOURCEPROPERTY = STANDARD;
};
RESOURCE Resource2 {
  RESOURCEPROPERTY = STANDARD;
};
RESOURCE Resource3 {
  RESOURCEPROPERTY = STANDARD;
};
RESOURCE Resource4 {
  RESOURCEPROPERTY = STANDARD;
};
RESOURCE Resource5 {
  RESOURCEPROPERTY = STANDARD;
};
RESOURCE Resource6 {
  RESOURCEPROPERTY = STANDARD;
};
RESOURCE ResourceA {
  RESOURCEPROPERTY = STANDARD;
};
```

Running task	Called OS service	Return Status	Test case
t1	GetResource(ResourceA)	E_OK	4, 5
t1	ReleaseResource(ResourceA)	E_OK	
t1	GetResource(INVALID_RESOURCE)	E_OS_ID	1
t1	GetResource(Resource1)	E_OK	
t1	GetResource(Resource2)	E_OK	
t1	GetResource(Resource3)	E_OK	
t1	GetResource(Resource4)	E_OK	
t1	GetResource(Resource5)	E_OK	
t1	GetResource(Resource6)	E_OK	
t1	ReleaseResource(Resource6)	E_OK	
t1	ReleaseResource(Resource5)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	ReleaseResource(Resource4)	E_OK	
t1	ReleaseResource(Resource3)	E_OK	
t1	ReleaseResource(Resource1)	E_OS_NOFUNC	9
t1	ReleaseResource(Resource2)	E_OK	
t1	GetResource(Resource1)	E_OS_ACCESS	3
t1	ActivateTask(t2)	E_OK	
t1	*force scheduling*		
t2	GetResource(Resource2)	E_OS_ACCESS	2
t2	ReleaseResource(Resource1)	E_OS_ACCESS	10
t2	TerminateTask()		
t1	ReleaseResource(Resource1)	E_OK	11, 12
t1	ReleaseResource(INVALID_RESOURCE)	E_OS_ID	7
t1	ReleaseResource(Resource1)	E_OS_NOFUNC	8
t1	ReleaseResource(RES_SCHEDULER)	E_OS_NOFUNC	8
t1	TerminateTask()		

Test Sequence 2 :

```
TEST CASES:      4, 11
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:    NON-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
  RESOURCE = Resource1;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON;
  RESOURCE = Resource1;
};

TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = NON;
};

RESOURCE Resource1{
  RESOURCEPROPERTY = STANDARD;
};
```

Running task	Called OS service	Return Status	Test case
t1	GetResource(Resource1)	E_OK	5
t1	ActivateTask(t2)	E_OK	
t1	Schedule()	E_OS_RESOURCE	
t1	ActivateTask(t3)	E_OK	
t1	Schedule()	E_OS_RESOURCE	
t1	ReleaseResource(Resource1)	E_OK	11
t1	Schedule()	E_OK	
t3	TerminateTask()		
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 3 :

```
TEST CASES:     5, 12
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:     FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = Resource1;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = Resource1;
};

TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

RESOURCE Resource1{
  RESOURCEPROPERTY = STANDARD;
};
```

Running task	Called OS service	Return Status	Test case

Running task	Called OS service	Return Status	Test case
t1	GetResource(Resource1)	E_OK	5
t1	ActivateTask(t2)	E_OK	
t1	ActivateTask(t3)	E_OK	
t3	TerminateTask()		
t1	ReleaseResource(Resource1)	E_OK	12
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 4 :

```
TEST CASES:      6, 13, 14
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};


TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
```

Running task	Called OS service	Return Status	Test case
t1	GetResource(RES_SCHEDULER)	E_OK	6
t1	ActivateTask(t2)	E_OK	
t1	*force scheduling*	E_OS_RESOURCE, None	
t1	ActivateTask(t3)	E_OK	
t1	*force scheduling*	E_OS_RESOURCE, None	
t1	ReleaseResource(RES_SCHEDULER)	E_OK	13, 14
t1	*force scheduling*	E_OK, None	
t3	TerminateTask()		

Running task	Called OS service	Return Status	Test case
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 5 :

```
TEST CASES:     14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:     FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
  RESOURCE = Resource3;
};
ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
  RESOURCE = Resource1;
  RESOURCE = Resource2;
};
ISR isr2 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 2;
  SOURCE = SIGUSR2;
  RESOURCE = Resource1;
};

ISR isr3 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 3;
  SOURCE = SIGPIPE;
};

RESOURCE Resource1 {
  RESOURCEPROPERTY = STANDARD;
};
RESOURCE Resource2 {
  RESOURCEPROPERTY = STANDARD;
};
RESOURCE Resource3 {
  RESOURCEPROPERTY = STANDARD;
};
```

Running task	Called OS service	Return Status	Test case
t1	*trigger interrupt isr1*		
isr1	GetResource(INVALID_RESOURCE)	E_OS_ID	15
isr1	GetResource(Resource3)	E_OS_ACCESS	16
isr1	GetResource(Resource1)	E_OK	19
isr1	GetResource(Resource1)	E_OS_ACCESS	17
isr1	GetResource(RES_SCHEDULER)	E_OS_ACCESS	18
isr1	GetResource(Resource2)	E_OK	
isr1	*trigger interrupt isr2*		
isr1	*trigger interrupt isr3*		
isr3	ReleaseResource(Resource1)	E_OS_ACCESS	23
isr1	ReleaseResource(Resource1)	E_OS_NOFUNC	22
isr1	ReleaseResource(Resource2)	E_OK	25
isr1	ReleaseResource(Resource1)	E_OK	25
isr2			
isr1	ReleaseResource(INVALID_RESOURCE)	E_OS_ID	20
isr1	ReleaseResource(Resource1)	E_OS_NOFUNC	21
isr1	ReleaseResource(RES_SCHEDULER)	E_OS_ACCESS	24
t1	TerminateTask()		

2.5 Alarm

This section comes from OSEK/VDX Test Procedure [3] in which we bring some modifications (see Trampoline Test Implementation [4] - Test code organisation and distribution - Alarm Test Sequences).
Test case 35 can not be tested, because it is not possible to trigger the alarm's counter while no task is running.
Since no API service calls are allowed in callback routine, test sequence 10 appears.

Test Sequence 1 :

```
TEST CASES:            1, 3, 8, 12, 13, 14, 15, 19, 23, 24, 25, 26, 30
RETURN STATUS:         EXTENDED
SCHEDULING POLICY:     NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

COUNTER Counter1 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 2;
```

```
};

ALARM Alarm1 {
  COUNTER = Counter1;
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = FALSE;
};
```

Running task	Called OS service	Return Status	Test case
t1	GetAlarmBase(INVALID_ALARM, &AlarmBase)	E_OS_ID	1
t1	GetAlarm(INVALID_ALARM, &Tick)	E_OS_ID	3
t1	GetAlarmBase(Alarm1, &AlarmBase)	E_OK	
t1	SetRelAlarm(INVALID_ALARM, 0, 0)	E_OS_ID	8
t1	SetRelAlarm(Alarm1, -1, 0)	E_OS_VALUE	12
t1	SetRelAlarm(Alarm1, AlarmBase.maxallowedvalue+1, 0)	E_OS_VALUE	13
t1	SetRelAlarm(Alarm1, 0, AlarmBase.mincycle-1)	E_OS_VALUE	14
t1	SetRelAlarm(Alarm1, 0, AlarmBase.maxallowedvalue+1)	E_OS_VALUE	15
t1	SetAbsAlarm(INVALID_ALARM, 0, 0)	E_OS_ID	19
t1	SetAbsAlarm(Alarm1, -1, 0)	E_OS_VALUE	23
t1	SetAbsAlarm(Alarm1, AlarmBase.maxallowedvalue+1, 0)	E_OS_VALUE	24
t1	SetAbsAlarm(Alarm1, 0, AlarmBase.mincycle-1)	E_OS_VALUE	25
t1	SetAbsAlarm(Alarm1, 0, AlarmBase.maxallowedvalue+1)	E_OS_VALUE	26
t1	CancelAlarm(INVALID_ALARM)	E_OS_ID	30
t1	TerminateTask()		

Test Sequence 2 :

```
TEST CASES:          2, 4, 5, 9, 16, 20, 27, 31, 32
RETURN STATUS:       STANDARD, EXTENDED
SCHEDULING POLICY:   NON–, MIXED–, FULL–PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

TASK t2 {
```

```
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

COUNTER Counter1{
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};

ALARM Alarm1{
  COUNTER = Counter1;
  ACTION = ACTIVATETASK {
    TASK = t2;
  };
  AUTOSTART = FALSE;
};
```

Running task	Called OS service	Return Status	Test case
t1	GetAlarmBase(Alarm1, &AlarmBase)	E_OK, MAXALLOWEDVALUE=16, TICKSPERBASE=10, MINCYCLE=1	2
t1	GetAlarm(Alarm1, &Tick)	E_OS_NOFUNC	4
t1	CancelAlarm(Alarm1)	E_OS_NOFUNC	31
t1	SetAbsAlarm(Alarm1, 2, 2)	E_OK	27
t1	SetAbsAlarm(Alarm1, 3, 0)	E_OS_STATE	20
t1	*Wait alarm expires & Force scheduling*		
t2	TerminateTask()		
t1	*Wait alarm expires & Force scheduling*		
t2	TerminateTask()		
t1	CancelAlarm(Alarm1)	E_OK	32
t1	SetRelAlarm(Alarm1, 2, 0)	E_OK	16
t1	SetRelAlarm(Alarm1, 3, 0)	E_OS_STATE	9
t1	GetAlarm(Alarm1, &Tick)	E_OK, Tick=2	5
t1	*Wait alarm expires & Force scheduling*		
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 3 :

```
TEST CASES:           6, 10, 17, 21, 28, 33, 40
RETURN STATUS:        STANDARD, EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
```

```
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

TASK t2 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event2;
};

COUNTER Counter1 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};

ALARM Alarm1 {
  COUNTER = Counter1;
  ACTION = SETEVENT {
    TASK = t2;
    EVENT = Event2;
  };
  AUTOSTART = FALSE;
};

EVENT Event2 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t2	WaitEvent(Event2)	E_OK	
t1	SetAbsAlarm(Alarm1, 2, 2)	E_OK	28
t1	SetAbsAlarm(Alarm1, 3, 0)	E_OS_STATE	21
t1	*Wait alarm expires & Force scheduling*		40
t2	ClearEvent(Event2)	E_OK	
t2	WaitEvent(Event2)	E_OK	
t1	*Wait alarm expires & Force scheduling*		40
t2	ClearEvent(Event2)	E_OK	
t2	WaitEvent(Event2)	E_OK	
t1	CancelAlarm(Alarm1)	E_OK	33
t1	SetRelAlarm(Alarm1, 2, 0)	E_OK	17
t1	SetRelAlarm(Alarm1, 3, 0)	E_OS_STATE	10
t1	GetAlarm(Alarm1, &Tick)	E_OK, Tick = 2	6

Running task	Called OS service	Return Status	Test case
t1	*Wait alarm expires & Force scheduling*		40
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 4 :

```
TEST CASES:            7, 11, 18, 22, 29, 34
RETURN STATUS:         STANDARD, EXTENDED
SCHEDULING POLICY:     NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};


COUNTER Counter1 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};

ALARM Alarm1 {
  COUNTER = Counter1;
  ACTION = ALARMCALLBACK {
    ALARMCALLBACKNAME = "CallBackC";
  };
  AUTOSTART = FALSE;
};
```

Running task	Called OS service	Return Status	Test case
t1	SetAbsAlarm(Alarm1, 2, 2)	E_OK	29
t1	SetAbsAlarm(Alarm1, 3, 0)	E_OS_STATE	22
t1	*Wait alarm expires*		43
CallBack			
t1	*Wait alarm expires*		
CallBack			43
t1	CancelAlarm(Alarm1)	E_OK	34
t1	SetRelAlarm(Alarm1, 2, 0)	E_OK	18
t1	SetRelAlarm(Alarm1, 3, 0)	E_OS_STATE	11
t1	GetAlarm(Alarm1, &Tick)	E_OK, Tick=2	7
t1	*Wait alarm expires*		

Running task	Called OS service	Return Status	Test case
CallBack			43
t1	TerminateTask()		

Test Sequence 5 :

```
TEST CASES:            36
RETURN STATUS:         STANDARD, EXTENDED
SCHEDULING POLICY:     NON-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON;
};

COUNTER Counter1 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};

ALARM Alarm1 {
  COUNTER = Counter1;
  ACTION = ACTIVATETASK {
    TASK = t2;
  };
  AUTOSTART = FALSE;
};
```

Running task	Called OS service	Return Status	Test case
t1	SetRelAlarm(Alarm1, 0, 0)	E_OK	
t1	*Wait alarm expires*		36
t1	GetTaskState(t2, &TaskState)	E_OK, TaskState=READY	
t1	TerminateTask()		
t2	TerminateTask()		

Test Sequence 6 :

```
TEST CASES:            37, 38
RETURN STATUS:         STANDARD, EXTENDED
SCHEDULING POLICY:     FULL-PREEMPTIVE
```

```
TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

COUNTER Counter1 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};

ALARM Alarm1{
  COUNTER = Counter1;
  ACTION = ACTIVATETASK {
    TASK = t2;
  };
  AUTOSTART = FALSE;
};
```

Running task	Called OS service	Return Status	Test case
t1	SetRelAlarm(Alarm1, 2, 0)	E_OK	
t1	*Wait alarm expires*		37
t2	TerminateTask()		
t1	ChainTask(t3)		
t3	SetRelAlarm(Alarm1, 2, 0)	E_OK	
t3	*Wait alarm expires*		38
t3	GetTaskState(t2, &State)	E_OK, TaskState=READY	
t3	TerminateTask()		
t2	TerminateTask()		

Test Sequence 7 :

TEST CASES: 39, 40

```
RETURN STATUS:          STANDARD, EXTENDED
SCHEDULING POLICY:      NON-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON;
  EVENT = Event1;
};
COUNTER Counter1 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};
ALARM Alarm1 {
  COUNTER = Counter1;
  ACTION = SETEVENT {
    TASK = t2;
    EVENT = Event1;
  };
  AUTOSTART = FALSE;
};
EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OK	
t1	SetRelAlarm(Alarm1, 2, 0)	E_OK	
t1	*Wait alarm expires*		40
t1	GetEvent(t2, &EventMask)	E_OK, EventMask=Event1	
t1	Schedule()	E_OK	
t2	ClearEvent(Event1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	SetRelAlarm(Alarm1, 2, 0)	E_OK	
t1	*Wait alarm expires*		39
t1	GetTaskState(t2, &TaskState)	E_OK, TaskState=READY	
t1	TerminateTask()		

Running task	Called OS service	Return Status	Test case
t2	TerminateTask()		

Test Sequence 8 :

```
TEST CASES:            41, 42
RETURN STATUS:         STANDARD, EXTENDED
SCHEDULING POLICY:     FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t2 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event2;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t4 {
  AUTOSTART = FALSE;
  PRIORITY = 4;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

COUNTER Counter1 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};

ALARM Alarm1{
  COUNTER = Counter1;
  ACTION = SETEVENT {
    TASK = t2;
    EVENT = Event2;
  };
  AUTOSTART = FALSE;
};
```

```
EVENT Event2{
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t2	WaitEvent(Event2)	E_OK	
t1	ActivateTask(t3)	E_OK	
t3	SetRelAlarm(Alarm1, 2, 0)	E_OK	
t3	*Wait alarm expires*		41
t3	GetTaskState(t2, &TaskState)	E_OK, TaskState=READY	
t3	TerminateTask()		
t2	ClearEvent(Event2)	E_OK	
t2	ActivateTask(t4)	E_OK	
t4	SetRelAlarm(Alarm1, 2, 0)	E_OK	
t4	*Wait alarm expires*		42
t4	GetEvent(t2, &EventMask)	E_OK, EventMask=Event2	
t4	TerminateTask()		
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 9 :

All alarm routines are allowed from ISR2. Test cases from 1 to 34 are tested from ISR2 in this sequence.

Test case 35 can not be tested, because it is not possible to trigger the alarm's counter while no task is running.

```
TEST CASES:            1 to 34
RETURN STATUS:         EXTENDED
SCHEDULING POLICY:     NON-, MIXED-, FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 4;
```

```
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event1;
};

TASK t4 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event1;
};

TASK t5 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};
COUNTER Counter0 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 2;
};

ALARM Alarm0 {
  COUNTER = Counter0;
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = FALSE;
};
COUNTER Counter1 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};

ALARM Alarm1_1 {
  COUNTER = Counter1;
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = FALSE;
```

```
};

ALARM Alarm1_2 {
  COUNTER = Counter1;
  ACTION = ACTIVATETASK {
    TASK = t2;
  };
  AUTOSTART = FALSE;
};

ALARM Alarm2_1 {
  COUNTER = Counter1;
  ACTION = SETEVENT {
    TASK = t3;
    EVENT = Event1;
  };
  AUTOSTART = FALSE;
};

ALARM Alarm2_2 {
  COUNTER = Counter1;
  ACTION = SETEVENT {
    TASK = t4;
    EVENT = Event1;
  };
  AUTOSTART = FALSE;
};

ALARM Alarm3 {
  COUNTER = Counter1;
  ACTION = ALARMCALLBACK {
    ALARMCALLBACKNAME = "CallBackC";
  };
  AUTOSTART = FALSE;
};
EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t5	*trigger interrupt isr1*		
isr1	ActivateTask(t4)	E_OK	
isr1	ActivateTask(t3)	E_OK	
isr1	GetAlarmBase(INVALID_ALARM, &Alarm-Base)	E_OS_ID	1
isr1	GetAlarmBase(Alarm0, &AlarmBase)	E_OK	2

Running task	Called OS service	Return Status	Test case
isr1	GetAlarm(INVALID_ALARM, &Tick)	E_OS_ID	3
isr1	GetAlarm(Alarm0, &Tick)	E_OS_NOFUNC	4
isr1	SetRelAlarm(INVALID_ALARM, 0, 0)	E_OS_ID	8
isr1	SetRelAlarm(Alarm0, -1, 0)	E_OS_VALUE	12
isr1	SetRelAlarm(Alarm0, AlarmBase.maxallowedvalue+1, 0)	E_OS_VALUE	13
isr1	SetRelAlarm(Alarm0, 0, AlarmBase.mincycle-1)	E_OS_VALUE	14
isr1	SetRelAlarm(Alarm0, 0, AlarmBase.maxallowedvalue+1)	E_OS_VALUE	15
isr1	SetAbsAlarm(INVALID_ALARM, 0, 0)	E_OS_ID	19
isr1	SetAbsAlarm(Alarm0, -1, 0)	E_OS_VALUE	23
isr1	SetAbsAlarm(Alarm0, AlarmBase.maxallowedvalue+1, 0)	E_OS_VALUE	24
isr1	SetAbsAlarm(Alarm0, 0, AlarmBase.mincycle-1)	E_OS_VALUE	25
isr1	SetAbsAlarm(Alarm0, 0, AlarmBase.maxallowedvalue+1)	E_OS_VALUE	26
isr1	CancelAlarm(INVALID_ALARM)	E_OS_ID	30
isr1	CancelAlarm(Alarm0)	E_OS_NOFUNC	31
isr1	SetRelAlarm(Alarm1_1, 2 , 2)	E_OK	16
isr1	SetRelAlarm(Alarm1_1, 2 , 2)	E_OS_STATE	9
isr1	GetAlarm(Alarm1_1, &Tick)	E_OK, Tick=2	5
isr1	*Wait alarm expires*		
isr1	CancelAlarm(Alarm1_1)	E_OK	32
isr1	SetAbsAlarm(Alarm1_2, 2 , 2)	E_OK	27
isr1	SetAbsAlarm(Alarm1_2, 2 , 2)	E_OS_STATE	20
isr1	*Wait alarm expires*		
isr1	SetRelAlarm(Alarm2_1, 2 , 2)	E_OK	17
isr1	SetRelAlarm(Alarm2_1, 2 , 2)	E_OS_STATE	10
isr1	GetAlarm(Alarm2_1, &Tick)	E_OK, Tick=2	6
isr1	*Wait alarm expires*		
isr1	CancelAlarm(Alarm2_1)	E_OK	33
isr1	SetAbsAlarm(Alarm2_2, 2 , 2)	E_OK	28
isr1	SetAbsAlarm(Alarm2_2, 2 , 2)	E_OS_STATE	21
isr1	*Wait alarm expires*		
isr1	SetRelAlarm(Alarm3, 2 , 2)	E_OK	18
isr1	SetRelAlarm(Alarm3, 2 , 2)	E_OS_STATE	11
isr1	GetAlarm(Alarm3, &Tick)	E_OK, Tick=2	7
isr1	*Wait alarm expires*		

Running task	Called OS service	Return Status	Test case
CallBack			
isr1	CancelAlarm(Alarm3)	E_OK	34
isr1	SetAbsAlarm(Alarm1_2, 2 , 2)	E_OK	29
isr1	SetAbsAlarm(Alarm1_2, 2 , 2)	E_OS_STATE	22
isr1	*Wait alarm expires*		
CallBack			
{NON}t5	TerminateTask()		
t3	TerminateTask()		
{FULL}t5	TerminateTask()		
t4	TerminateTask()		
t1	TerminateTask()		
t2	TerminateTask()		

Test Sequence 10 :

This test sequence should return E_OS_CALLEVEL (instead of E_OK) because service calls in call-back routines are forbidden!!

```
TEST CASES:            41, 42
RETURN STATUS:         STANDARD, EXTENDED
SCHEDULING POLICY:     FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Counter1 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};

ALARM Alarm1 {
  COUNTER = Counter1;
  ACTION = ALARMCALLBACK {
    ALARMCALLBACKNAME = "CallBackC";
  };
  AUTOSTART = FALSE;
};
```

Running task	Called OS service	Return Status	Test case
t1	SetRelAlarm(Alarm1, 2, 0)	E_OK	
t1	*Wait alarm expires*		
CallBack	ActivateTask(t2)	E_OS_CALLEVEL	
t1	TerminateTask()		

Test Sequence 11 :

```
TEST CASES:              ...
RETURN STATUS:           STANDARD, EXTENDED
SCHEDULING POLICY:       FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Counter1 {
  MAXALLOWEDVALUE = 15;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};
ALARM Alarm1 {
  COUNTER = Counter1;
  ACTION = ACTIVATETASK {
    TASK = t2;
  };
  AUTOSTART = TRUE {
    ALARMTIME = 7;
    CYCLETIME = 0;
    APPMODE = std;
  };
};
ALARM Alarm2 {
  COUNTER = Counter1;
  ACTION = ACTIVATETASK {
```

```
    TASK = t3;
  };
  AUTOSTART = TRUE {
    ALARMTIME = 15;
    CYCLETIME = 15;
    APPMODE = std;
  };
};
```

Running task	Called OS service	Return Status	Test case
t1	*Wait alarm expires*		
t2	TerminateTask()		
t1	*Wait alarm expires*		
t3	TerminateTask()		
t1	CancelAlarm(Alarm2)	E_OK	
t1	TerminateTask()		

2.6 Error handling, hook routines and OS execution control

Test case 2 (call StartOS() to start OSEK OS) can not be tested, because the startup code is implementation specific.
Test case 27 doesn't appear because activation of an ISR is one and this ISR has been already activated (see test sequence 4).
As you can see in test sequences 4, 5 and 6, each time we send an interrupt, the following code is inserted :

```
trigger interrupt isr1
SuspendAllInterrupts()
trigger interrupt isr1
ResumeAllInterrupts()
```

The first triggering interrupt, tests if interrupts are allowed in Post-Pre/taskhook because it shouldn't.
The second is between Suspend-Resume/AllInterrupts() and tests those two service calls.
Since few API service calls are allowed in hook routines, test sequence 7 appears.

Test Sequence 1 :

```
TEST CASES:      1, 3, 7
RETURN STATUS:     STANDARD, EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE
HOOKS:          StartupHook, ShutdownHook

TASK t1 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};
```

Running task	Called OS service	Return Status	Test case
Startup-Hook	GetActiveApplicationMode()	OSDEFAULTAPPMODE	1, 6
Startup-Hook	ShutdownOS()		3
Shutdown-Hook	GetActiveApplicationMode()	OSDEFAULTAPPMODE	1, 7

Test Sequence 2 :

```
TEST CASES:      1, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE
HOOKS:          StartupHook, ShutdownHook, ErrorHook, PostTaskHook, PreTaskHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event1;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

COUNTER Counter1 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};

ALARM Alarm1 {
  COUNTER = Counter1;
  ACTION = SETEVENT {
    TASK = t1;
    EVENT = Event1;
  };
  AUTOSTART = FALSE;
};

EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
PreTask-Hook	GetActiveApplicationMode()	OSDEFAULTAPPMODE	1, 4
PreTask-Hook	GetTaskID()	E_OK, TaskID=t1	8
PreTask-Hook	GetTaskState()	E_OK, TaskState=RUNNING	9
PreTask-Hook	GetEvent()	E_OK, EventMask=0x0	12
PreTask-Hook	GetAlarmBase()	E_OK, MAXALLOWEDVALUE=16, TICKSPERBASE=10, MINCYCLE=1	13
PreTask-Hook	GetAlarm()	E_OS_NOFUNC	
ErrorHook			
t1	SetAbsAlarm(Alarm1, MAXALLOWEDVALUE, 0)	E_OK	
t1	WaitEvent(Event1)	E_OK	
PostTask-Hook	GetTaskID()	E_OK, TaskID=t1	8
PostTask-Hook	GetTaskState()	E_OK, TaskState=RUNNING	9
PostTask-Hook	GetEvent()	E_OK, EventMask=0x0	12
PostTask-Hook	GetAlarmBase()	E_OK, MAXALLOWEDVALUE=16, TICKSPERBASE=10, MINCYCLE=1	13
PostTask-Hook	GetAlarm()	E_OK, Tick=MAXALLOWEDVALUE	14
PreTask-Hook	GetTaskID()	E_OK, TaskID=INVALID_TASK	8
PreTask-Hook	GetTaskState()	E_OS_ID	
ErrorHook			
PreTask-Hook	GetEvent()	E_OS_ID	
ErrorHook			
PreTask-Hook	GetAlarmBase()	E_OK, MAXALLOWEDVALUE=16, TICKSPERBASE=10, MINCYCLE=1	13
PreTask-Hook	GetAlarm()	E_OK, Tick=MAXALLOWEDVALUE	14
Idle	*Wait for the alarm*		
PostTask-Hook	GetTaskID()	E_OK, TaskID=INVALID_TASK	8
PostTask-Hook	GetAlarm()	E_OS_NOFUNC	
ErrorHook			
PreTask-Hook	GetTaskID()	E_OK, TaskID=t1	8
PreTask-Hook	GetTaskState()	E_OK, TaskState=RUNNING	9
PreTask-Hook	GetEvent()	E_OK, EventMask=0x0	12
PreTask-Hook	GetAlarmBase()	E_OK, MAXALLOWEDVALUE=16, TICKSPERBASE=10, MINCYCLE=1	13
PreTask-Hook	GetAlarm()	E_OS_NOFUNC	
ErrorHook			
t1	SetEvent(t1, &Event1)	E_OK	
t1	ChainTask(t2)		
PostTask-Hook	GetTaskID()	E_OK, TaskID=t1	8

Running task	Called OS service	Return Status	Test case
PostTask-Hook	GetTaskState()	E_OK, TaskState=RUNNING	9
PostTask-Hook	GetEvent()	E_OK, EventMask=Event1	12
PreTask-Hook	GetTaskID()	E_OK, TaskID=t2	8
PreTask-Hook	GetTaskState()	E_OK, TaskState=RUNNING	9
PreTask-Hook	GetEvent()	E_OS_ACCESS	
ErrorHook			
t2	ShutdownOS()		3
Shutdown-Hook			7

Test Sequence 3 :

```
TEST CASES:      1, 5, 8, 9, 12, 13, 14
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE
HOOKS:          StartupHook, ShutdownHook, ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  EVENT = Event1;
};

COUNTER Counter1 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 10;
  MINCYCLE = 1;
};

ALARM Alarm1 {
  COUNTER = Counter1;
  ACTION = SETEVENT {
    TASK = t1;
    EVENT = Event1;
  };
  AUTOSTART = FALSE;
};

EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	SetAbsAlarm(Alarm1, MAXALLOWEDVALUE, MAXALLOWEDVALUE)	E_OK	
t1	WaitEvent(Event1)	E_OK	
Idle	*Wait for the alarm*		
t1	SetAbsAlarm(Alarm1, 2, 2)	E_OS_STATE	
Error-Hook	GetActiveApplicationMode()	OSDEFAULTAPPMODE	1, 5
Error-Hook	GetTaskID()	E_OK, TaskID=t1	8
Error-Hook	GetTaskState()	E_OK, TaskState=RUNNING	9
Error-Hook	GetEvent()	E_OK, EventMask=0x0	12
Error-Hook	GetAlarmBase()	E_OK, MAXALLOWEDVALUE=16, TICKSPERBASE=10, MINCYCLE=1	13
Error-Hook	GetAlarm()	E_OK	14
t1	TerminateTask()		

Test Sequence 4 :

```
TEST CASES:     10, 11, 17, 18, 20, 23, 24, 28, 30, 33, 34
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE
HOOKS:          PostTaskHook, PreTaskHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};
ISR isr2 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 2;
  SOURCE = SIGUSR2;
};
```

Running task	Called OS service	Return Status	Test case
PreTask-Hook			
t1	ChainTask(t2)		
PostTask-Hook			
PreTask-Hook	*trigger interrupt isr1*		
PreTask-Hook	SuspendAllInterrupts()		10
PreTask-Hook	*trigger interrupt isr1*		20
PreTask-Hook	ResumeAllInterrupts()		11
PostTask-Hook	*trigger interrupt isr1*		
PostTask-Hook	SuspendAllInterrupts()		
PostTask-Hook	*trigger interrupt isr1*		17
PostTask-Hook	ResumeAllInterrupts()		
PreTask-Hook	*trigger interrupt isr2*		
PreTask-Hook	SuspendAllInterrupts()		
PreTask-Hook	*trigger interrupt isr2*		18
PreTask-Hook	ResumeAllInterrupts()		
PostTask-Hook			
PreTask-Hook			
isr2	GetActiveApplicationMode()	OSDEFAULTAPPMODE	1, 28
PostTask-Hook			
PreTask-Hook			
isr1			30
PostTask-Hook	*trigger interrupt isr1*		
PostTask-Hook	SuspendAllInterrupts()		
PostTask-Hook	*trigger interrupt isr1*		23
PostTask-Hook	ResumeAllInterrupts()		
PreTask-Hook	*trigger interrupt isr2*		
PreTask-Hook	SuspendAllInterrupts()		
PreTask-Hook	*trigger interrupt isr2*		24
PreTask-Hook	ResumeAllInterrupts()		
PostTask-Hook			
PreTask-Hook			
PostTask-Hook			
PreTask-Hook			
isr2			34
PostTask-Hook			
PreTask-Hook			
isr1			33
PostTask-Hook			
PreTask-Hook			

Running task	Called OS service	Return Status	Test case
t2	TerminateTask()		

Test Sequence 5 :

```
TEST CASES:     10, 11, 15, 16, 21, 22, 25, 26, 31, 32
RETURN STATUS:     EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE
HOOKS:          PostTaskHook, PreTaskHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};

COUNTER Counter1 {
  MAXALLOWEDVALUE = 16;
  TICKSPERBASE = 100;
  MINCYCLE = 1;
};

ALARM Alarm1 {
  COUNTER = Counter1;
  ACTION = ACTIVATETASK {
    TASK = t2;
  };
  AUTOSTART = FALSE;
};
```

Running task	Called OS service	Return Status	Test case

Running task	Called OS service	Return Status	Test case
PreTask-Hook			
t1	SetRelAlarm(Alarm1, 2, 0)	E_OK	
t1	*Wait alarm expires & Force scheduling*		
PostTask-Hook	*trigger interrupt isr1*		
PostTask-Hook	SuspendAllInterrupts()		
PostTask-Hook	*trigger interrupt isr1*		15
PostTask-Hook	ResumeAllInterrupts()		
PreTask-Hook			
PostTask-Hook			
PreTask-Hook			
isr1			25
PostTask-Hook			
PreTask-Hook			
t2	TerminateTask()		
PostTask-Hook	*trigger interrupt isr1*		
PostTask-Hook	SuspendAllInterrupts()		
PostTask-Hook	*trigger interrupt isr1*		21
PostTask-Hook	ResumeAllInterrupts()		
PreTask-Hook			
PostTask-Hook			
PreTask-Hook			
isr1			31
PostTask-Hook			
PreTask-Hook			
t1	SetRelAlarm(Alarm1, 2, 0)	E_OK	
t1	*Wait alarm expires & Force scheduling*		
PostTask-Hook			
PreTask-Hook	*trigger interrupt isr1*		
PreTask-Hook	SuspendAllInterrupts()		
PreTask-Hook	*trigger interrupt isr1*		16
PreTask-Hook	ResumeAllInterrupts()		
PostTask-Hook			
PreTask-Hook			
isr1			26
PostTask-Hook			
PreTask-Hook			
t2	TerminateTask()		
PostTask-Hook			
PreTask-Hook	*trigger interrupt isr1*		
PreTask-Hook	SuspendAllInterrupts()		

Running task	Called OS service	Return Status	Test case
PreTask-Hook	*trigger interrupt isr1*		22
PreTask-Hook	ResumeAllInterrupts()		
PostTask-Hook			
PreTask-Hook			
isr1			32
PostTask-Hook			
PreTask-Hook			
t1	TerminateTask()		

Test Sequence 6 :

```
TEST CASES:      10, 11, 19, 29, 35, 36
RETURN STATUS:      EXTENDED
SCHEDULING POLICY:    NON-, MIXED-, FULL-PREEMPTIVE
HOOKS:          ErrorHook, PostTaskHook, PreTaskHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
};

ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};
ISR isr2{
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 2;
  SOURCE = SIGUSR2;
};
```

Running task	Called OS service	Return Status	Test case
PreTask-Hook			
t1	ActivateTask(T1)	E_OS_LIMIT	
Error-Hook	*trigger interrupt isr1*		
Error-Hook	SuspendAllInterrupts()		
Error-Hook	*trigger interrupt isr1*		35
Error-Hook	ResumeAllInterrupts()		
PostTask-Hook			

Running task	Called OS service	Return Status	Test case
PreTask-Hook			
isr1			36
PostTask-Hook			
PreTask-Hook			
t1	ChainTask(t1)		
PostTask-Hook	*trigger interrupt isr2*		
PostTask-Hook	SuspendAllInterrupts()		
PostTask-Hook	*trigger interrupt isr2*		19
PostTask-Hook	ResumeAllInterrupts()		
PreTask-Hook			
PostTask-Hook			
PreTask-Hook			
isr2	ShutdownOS()		3, 29

2.7 Internal COM

Message flag mechanism isn't implement yet (test sequence 3, test case 25).
Since no API service calls are allowed in COM callback routines, tests are inserted in test sequence 3.
Since "Never" filter block all messages, it's not possible to receive a message with this filter. Thus, test case 9 is never tested.

Test Sequence 1 :

```
TEST CASES:             1, 2, 5, 6, 31, 36, 37, 38, 39, 40, 41, 42, 43
RETURN COM STATUS:      EXTENDED
SCHEDULING POLICY:      NON-, MIXED-, FULL-PREEMPTIVE
HOOKS:                  COMErrorHook
MESSAGE TYPE:           Unqueued

TASK t1 {
  PRIORITY = 1;
  AUTOSTART = TRUE { APPMODE = std; };
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = sm;
};
TASK t2 {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm;
};

MESSAGE sm {
```

```
    MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
      CDATATYPE = "uint8";
    };
    NOTIFICATION = NONE;
  };

  MESSAGE rm {
    MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
      SENDINGMESSAGE = sm;
      INITIALVALUE = 0;
      FILTER = ALWAYS;
    };
    NOTIFICATION = NONE;
  };
```

Running task	Called OS service	Return Status	Test case
t1	GetMessageStatus(sm)	E_COM_ID	31
COMErrorHook	COMErrorGetServiceId()	GetMessageStatusID	42
COMErrorHook	COMError_GetMessageStatus_Message()	sm_id	43
t1	SendMessage(INVALID_MESSAGE, "3")	E_COM_ID	2
COMErrorHook	COMErrorGetServiceId()	SendMessageID	36
COMErrorHook	COMError_SendMessage_DataRef()	"3"	38
COMErrorHook	COMError_SendMessage_Message()	INVALID_MESSAGE	37
t1	SendMessage(sm, "0")	E_OK	1
t1	SendMessage(sm, "1")	E_OK	
t1	ActivateTask(t2)	E_OK	
t1	*Force scheduling*		
t2	GetMessageStatus(rm)	E_COM_ID	31
COMErrorHook	COMErrorGetServiceId()	GetMessageStatusID	
COMErrorHook	COMError_GetMessageStatus_Message()	rm_id	
t2	ReceiveMessage(rm, & DataRef)	E_OK, DataRef="1"	6
t2	ReceiveMessage(rm, & DataRef)	E_OK, DataRef="1"	
t2	ReceiveMessage(INVALID_MESSAGE, & DataRef)	E_COM_ID	5
COMErrorHook	COMErrorGetServiceId()	ReceiveMessageID	39
COMErrorHook	COMError_ReceiveMessage_DataRef()	"1"	41
COMErrorHook	COMError_ReceiveMessage_Message()	INVALID_MESSAGE	40
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 2 :

TEST CASES: 3, 4, 26, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43
RETURN COM STATUS: EXTENDED
SCHEDULING POLICY: NON–, MIXED–, FULL–PREEMPTIVE
HOOKS: COMErrorHook
MESSAGE TYPE: Queued

```
TASK t1 {
  PRIORITY = 1;
  AUTOSTART = TRUE { APPMODE = std; };
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = sm;
};
TASK t2 {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm;
};

MESSAGE sm {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
    CDATATYPE = "uint8";
  };
  NOTIFICATION = NONE;
};

MESSAGE rm {
  MESSAGEPROPERTY = RECEIVE_QUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    QUEUESIZE = 3;
  };
  NOTIFICATION = NONE;
};
```

Running task	Called OS service	Return Status	Test case
t1	GetMessageStatus(sm)	E_COM_NOMSG	35
t1	SendMessage(INVALID_MESSAGE, "5")	E_COM_ID	4
COMErrorHook	COMErrorGetServiceId()	SendMessageID	36
COMErrorHook	COMError_SendMessage_DataRef()	"5"	38
COMErrorHook	COMError_SendMessage_Message()	INVALID_MESSAGE	37
t1	GetMessageStatus(sm)	E_COM_NOMSG	
t1	SendMessage(sm, "1")	E_OK	3
t1	GetMessageStatus(sm)	E_OK	32
t1	SendMessage(sm, "2")	E_OK	
t1	GetMessageStatus(sm)	E_OK	
t1	SendMessage(sm, "3")	E_OK	
t1	GetMessageStatus(sm)	E_OK	
t1	SendMessage(sm, "4")	E_OK	
t1	GetMessageStatus(sm)	E_COM_LIMIT	

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OK	
t1	*Force scheduling*		
t2	GetMessageStatus(rm)	E_COM_LIMIT	34
t2	ReceiveMessage(rm, & DataRef)	E_COM_LIMIT, DataRef="1"	27
COMErrorHook	COMErrorGetServiceId( )	ReceiveMessageID	39
COMErrorHook	COMError_ReceiveMessage_DataRef()	"1"	41
COMErrorHook	COMError_ReceiveMessage_Message()	rm_id	40
t2	GetMessageStatus(rm)	E_OK	32
t2	ReceiveMessage(rm, & DataRef)	E_OK, DataRef="2"	28
t2	GetMessageStatus(rm)	E_OK	
t2	ReceiveMessage(rm, & DataRef)	E_OK, DataRef="3"	30
t2	GetMessageStatus(rm)	E_COM_NOMSG	
t2	ReceiveMessage(rm, & DataRef)	E_COM_NOMSG	29
COMErrorHook	COMErrorGetServiceId()	ReceiveMessageID	
COMErrorHook	COMError_ReceiveMessage_DataRef()	"3"	
COMErrorHook	COMError_ReceiveMessage_Message()	rm_id	
t2	GetMessageStatus(rm)	E_COM_NOMSG	
t2	ReceiveMessage(INVALID_MESSAGE, & DataRef)	E_COM_ID	26
COMErrorHook	COMErrorGetServiceId()	ReceiveMessageID	
COMErrorHook	COMError_ReceiveMessage_DataRef()	"3"	
COMErrorHook	COMError_ReceiveMessage_Message()	INVALID_MESSAGE	
t2	GetMessageStatus(INVALID_MESSAGE)	E_COM_ID	33
COMErrorHook	COMErrorGetServiceId()	GetMessageStatusID	42
COMErrorHook	COMError_GetMessageStatus_Message()	INVALID_MESSAGE	43
t2	TerminateTask()		
t1	TerminateTask()		

Test Sequence 3 :

This test sequence should return E_OS_CALLEVEL (instead of E_OK) because service calls in call-back routines are forbidden!!

```
TEST CASES:            1, 7, 23, 24, 25
RETURN COM STATUS:     EXTENDED
SCHEDULING POLICY:     FULL-PREEMPTIVE
HOOKS:                 ...
MESSAGE TYPE:          Unqueued

TASK t1 {
  PRIORITY = 1;
  AUTOSTART = TRUE { APPMODE = std; };
  ACTIVATION = 1;
  SCHEDULE = FULL;
  MESSAGE = sm_activatetask;
  MESSAGE = sm_setevent;
  MESSAGE = sm_comcallback;
```

```
  MESSAGE = sm_flag;
  MESSAGE = rm_comcallback;
};
TASK t2 {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  MESSAGE = rm_activatetask;
};

TASK t3 {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  MESSAGE = rm_setevent;
  EVENT = Event1;
};

TASK t4 {
  PRIORITY = 1;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  MESSAGE = rm_flag;
};

MESSAGE sm_activatetask {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
    CDATATYPE = "uint8";
  };
  NOTIFICATION = NONE;
};

MESSAGE rm_activatetask {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm_activatetask;
    INITIALVALUE = 0;
    FILTER = ALWAYS;
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t2;
  };
};

MESSAGE sm_setevent {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
    CDATATYPE = "uint8";
  };
```

```
  NOTIFICATION = NONE;
};
MESSAGE rm_setevent {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm_setevent;
    INITIALVALUE = 0;
    FILTER = ALWAYS;
  };
  NOTIFICATION = SETEVENT {
    TASK = t3;
    EVENT = Event1;
  };
};

MESSAGE sm_comcallback {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
    CDATATYPE = "uint8";
  };
  NOTIFICATION = NONE;
};

MESSAGE rm_comcallback {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm_comcallback;
    INITIALVALUE = 0;
    FILTER = ALWAYS;
  };
  NOTIFICATION = COMCALLBACK {
    CALLBACKROUTINENAME = "ComCallBack";
  };
};
MESSAGE sm_flag {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
    CDATATYPE = "uint8";
  };
  NOTIFICATION = NONE;
};

MESSAGE rm_flag {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm_flag;
    INITIALVALUE = 0;
    FILTER = ALWAYS;
  };
  NOTIFICATION = FLAG {
    FLAGNAME = "flagnam";
  };
};
EVENT Event1 {
  MASK = AUTO;
```

};

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t3)	E_OK	
t3	WaitEvent(Event1)	E_OK	
t1	SendMessage(sm_activatetask, "1")	E_OK	1
t2	ReceiveMessage(rm_activatetask, & DataRef)	E_OK, DataRef="1"	7
t2	TerminateTask()		
t1	SendMessage(sm_setevent, "2")	E_OK	1
t3	ReceiveMessage(rm_setevent, & DataRef)	E_OK, DataRef="2"	23
t3	TerminateTask()		
t1	SendMessage(sm_comcallback, "3")	E_OK	1
COMCallBack	ReceiveMessage(rm_comcallback, & DataRef)	E_OS_CALLEVEL	24
t1	TerminateTask()		

Test Sequence 4 :

This test sequence should return E_OS_CALLEVEL (instead of E_OK) because service calls in call-back routines are forbidden!!

```
TEST CASES:              1, 7, 23, 24, 25
RETURN COM STATUS:       EXTENDED
SCHEDULING POLICY:       NON-PREEMPTIVE
HOOKS:                   ...
MESSAGE TYPE:            Unqueued

TASK t1 {
  PRIORITY = 1;
  AUTOSTART = TRUE { APPMODE = std; };
  ACTIVATION = 1;
  SCHEDULE = NON;
  MESSAGE = sm_activatetask;
  MESSAGE = sm_setevent;
  MESSAGE = sm_comcallback;
  MESSAGE = sm_flag;
  MESSAGE = rm_comcallback;
};
TASK t2 {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON;
  MESSAGE = rm_activatetask;
};

TASK t3 {
  PRIORITY = 2;
  AUTOSTART = FALSE;
```

```
  ACTIVATION = 1;
  SCHEDULE = NON;
  MESSAGE = rm_setevent;
  EVENT = Event1;
};

TASK t4 {
  PRIORITY = 1;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON;
  MESSAGE = rm_flag;
};

MESSAGE sm_activatetask {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
    CDATATYPE = "uint8";
  };
  NOTIFICATION = NONE;
};

MESSAGE rm_activatetask {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm_activatetask;
    INITIALVALUE = 0;
    FILTER = ALWAYS;
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t2;
  };
};

MESSAGE sm_setevent {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
    CDATATYPE = "uint8";
  };
  NOTIFICATION = NONE;
};

MESSAGE rm_setevent {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm_setevent;
    INITIALVALUE = 0;
    FILTER = ALWAYS;
  };
  NOTIFICATION = SETEVENT {
    TASK = t3;
    EVENT = Event1;
  };
};
```

```
MESSAGE sm_comcallback {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
    CDATATYPE = "uint8";
  };
  NOTIFICATION = NONE;
};

MESSAGE rm_comcallback {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm_comcallback;
    INITIALVALUE = 0;
    FILTER = ALWAYS;
  };
  NOTIFICATION = COMCALLBACK {
    CALLBACKROUTINENAME = "ComCallBack";
  };
};
MESSAGE sm_flag {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
    CDATATYPE = "uint8";
  };
  NOTIFICATION = NONE;
};

MESSAGE rm_flag {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm_flag;
    INITIALVALUE = 0;
    FILTER = ALWAYS;
  };
  NOTIFICATION = FLAG {
    FLAGNAME = "flagnam";
  };
};
EVENT Event1{
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t3)	E_OK	
t1	SendMessage(sm_activatetask, "1")	E_OK	1
t1	SendMessage(sm_setevent, "2")	E_OK	1
t1	SendMessage(sm_comcallback, "3")	E_OK	1
COMCallBack	ReceiveMessage(rm_comcallback, & DataRef)	E_OS_CALLEVEL	24

Running task	Called OS service	Return Status	Test case
t1	TerminateTask()		
t3	WaitEvent(Event1)	E_OK	
t3	ReceiveMessage(rm_setevent, & DataRef)	E_OK, DataRef="2"	23
t3	TerminateTask()		
t2	ReceiveMessage(rm_activatetask, & DataRef)	E_OK, DataRef="1"	7
t2	TerminateTask()		

Test Sequence 5 :

```
TEST CASES:              8, 18, 19, 20, 21, 22
RETURN COM STATUS:       EXTENDED
SCHEDULING POLICY:       NON-, MIXED-, FULL-PREEMPTIVE
HOOKS:                   ...
MESSAGE TYPE:            Unqueued

TASK t1 {
  PRIORITY = 1;
  AUTOSTART = TRUE { APPMODE = std; };
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = sm;
};
TASK t2 {
  PRIORITY = 8;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_always;
};

TASK t3 {
  PRIORITY = 7;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_never;
};

TASK t4 {
  PRIORITY = 6;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_newisgreater;
};

TASK t5 {
  PRIORITY = 5;
```

```
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_newislessorequal;
};

TASK t6 {
  PRIORITY = 4;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_newisless;
};

TASK t7 {
  PRIORITY = 3;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_newisgreaterorequal;
};

TASK t8 {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_oneeveryn;
};

MESSAGE sm {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
    CDATATYPE = "uint8";
  };
  NOTIFICATION = NONE;
};

MESSAGE rm_always {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 0;
    FILTER = ALWAYS;
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t2;
  };
};

MESSAGE rm_never {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
```

```
    SENDINGMESSAGE = sm;
    INITIALVALUE = 0;
    FILTER = NEVER;
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t3;
  };
};

MESSAGE rm_newisgreater {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 2;
    FILTER = NEWISGREATER;
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t4;
  };
};

MESSAGE rm_newislessorequal {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 2;
    FILTER = NEWISLESSOREQUAL;
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t5;
  };
};

MESSAGE rm_newisless {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 2;
    FILTER = NEWISLESS;
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t6;
  };
};

MESSAGE rm_newisgreaterorequal {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 2;
    FILTER = NEWISGREATEROREQUAL;
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t7;
```

```
  };
};
MESSAGE rm_oneeveryn {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 0;
    FILTER = ONEEVERYN {
      PERIOD = 2;
      OFFSET = 3;
    };
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t8;
  };
};
```

Running task	Called OS service	Return Status	Test case
t1	SendMessage(sm, 1)	E_OK	
t1	*Force scheduling*		
t2	ReceiveMessage(rm_always, & DataRef)	E_OK, DataRef=1	8
t2	TerminateTask()		
t5	ReceiveMessage(rm_newislessorequal, & DataRef)	E_OK, DataRef=1	19
t5	TerminateTask()		
t6	ReceiveMessage(rm_newisless, & DataRef)	E_OK, DataRef=1	20
t6	TerminateTask()		
t1	SendMessage(sm, 2)	E_OK	
t1	*Force scheduling*		
t2	ReceiveMessage(rm_always, & DataRef)	E_OK, DataRef=2	8
t2	TerminateTask()		
t7	ReceiveMessage(rm_newisgreaterorequal, & DataRef)	E_OK, DataRef=2	21
t7	TerminateTask()		
t1	SendMessage(sm, 3)	E_OK	
t1	*Force scheduling*		
t2	ReceiveMessage(rm_always, & DataRef)	E_OK, DataRef=3	8
t2	TerminateTask()		
t4	ReceiveMessage(rm_newisgreater, & DataRef)	E_OK, DataRef=3	18
t4	TerminateTask()		
t7	ReceiveMessage(rm_newisgreaterorequal, & DataRef)	E_OK, DataRef=3	21
t7	TerminateTask()		
t1	SendMessage(sm, 2)	E_OK	
t1	*Force scheduling*		

Running task	Called OS service	Return Status	Test case
t2	ReceiveMessage(rm_always, & DataRef)	E_OK, DataRef=2	8
t2	TerminateTask()		
t1	SendMessage(sm, 1)	E_OK	
t1	*Force scheduling*		
t2	ReceiveMessage(rm_always, & DataRef)	E_OK, DataRef=1	8
t2	TerminateTask()		
t5	ReceiveMessage(rm_newislessorequal, & DataRef)	E_OK, DataRef=1	19
t5	TerminateTask()		
t8	ReceiveMessage(rm_oneeveryn, & DataRef)	E_OK, DataRef=1	22
t8	TerminateTask()		
t1	SendMessage(sm, 0)	E_OK	
t1	*Force scheduling*		
t2	ReceiveMessage(rm_always, & DataRef)	E_OK, DataRef=0	8
t2	TerminateTask()		
t5	ReceiveMessage(rm_newislessorequal, & DataRef)	E_OK, DataRef=0	19
t5	TerminateTask()		
t6	ReceiveMessage(rm_newisless, & DataRef)	E_OK, DataRef=0	20
t6	TerminateTask()		
t1	SendMessage(sm, 1)	E_OK	
t1	*Force scheduling*		
t2	ReceiveMessage(rm_always, & DataRef)	E_OK, DataRef=1	8
t2	TerminateTask()		
t8	ReceiveMessage(rm_oneeveryn, & DataRef)	E_OK, DataRef=1	22
t8	TerminateTask()		
t1	SendMessage(sm, 2)	E_OK	
t1	*Force scheduling*		
t2	ReceiveMessage(rm_always, & DataRef)	E_OK, DataRef=2	8
t2	TerminateTask()		
t1	SendMessage(sm, 5)	E_OK	
t1	*Force scheduling*		
t2	ReceiveMessage(rm_always, & DataRef)	E_OK, DataRef=5	8
t2	TerminateTask()		
t4	ReceiveMessage(rm_newisgreater, & DataRef)	E_OK, DataRef=5	18
t4	TerminateTask()		
t7	ReceiveMessage(rm_newisgreaterorequal, & DataRef)	E_OK, DataRef=5	21
t7	TerminateTask()		
t8	ReceiveMessage(rm_oneeveryn, & DataRef)	E_OK, DataRef=5	22
t8	TerminateTask()		
t1	TerminateTask()		

Running task	Called OS service	Return Status	Test case

Test Sequence 6 :

TEST CASES: 10, 11, 12, 13, 14, 15, 16, 17
RETURN COM STATUS: EXTENDED
SCHEDULING POLICY: NON–, MIXED–, FULL–PREEMPTIVE
HOOKS: ...
MESSAGE TYPE: Unqueued

```
TASK t1 {
  PRIORITY = 1;
  AUTOSTART = TRUE { APPMODE = std; };
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = sm;
};

TASK t2 {
  PRIORITY = 9;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_maskednewequalsx;
};

TASK t3 {
  PRIORITY = 8;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_maskednewdiffersx;
};

TASK t4 {
  PRIORITY = 7;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_newisequal;
};

TASK t5 {
  PRIORITY = 6;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_newisdifferent;
};
```

```
TASK t6 {
  PRIORITY = 5;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_maskednewequalsmaskedold;
};

TASK t7 {
  PRIORITY = 4;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_maskednewdiffersmaskedold;
};

TASK t8 {
  PRIORITY = 3;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_newiswithin;
};

TASK t9 {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON,FULL;
  MESSAGE = rm_newisoutside;
};

MESSAGE sm {
  MESSAGEPROPERTY = SEND_STATIC_INTERNAL {
    CDATATYPE = "uint8";
  };
};

MESSAGE rm_maskednewequalsx {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 0;
    FILTER = MASKEDNEWEQUALSX {
      MASK = 4;
      X = 4;
    };
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t2;
  };
```

```
};

MESSAGE rm_maskednewdiffersx {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 0;
    FILTER = MASKEDNEWDIFFERSX {
      MASK = 4;
      X = 4;
    };
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t3;
  };
};

MESSAGE rm_newisequal {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 2;
    FILTER = NEWISEQUAL;
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t4;
  };
};

MESSAGE rm_newisdifferent {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 2;
    FILTER = NEWISDIFFERENT;
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t5;
  };
};

MESSAGE rm_maskednewequalsmaskedold {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 0;
    FILTER = MASKEDNEWEQUALSMASKEDOLD {
      MASK = 4;
    };
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t6;
  };
};
```

```
MESSAGE rm_maskednewdiffersmaskedold {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 0;
    FILTER = MASKEDNEWDIFFERSMASKEDOLD {
      MASK = 4;
    };
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t7;
  };
};

MESSAGE rm_newiswithin {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 0;
    FILTER = NEWISWITHIN {
      MIN = 5;
      MAX = 10;
    };
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t8;
  };
};

MESSAGE rm_newisoutside {
  MESSAGEPROPERTY = RECEIVE_UNQUEUED_INTERNAL {
    SENDINGMESSAGE = sm;
    INITIALVALUE = 0;
    FILTER = NEWISOUTSIDE {
      MIN = 5;
      MAX = 10;
    };
  };
  NOTIFICATION = ACTIVATETASK {
    TASK = t9;
  };
};
```

Running task	Called OS service	Return Status	Test case
t1	SendMessage(sm, 3)	E_OK	
t1	*Force scheduling*		
t3	ReceiveMessage(rm_maskednewdiffersx, & DataRef)	E_OK, DataRef=3	11
t3	TerminateTask()		

Running task	Called OS service	Return Status	Test case
t5	ReceiveMessage(rm_newisdifferent, & DataRef)	E_OK, DataRef=3	13
t5	TerminateTask()		
t6	ReceiveMessage(rm_maskednewequalsmaskedold, & DataRef)	E_OK, DataRef=3	14
t6	TerminateTask()		
t9	ReceiveMessage(rm_newisoutside, & DataRef)	E_OK, DataRef=3	17
t9	TerminateTask()		
t1	SendMessage(sm, 12)	E_OK	
t1	*Force scheduling*		
t2	ReceiveMessage(rm_maskednewequalsx, & DataRef)	E_OK, DataRef=12	10
t2	TerminateTask()		
t5	ReceiveMessage(rm_newisdifferent, & DataRef)	E_OK, DataRef=12	13
t5	TerminateTask()		
t7	ReceiveMessage(rm_maskednewdiffersmaskedold, & DataRef)	E_OK, DataRef=12	15
t7	TerminateTask()		
t9	ReceiveMessage(rm_newisoutside, & DataRef)	E_OK, DataRef=12	17
t9	TerminateTask()		
t1	SendMessage(sm, 7)	E_OK	
t1	*Force scheduling*		
t2	ReceiveMessage(rm_maskednewequalsx, & DataRef)	E_OK, DataRef=7	10
t2	TerminateTask()		
t5	ReceiveMessage(rm_newisdifferent, & DataRef)	E_OK, DataRef=7	13
t5	TerminateTask()		
t8	ReceiveMessage(rm_newiswithin, & DataRef)	E_OK, DataRef=7	16
t8	TerminateTask()		
t1	SendMessage(sm, 7)	E_OK	
t1	*Force scheduling*		
t2	ReceiveMessage(rm_maskednewequalsx, & DataRef)	E_OK, DataRef=7	10
t2	TerminateTask()		
t8	ReceiveMessage(rm_newiswithin, & DataRef)	E_OK, DataRef=7	16
t8	TerminateTask()		
t1	SendMessage(sm, 2)	E_OK	
t1	*Force scheduling*		
t3	ReceiveMessage(rm_maskednewdiffersx, & DataRef)	E_OK, DataRef=2	11
t3	TerminateTask()		
t4	ReceiveMessage(rm_newisequal, & DataRef)	E_OK, DataRef=2	12
t4	TerminateTask()		
t5	ReceiveMessage(rm_newisdifferent, & DataRef)	E_OK, DataRef=2	13
t5	TerminateTask()		

Running task	Called OS service	Return Status	Test case
t6	ReceiveMessage(rm_maskednewequalsmaskedold, & DataRef)	E_OK, DataRef=2	14
t6	TerminateTask()		
t7	ReceiveMessage(rm_maskednewdiffersmaskedold, & DataRef)	E_OK, DataRef=2	15
t7	TerminateTask()		
t9	ReceiveMessage(rm_newisoutside, & DataRef)	E_OK, DataRef=2	17
t9	TerminateTask()		
t1	TerminateTask()		

2.8 AUTOSAR - Core OS

Test cases 3, 5 and 7 are GOIL tests.

Test Sequence 1 :

```
TEST CASES:              1, 2, 4, 6
RETURN STATUS:          EXTENDED
SCHEDULING POLICY:     FULL_PREEMPTIVE
HOOKS:                  ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter {
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};

COUNTER Software_Counter_By_Alarm {
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
```

```
ALARM Alarm_ActivateTask {
  COUNTER = Software_Counter;
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = FALSE;
};

ALARM Alarm_SetEvent_suspendedtask {
  COUNTER = Software_Counter;
  ACTION = SETEVENT {
    TASK = t2;
    EVENT = Event1;
  };
  AUTOSTART = FALSE;
};
ALARM Alarm_IncrementCounter {
  COUNTER = Software_Counter;
  ACTION = INCREMENTCOUNTER {
    COUNTER = Software_Counter_By_Alarm;
  };
  AUTOSTART = FALSE;
};

EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	GetCounterValue(Software_Counter, &Tick)	E_OK, Tick=0	
t1	SetRelAlarm(Alarm_ActivateTask, 2, 0)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	2
ErrorHook	OSErrorGetServiceId()	OSServiceId_ActivateTask	
t1	SetRelAlarm(Alarm_SetEvent_suspendedtask, 2, 0)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	4
ErrorHook	OSErrorGetServiceId()	OSServiceId_SetEvent	
t1	GetCounterValue(Software_Counter_By_Alarm, &Tick)	E_OK, Tick=0	
t1	SetRelAlarm(Alarm_IncrementCounter, 2, 0)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	6
t1	GetCounterValue(Software_Counter_By_Alarm, &Tick)	E_OK, Tick=1	

Running task	Called OS service	Return Status	Test case
t1	SetRelAlarm(Alarm_ActivateTask, 0, 0)	E_OS_VALUE	1
ErrorHook	OSErrorGetServiceId()	OSServiceId_SetRelAlarm	
ErrorHook	OSError_SetRelAlarm_AlarmID()	Alarm_ActivateTask	
ErrorHook	OSError_SetRelAlarm_increment()	0	
ErrorHook	OSError_SetRelAlarm_cycle()	0	
t1	TerminateTask()		

Test Sequence 2 :

```
TEST CASES:              8, 9, 10
RETURN STATUS:          EXTENDED
SCHEDULING POLICY:      FULL_PREEMPTIVE
HOOKS:                   ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
ISR isr1 {
  CATEGORY = 2;
  SOURCE = SIGTERM;
  PRIORITY = 1;
  STACKSIZE = 32768;
};
COUNTER Software_Counter {
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
COUNTER Software_Counter_By_Alarm {
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
ALARM Alarm_ActivateTask {
  COUNTER = Software_Counter;
  ACTION = ACTIVATETASK {
    TASK = t1;
```

```
  };
  AUTOSTART = FALSE;
};
ALARM Alarm_SetEvent_suspendedtask {
  COUNTER = Software_Counter;
  ACTION = SETEVENT {
    TASK = t2;
  EVENT = Event1;
  };
  AUTOSTART = FALSE;
};
ALARM Alarm_IncrementCounter {
  COUNTER = Software_Counter;
  ACTION = INCREMENTCOUNTER {
    COUNTER = Software_Counter_By_Alarm;
  };
  AUTOSTART = FALSE;
};
EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	*trigger interrupt isr1*		
isr1	GetCounterValue(Software_Counter, &Tick)	E_OK, Tick=0	
isr1	SetRelAlarm(Alarm_ActivateTask, 2, 0)	E_OK	
isr1	IncrementCounter(Software_Counter)	E_OK	
isr1	IncrementCounter(Software_Counter)	E_OK	9
ErrorHook	OSErrorGetServiceId()	OSServiceId_ActivateTask	
isr1	SetRelAlarm(Alarm_SetEvent_suspendedtask, 2, 0)	E_OK	
isr1	IncrementCounter(Software_Counter)	E_OK	
isr1	IncrementCounter(Software_Counter)	E_OK	10
ErrorHook	OSErrorGetServiceId()	OSServiceId_SetEvent	
isr1	GetCounterValue(Software_Counter_By_Alarm, &Tick)	E_OK, Tick=0	
isr1	SetRelAlarm(Alarm_ActivateTask, 0, 0)	E_OS_VALUE	8
ErrorHook	OSErrorGetServiceId()	OSServiceId_SetRelAlarm	
ErrorHook	OSError_SetRelAlarm_AlarmID()	Alarm_ActivateTask	
ErrorHook	OSError_SetRelAlarm_increment()	0	
ErrorHook	OSError_SetRelAlarm_cycle()	0	
t1	TerminateTask()		

Test Sequence 3 :

```
TEST CASES:              11, 12
RETURN STATUS:           EXTENDED
SCHEDULING POLICY:       FULL_PREEMPTIVE
```

```
TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
ISR isr1 {
  CATEGORY = 2;
  SOURCE = SIGTERM;
  PRIORITY = 1;
  STACKSIZE = 32768;
};
```

Running task	Called OS service	Return Status	Test case
t1	*trigger interrupt isr1*		
isr1	GetISRID()	isr1	12
isr1	ActivateTask(t1)	E_OS_LIMIT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ActivateTask	
ErrorHook	GetISRID()	isr1	12
t1	GetISRID()	INVALID_ISR	11
t1	ActivateTask(t1)	E_OS_LIMIT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ActivateTask	
ErrorHook	GetISRID()	INVALID_ISR	11
t1	TerminateTask()		

2.9 AUTOSAR - Software Counter

Test Sequence 1 :

```
TEST CASES:            1, 2, 9, 15
RETURN STATUS:         STANDARD, EXTENDED
SCHEDULING POLICY:     FULL-PREEMPTIVE
HOOKS:                 ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter {
  MAXALLOWEDVALUE = 3;
  TICKSPERBASE = 2;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
ALARM Alarm1 {
  COUNTER = Software_Counter;
```

```
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = FALSE;
};
```

Running task	Called OS service	Return Status	Test case
t1	GetCounterValue(Software_Counter, &Tick)	E_OK, Tick=0	9
t1	SetRelAlarm(Alarm1, 2, 0)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	1
t1	IncrementCounter(Software_Counter)	E_OK	
t1	GetCounterValue(Software_Counter, &Tick)	E_OK, Tick=1	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	2
ErrorHook	OSErrorGetServiceId()	OSServiceId_ActivateTask	
ErrorHook	OSError_ActivateTask_TaskID()	t1	
t1	GetCounterValue(Software_Counter, &Tick)	E_OK, Tick=2	
t1	GetElapsedCounterValue(Software_Counter, &Tick_value = 0, &Tick_elasped_value)	E_OK, Tick_value=2, Tick_elapsed_value=2	15
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	GetCounterValue(Software_Counter, &Tick)	E_OK, Tick=3	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	GetCounterValue(Software_Counter, &Tick)	E_OK, Tick=0	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	GetCounterValue(Software_Counter, &Tick)	E_OK, Tick=1	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ActivateTask	
ErrorHook	OSError_ActivateTask_TaskID()	t1	
t1	CancelAlarm(Alarm1)	E_OK	
t1	GetCounterValue(Software_Counter, &Tick)	E_OK, Tick=2	
t1	TerminateTask()		

Test Sequence 2 :

```
TEST CASES:              10, 17
RETURN STATUS:           STANDARD, EXTENDED
SCHEDULING POLICY:       FULL_PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
```

```
  EVENT = Event1;
};

COUNTER Hardware_Counter{
  MAXALLOWEDVALUE = 3;
  TICKSPERBASE = 2;
  MINCYCLE = 1;
  TYPE = HARDWARE {};
};

ALARM Alarm1 {
  COUNTER = Hardware_Counter;
  ACTION = SETEVENT {
    TASK = t1;
    EVENT = Event1;
  };
  AUTOSTART = FALSE;
};

EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	GetCounterValue(Hardware_Counter, &Tick1)	E_OK	10
t1	*Wait 3 Counter tick*		
t1	GetCounterValue(Hardware_Counter, &Tick2)	E_OK, Tick2=Tick1+3	
t1	SetRelAlarm(Alarm1, 2, 0)	E_OK	
t1	*Wait alarm expires*		
t1	GetElapsedCounterValue(Hardware_Counter, &Tick2, &Tick_elasped_value)	E_OK, Tick2=Tick1+3+2, Tick_elapsed_value=2	17
t1	GetEvent(t1, &EventMask)	E_OK, EventMask=Event1	
t1	TerminateTask()		

Test Sequence 3 :

```
TEST CASES:              3, 4, 11, 16, 18, 19
RETURN STATUS:           EXTENDED
SCHEDULING POLICY:       FULL_PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

COUNTER Hardware_Counter{
  MAXALLOWEDVALUE = 3;
  TICKSPERBASE = 2;
```

```
  MINCYCLE = 1;
  TYPE = HARDWARE {};
};

COUNTER Software_Counter{
  MAXALLOWEDVALUE = 3;
  TICKSPERBASE = 2;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
```

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Hardware_Counter)	E_OS_ID	3
ErrorHook	OSErrorGetServiceId()	OSServiceId_ IncrementCounter	
ErrorHook	OSServiceId_ IncrementCounter_ CounterID()	Hardware_Counter	
t1	IncrementCounter(INVALID_COUNTER)	E_OS_ID	4
ErrorHook	OSErrorGetServiceId()	OSServiceId_ IncrementCounter	
ErrorHook	OSServiceId_ IncrementCounter_ CounterID()	INVALID_COUNTER	
t1	GetCounterValue(INVALID_COUNTER, &Tick)	E_OS_ID	11
ErrorHook	OSErrorGetServiceId()	OSServiceId_ GetCounterValue	
ErrorHook	OSServiceId_ GetCounterValue_ value()	OSMAXALLOWEDVALUE + 1	
ErrorHook	OSServiceId_ GetCounterValue_ CounterID()	INVALID_COUNTER	
t1	GetElapsedCounterValue(Software_Counter, OSMAXALLOWEDVALUE + 1, &Tick_elasped_value)	E_OS_VALUE	16
ErrorHook	OSErrorGetServiceId()	OSServiceId_ GetElapsedCounterValue	
ErrorHook	OSServiceId_ GetElapsedCounterValue_ CounterID()	Software_Counter	
ErrorHook	OSServiceId_ GetElapsedCounterValue_ value()	OSMAXALLOWEDVALUE + 1	
ErrorHook	OSServiceId_ GetElapsedCounterValue_ previous_value()	OSMAXALLOWEDVALUE + 1	
t1	GetElapsedCounterValue(Hardware_Counter, OSMAXALLOWEDVALUE + 1, &Tick_elasped_value)	E_OS_VALUE	18
ErrorHook	OSErrorGetServiceId()	OSServiceId_ GetElapsedCounterValue	
ErrorHook	OSServiceId_ GetElapsedCounterValue_ CounterID()	Hardware_Counter	

Running task	Called OS service	Return Status	Test case
ErrorHook	OSServiceId_ GetElapsedCounterValue_ previous_value()	OSMAXALLOWEDVALUE + 1	
t1	GetElapsedCounterValue(INVALID_COUNTER, &Tick_value, &Tick_elasped_value)	E_OS_ID	19
ErrorHook	OSErrorGetServiceId()	OSServiceId_ GetElapsed-CounterValue	
ErrorHook	OSServiceId_ GetElapsedCounterValue_ CounterID()	INVALID_COUNTER	
t1	TerminateTask()		

Test Sequence 4 :

```
TEST CASES:             5, 6, 12, 20
RETURN STATUS:          STANDARD, EXTENDED
SCHEDULING POLICY:      FULL-PREEMPTIVE
HOOKS:                  ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};

COUNTER Software_Counter {
  MAXALLOWEDVALUE = 3;
  TICKSPERBASE = 2;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};

ALARM Alarm1 {
  COUNTER = Software_Counter;
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = FALSE;
};
```

Running task	Called OS service	Return Status	Test case
t1	*trigger interrupt isr1*		
isr1	GetCounterValue(Software_Counter, &Tick)	E_OK, Tick=0	12
isr1	SetRelAlarm(Alarm1, 2, 0)	E_OK	
isr1	IncrementCounter(Software_Counter)	E_OK	5
isr1	IncrementCounter(Software_Counter)	E_OK	
isr1	GetCounterValue(Software_Counter, &Tick)	E_OK, Tick=1	
isr1	IncrementCounter(Software_Counter)	E_OK	
isr1	IncrementCounter(Software_Counter)	E_OK	6
ErrorHook	OSErrorGetServiceId()	OSServiceId_ActivateTask	
ErrorHook	OSError_ActivateTask_TaskID()	t1	
isr1	GetCounterValue(Software_Counter, &Tick)	E_OK, Tick=2	
isr1	GetElapsedCounterValue(Software_Counter, &Tick_value = 0, &Tick_elasped_value)	E_OK, Tick_value=2, Tick_elapsed_value=2	20
t1	TerminateTask()		

Test Sequence 5 :

```
TEST CASES:            7, 8, 13, 14, 21, 22, 23, 24
RETURN STATUS:         EXTENDED
SCHEDULING POLICY:     FULL_PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};

ISR isr1 {
  CATEGORY = 2;
  STACKSIZE = 32768;
  PRIORITY = 1;
  SOURCE = SIGTERM;
};

COUNTER Hardware_Counter {
  MAXALLOWEDVALUE = 3;
  TICKSPERBASE = 2;
  MINCYCLE = 1;
  TYPE = HARDWARE {};
};

COUNTER Software_Counter {
  MAXALLOWEDVALUE = 3;
  TICKSPERBASE = 2;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
```

```
};

ALARM Alarm1 {
  COUNTER = Hardware_Counter;
  ACTION = SETEVENT {
    TASK = t1;
    EVENT = Event1;
  };
  AUTOSTART = FALSE;
};

EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	*trigger interrupt isr1*		
isr1	GetCounterValue(Hardware_Counter, &Tick1)	E_OK	13
isr1	*Wait 3 Counter tick*		
isr1	GetCounterValue(Hardware_Counter, &Tick2)	E_OK, Tick2=Tick1+3	
isr1	SetRelAlarm(Alarm1, 2, 0)	E_OK	
isr1	*Wait alarm expires*		
isr1	GetElapsedCounterValue(Hardware_Counter, &Tick2, &Tick_elasped_value)	E_OK, Tick2=Tick1+3+2, Tick_elapsed_value=2	22
isr1	GetEvent(t1, &EventMask)	E_OK, EventMask=Event1	
isr1	IncrementCounter(Hardware_Counter)	E_OS_ID	7
isr1	IncrementCounter(INVALID_COUNTER)	E_OS_ID	8
isr1	GetCounterValue(INVALID_COUNTER, &Tick)	E_OS_ID	14
isr1	GetElapsedCounterValue(Software_Counter, OSMAXALLOWEDVALUE + 1, &Tick_elasped_value)	E_OS_VALUE	21
isr1	GetElapsedCounterValue(Hardware_Counter, OSMAXALLOWEDVALUE + 1, &Tick_elasped_value)	E_OS_VALUE	23
isr1	GetElapsedCounterValue(INVALID_COUNTER, &Tick_value, &Tick_elasped_value)	E_OS_ID	24
t1	TerminateTask()		

2.10 AUTOSAR - Schedule Table

Test Sequence 1 :

```
TEST CASES:              1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 20, 23, 24
RETURN STATUS:       EXTENDED
SCHEDULING POLICY:   FULL–PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
```

```
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

COUNTER Software_Counter {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};

SCHEDULETABLE sched1 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 10;
  EXPIRY_POINT t2_acti {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
  };
};
```

Running task	Called OS service	Return Status	Test case
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	20, 23
t1	StartScheduleTableRel(INVALID_SCHEDULETABLE , 1)	E_OS_ID	2
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableRel	
ErrorHook	OSServiceId_StartScheduleTableRel_ ScheduleTableID()	INVALID_SCHEDULETABLE	
ErrorHook	OSServiceId_StartScheduleTableRel_ offset()	1	
t1	StartScheduleTableRel(sched1, 0)	E_OS_VALUE	3
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableRel	

Running task	Called OS service	Return Status	Test case
ErrorHook	OSServiceId_StartScheduleTableRel_ ScheduleTableID()	sched1	
ErrorHook	OSServiceId_StartScheduleTableRel_ offset()	0	
t1	StartScheduleTableRel(sched1, OSMAXALLOWEDVALUE - InitOffset)	E_OS_VALUE	4
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableRel	
ErrorHook	OSServiceId_StartScheduleTableRel_ ScheduleTableID()	sched1	
ErrorHook	OSServiceId_StartScheduleTableRel_ offset()	OSMAXALLOWEDVALUE - InitOffset	
t1	StartScheduleTableRel(sched1, 1)	E_OK	1
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	24
t1	StartScheduleTableRel(sched1, 1)	E_OS_STATE	5
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableRel	
ErrorHook	OSServiceId_StartScheduleTableRel_ ScheduleTableID()	sched1	
ErrorHook	OSServiceId_StartScheduleTableRel_ offset()	1	
t1	StopScheduleTable(INVALID_SCHEDULETABLE)	E_OS_ID	11
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StopScheduleTable	
ErrorHook	OSServiceId_ StopScheduleTable_ ScheduleTableID()	INVALID_SCHEDULETABLE	
t1	StopScheduleTable(sched1)	E_OK	10
t1	GetScheduleTableStatus(sched1 , &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	StopScheduleTable(sched1)	E_OS_NOFUNC	12
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StopScheduleTable	
ErrorHook	OSServiceId_ StopScheduleTable_ ScheduleTableID()	sched1	
t1	StartScheduleTableAbs(INVALID_SCHEDULETABLE , 1)	E_OS_ID	7
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableAbs	

Running task	Called OS service	Return Status	Test case
ErrorHook	OSServiceId_StartScheduleTableAbs_ ScheduleTableID()	INVALID_SCHEDULETABLE	
ErrorHook	OSServiceId_StartScheduleTableAbs_ value()	1	
t1	StartScheduleTableAbs(sched1, OSMAXALLOWEDVALUE)	E_OS_VALUE	8
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableAbs	
ErrorHook	OSServiceId_StartScheduleTableAbs_ ScheduleTableID()	sched1	
ErrorHook	OSServiceId_StartScheduleTableAbs_ value()	OSMAXALLOWEDVALUE	
t1	StartScheduleTableAbs(sched1, 1)	E_OK	6
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	StartScheduleTableAbs(sched1, 1)	E_OS_STATE	9
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableAbs	
ErrorHook	OSServiceId_StartScheduleTableAbs_ ScheduleTableID()	sched1	
ErrorHook	OSServiceId_StartScheduleTableAbs_ value()	1	
t1	StopScheduleTable(sched1)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	

Test Sequence 2 :

```
TEST CASES:            5, 9, 10, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
    24
RETURN STATUS:       EXTENDED
SCHEDULING POLICY:    FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

COUNTER Software_Counter {
```

```
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};

COUNTER Hardware_Counter{
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = HARDWARE {};
};

SCHEDULETABLE sched1 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 10;
  EXPIRY_POINT t2_acti {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
  };
};

SCHEDULETABLE sched2 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 10;
  EXPIRY_POINT t2_acti {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
  };
};

SCHEDULETABLE sched3 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 3;
  EXPIRY_POINT t2_acti {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
```

```
      TASK = t2;
    };
  };
};

SCHEDULETABLE sched4 {
  COUNTER = Hardware_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 10;
  EXPIRY_POINT t2_acti {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
  };
};
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableRel(sched1, 1)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	24
t1	NextScheduleTable(INVALID_SCHEDULETABLE , sched2)	E_OS_ID	14
t1	NextScheduleTable(sched1 , INVALID_SCHEDULETABLE)	E_OS_ID	15
t1	NextScheduleTable(sched1 , sched4)	E_OS_ID	16
t1	NextScheduleTable(sched2 , sched2)	E_OS_NOFUNC	18
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	23
t1	NextScheduleTable(sched1 , sched2)	E_OK	13
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_NEXT	20, 22
t1	NextScheduleTable(sched2 , sched3)	E_OS_NOFUNC	17
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_NEXT	
t1	GetScheduleTableStatus(sched3, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	

Running task	Called OS service	Return Status	Test case
t1	NextScheduleTable(sched1 , sched2)	E_OS_STATE	19
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_NEXT	
t1	NextScheduleTable(sched1 , sched1)	E_OS_STATE	19
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	NextScheduleTable(sched1 , sched3)	E_OK	13
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	GetScheduleTableStatus(sched3, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_NEXT	
t1	StartScheduleTableRel(sched3, 1)	E_OS_STATE	5
t1	StartScheduleTableAbs(sched3, 1)	E_OS_STATE	9
t1	StopScheduleTable(sched1)	E_OK	10
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	GetScheduleTableStatus(sched3, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	GetScheduleTableStatus(INVALID_SCHEDULETABLE, &STStatusType)	E_OS_ID	21

Test Sequence 3 :

```
TEST CASES:              25, 26, 27, 28, 29, 30, 32
RETURN STATUS:       STANDARD, EXTENDED
SCHEDULING POLICY:    FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
```

```
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
TASK t4 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t5 {
  AUTOSTART = FALSE;
  PRIORITY = 4;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched1 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 3;
  EXPIRY_POINT first_point {
    OFFSET = 0;
    ACTION = SETEVENT {
      TASK = t3;
      EVENT = Event1;
    };
    ACTION = ACTIVATETASK {
      TASK = t5;
    };
  };
  EXPIRY_POINT second_point {
    OFFSET = 1;
    ACTION = SETEVENT {
      TASK = t3;
      EVENT = Event1;
```

```
    };
    ACTION = ACTIVATETASK {
      TASK = t4;
    };
  };
};
SCHEDULETABLE sched2 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 1;
  EXPIRY_POINT many_things {
    OFFSET = 1;
    ACTION = SETEVENT {
      TASK = t3;
      EVENT = Event1;
    };
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
  };
};
EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t3)	E_OK	
t3	WaitEvent(Event1)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	StartScheduleTableRel(sched1, 1)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter)	E_OK	26, 30, 32

Running task	Called OS service	Return Status	Test case
t5	TerminateTask()		
t3	ClearEvent(Event1)	E_OK	
t3	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	27
t4	TerminateTask()		
t3	ClearEvent(Event1)	E_OK	
t3	WaitEvent(Event1)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter)	E_OK	25
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	StartScheduleTableRel(sched2, 1)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	28, 29
t3	TerminateTask()		
t2	WaitEvent(Event1)	E_OK	
t2	TerminateTask()		
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	TerminateTask()		

Test Sequence 4 :

```
TEST CASES:              31, 34
RETURN STATUS:       STANDARD, EXTENDED
SCHEDULING POLICY:    FULL-PREEMPTIVE
HOOKS:                ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
```

```
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter {
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 1;
  EXPIRY_POINT first_one {
    OFFSET = 0;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
  EXPIRY_POINT second_one {
    OFFSET = 1;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
  };
};
EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableRel(sched1, 1)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	31
Errorhook	OSErrorGetServiceId()	OSServiceId_ActivateTask	
t1	IncrementCounter(Software_Counter)	E_OK	34
Errorhook	OSErrorGetServiceId()	OSServiceId_SetEvent	
t1	GetScheduleTableStatus(sched, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	TerminateTask()		

Test Sequence 5 :

TEST CASES: 5, 9, 12, 17, 18, 35, 36
RETURN STATUS: STANDARD, EXTENDED
SCHEDULING POLICY: FULL–PREEMPTIVE
HOOKS: ErrorHook

```
TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
  EVENT = Event2;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched1 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 2;
  EXPIRY_POINT sched1_exp1 {
    OFFSET = 1;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
  };
};
SCHEDULETABLE sched2 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 1;
  EXPIRY_POINT sched2_exp1 {
    OFFSET = 1;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
```

```
    };
  };
};
SCHEDULETABLE sched3 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 2;
  EXPIRY_POINT sched3_exp1 {
    OFFSET = 0;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event2;
    };
  };
  EXPIRY_POINT sched3_exp2 {
    OFFSET = 1;
    ACTION = ACTIVATETASK {
      TASK = t3;
    };
  };
};
EVENT Event1 {
  MASK = AUTO;
};
EVENT Event2{
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableRel(sched1, 1)	E_OK	
t1	NextScheduleTable(sched1 , sched2)	E_OK	
t1	StartScheduleTableRel(sched2, 1)	E_OS_STATE	5
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableRel	
ErrorHook	OSServiceId_StartScheduleTableRel_ ScheduleTableID()	sched_2	
ErrorHook	OSServiceId_StartScheduleTableRel_ offset()	1	
t1	StartScheduleTableAbs(sched2, 1)	E_OS_STATE	9
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableAbs	
ErrorHook	OSServiceId_StartScheduleTableAbs_ ScheduleTableID()	sched_2	

Running task	Called OS service	Return Status	Test case
ErrorHook	OSServiceId_StartScheduleTableAbs_ value()	1	
t1	StartScheduleTableRel(sched1, 1)	E_OS_STATE	5
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableRel	
ErrorHook	OSServiceId_StartScheduleTableRel_ ScheduleTableID()	sched_1	
ErrorHook	OSServiceId_StartScheduleTableRel_ offset()	1	
t1	StartScheduleTableAbs(sched1, 1)	E_OS_STATE	9
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableAbs	
ErrorHook	OSServiceId_StartScheduleTableAbs_ ScheduleTableID()	sched_1	
ErrorHook	OSServiceId_StartScheduleTableAbs_ value()	1	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	StopScheduleTable(sched1)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	StartScheduleTableRel(sched1, 1)	E_OK	36
t1	NextScheduleTable(sched1 , sched2)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
Errorhook	OSErrorGetServiceId()	OSServiceId_ActivateTask	
t1	NextScheduleTable(sched2 , sched3)	E_OS_NOFUNC	17
ErrorHook	OSErrorGetServiceId()	OSServiceId_ NextScheduleTable	
ErrorHook	OSServiceId_ NextScheduleTable_ ScheduleTableID()	sched_2	
ErrorHook	OSServiceId_ NextScheduleTable_ ScheduleTableID()	sched_3	
t1	IncrementCounter(Software_Counter)	E_OK	35
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	

Running task	Called OS service	Return Status	Test case
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	NextScheduleTable(sched2 , sched3)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	35
t2	WaitEvent(Event2)	E_OK	
t2	TerminateTask()		
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	GetScheduleTableStatus(sched3, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	StopScheduleTable(sched2)	E_OS_NOFUNC	12
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StopScheduleTable	
ErrorHook	OSServiceId_ StopScheduleTable_ ScheduleTableID()	sched_2	
t1	NextScheduleTable(sched2 , sched2)	E_OS_NOFUNC	18
ErrorHook	OSErrorGetServiceId()	OSServiceId_ NextScheduleTable	
ErrorHook	OSServiceId_ NextScheduleTable_ ScheduleTableID()	sched_2	
ErrorHook	OSServiceId_ NextScheduleTable_ ScheduleTableID()	sched_2	
t1	IncrementCounter(Software_Counter)	E_OK	
t3	TerminateTask()		
t1	GetScheduleTableStatus(sched3, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	GetScheduleTableStatus(sched3, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	TerminateTask()		

Test Sequence 6 :

```
TEST CASES:          71, 72, 73, 74, 75, 76, 77, 78, 79, 80
RETURN STATUS:       STANDARD, EXTENDED
SCHEDULING POLICY:   FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
```

```
  SCHEDULE = FULL;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t4 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t5 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t6 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t7 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

TASK t8 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
```

```
};
TASK t9 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter {
  MAXALLOWEDVALUE = 6;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};

SCHEDULETABLE sched_abs_more_1 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 6;
  EXPIRY_POINT one {
    OFFSET = 0;
    ACTION = ACTIVATETASK {
        TASK = t2;
    };
  };
};

SCHEDULETABLE sched_abs_more_2 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 7;
  EXPIRY_POINT two {
    OFFSET = 5;
    ACTION = ACTIVATETASK {
      TASK = t3;
    };
  };
};

SCHEDULETABLE sched_abs_more_3 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 7;
  EXPIRY_POINT three {
    OFFSET = 6;
```

```
    ACTION = ACTIVATETASK {
      TASK = t4;
    };
  };
};

SCHEDULETABLE sched_abs_less_1 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 6;
  EXPIRY_POINT four {
    OFFSET = 0;
    ACTION = ACTIVATETASK {
      TASK = t5;
    };
  };
};

SCHEDULETABLE sched_abs_less_2 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 7;
  EXPIRY_POINT five {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t6;
    };
  };
};

SCHEDULETABLE sched_abs_less_3 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 7;
  EXPIRY_POINT six {
    OFFSET = 5;
    ACTION = ACTIVATETASK {
      TASK = t7;
    };
  };
};
SCHEDULETABLE sched_abs_less_4 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
```

```
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
    PERIODIC = FALSE;
    LENGTH = 7;
    EXPIRY_POINT six {
      OFFSET = 5;
      ACTION = ACTIVATETASK {
        TASK = t8;
      };
    };
  };
  SCHEDULETABLE sched_rel {
    COUNTER = Software_Counter;
    AUTOSTART = NONE;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
    PERIODIC = FALSE;
    LENGTH = 7;
    EXPIRY_POINT seven {
      OFFSET = 1;
      ACTION = ACTIVATETASK {
        TASK = t9;
      };
    };
  };
```

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	StartScheduleTableAbs(sched_abs_more_1, 5)	E_OK	77
t1	StartScheduleTableAbs(sched_abs_more_2, 5)	E_OK	76
t1	StartScheduleTableAbs(sched_abs_more_3, 6)	E_OK	75
t1	StartScheduleTableAbs(sched_abs_less_1, 1)	E_OK	72
t1	StartScheduleTableAbs(sched_abs_less_2, 1)	E_OK	73
t1	StartScheduleTableAbs(sched_abs_less_3, 1)	E_OK	74
t1	StartScheduleTableRel(sched_rel,2)	E_OK	71
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	80
t2	TerminateTask()		
t1	IncrementCounter(Software_Counter)	E_OK	79
t9	TerminateTask()		
t1	IncrementCounter(Software_Counter)	E_OK	78
t1	IncrementCounter(Software_Counter)	E_OK	
t5	TerminateTask()		
t1	IncrementCounter(Software_Counter)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Software_Counter)	E_OK	
t3	TerminateTask()		
t6	TerminateTask()		
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t4	TerminateTask()		
t1	IncrementCounter(Software_Counter)	E_OK	
t7	TerminateTask()		
t1	IncrementCounter(Software_Counter)	E_OK	
t8	TerminateTask()		
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	TerminateTask()		

Test Sequence 7 :

```
TEST CASES:            37, 38, 39, 40, 41
RETURN STATUS:       STANDARD, EXTENDED
SCHEDULING POLICY:    FULL-PREEMPTIVE
HOOK:       ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter{
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched1 {
  COUNTER = Software_Counter;
  AUTOSTART = RELATIVE { OFFSET = 2; APPMODE = std; };
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = TRUE;
  LENGTH = 3;
  EXPIRY_POINT s1 {
    OFFSET = 1;
```

```
      ACTION = ACTIVATETASK {
        TASK = t2;
      };
    };
  };
  SCHEDULETABLE sched2 {
    COUNTER = Software_Counter;
    AUTOSTART = ABSOLUTE { START = 7; APPMODE = std; };
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
    PERIODIC = TRUE;
    LENGTH = 1;
    EXPIRY_POINT s2 {
      OFFSET = 0;
      ACTION = SETEVENT {
        TASK = t2;
        EVENT = Event1;
      };
    };
  };
  EVENT Event1 {
    MASK = AUTO;
  };
```

Running task	Called OS service	Return Status	Test case
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	
t1	IncrementCounter(Software_Counter)	E_OK	37
Errorhook	OSErrorGetServiceId()	OSServiceId_ActivateTask	
t1	IncrementCounter(Software_Counter)	E_OK	38
t2	TerminateTask()		
t1	IncrementCounter(Software_Counter)	E_OK	
Errorhook	OSErrorGetServiceId()	OSServiceId_SetEvent	38
t1	IncrementCounter(Software_Counter)	E_OK	
t2	WaitEvent(Event1)	E_OK	

Running task	Called OS service	Return Status	Test case
t2	TerminateTask()		
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	StopScheduleTable(sched1)	E_OK	
t1	StopScheduleTable(sched2)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	TerminateTask()		

2.11 AUTOSAR - Schedule Table Synchronisation

Test Sequence 1 :

```
TEST CASES:              17, 21, 25
RETURN STATUS:       STANDARD, EXTENDED
SCHEDULING POLICY:    FULL-PREEMPTIVE
HOOKS:      ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 2;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
COUNTER Software_Counter2 {
```

```
  MAXALLOWEDVALUE = 5;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched1 {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    SYNC_STRATEGY = IMPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 3;
  EXPIRY_POINT sched1_offset1 {
    OFFSET = 1;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
  };
};
SCHEDULETABLE sched2 {
  COUNTER = Software_Counter2;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    SYNC_STRATEGY = IMPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 6;
  EXPIRY_POINT sched2_offset0 {
    OFFSET = 0;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
  };
  EXPIRY_POINT sched2_offset5 {
    OFFSET = 5;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
  };
};
EVENT Event1{
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableAbs(sched1, 1)	E_OK	21
t1	StartScheduleTableAbs(sched2, 1)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	17
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter2)	E_OK	
ErrorHook	OSErrorGetServiceId()	OSServiceId_SetEvent	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter2)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter2)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter2)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ActivateTask	
t1	IncrementCounter(Software_Counter2)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter2)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter2)	E_OK	
ErrorHook	OSErrorGetServiceId()	OSServiceId_SetEvent	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	

Running task	Called OS service	Return Status	Test case
t1	StopScheduleTable(sched1)	E_OK	
t1	StopScheduleTable(sched2)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	TerminateTask()	E_OK	

Test Sequence 2 :

```
TEST CASES:            2, 3, 6, 7, 8, 9, 10, 13, 14, 18
RETURN STATUS:         EXTENDED
SCHEDULING POLICY:     FULL-PREEMPTIVE
HOOKS:       ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_explicit {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 1;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 5;
  EXPIRY_POINT sched_explicit_offset1 {
    OFFSET = 1;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
SCHEDULETABLE sched_explicit_next {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
```

```
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
      PRECISION = 0;
      SYNC_STRATEGY = EXPLICIT;
    };
    PERIODIC = TRUE;
    LENGTH = 5;
    EXPIRY_POINT sched_explicit_next_offset1{
      OFFSET = 1;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  SCHEDULETABLE sched_explicit_autostart {
    COUNTER = Software_Counter1;
    AUTOSTART = SYNCHRON { APPMODE = std; };
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
      PRECISION = 0;
      SYNC_STRATEGY = EXPLICIT;
    };
    PERIODIC = TRUE;
    LENGTH = 5;
    EXPIRY_POINT sched_explicit_autostart_offset1 {
      OFFSET = 1;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  SCHEDULETABLE sched_implicit {
    COUNTER = Software_Counter1;
    AUTOSTART = NONE;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
      SYNC_STRATEGY = IMPLICIT;
    };
    PERIODIC = TRUE;
    LENGTH = 11;
    EXPIRY_POINT sched_implicit_offset1 {
      OFFSET = 1;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  SCHEDULETABLE sched_nosync {
    COUNTER = Software_Counter1;
    AUTOSTART = NONE;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
    PERIODIC = FALSE;
    LENGTH = 11;
```

```
    EXPIRY_POINT sched_nosync_offset1 {
      OFFSET = 1;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableSynchron(INVALID_ SCHEDULETABLE)	E_OS_ID	2
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableSynchron	
ErrorHook	OSServiceId_StartScheduleTableSynchron_ ScheduleTableID()	INVALID_SCHEDULETABLE	
t1	StartScheduleTableSynchron(sched_implicit)	E_OS_ID	3
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableSynchron	
ErrorHook	OSServiceId_StartScheduleTableSynchron_ ScheduleTableID()	sched_implicit	
t1	StartScheduleTableSynchron(sched_nosync)	E_OS_ID	3
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableSynchron	
ErrorHook	OSServiceId_StartScheduleTableSynchron_ ScheduleTableID()	sched_nosync	
t1	SyncScheduleTable(INVALID_SCHEDULETABLE, 1)	E_OS_ID	6
ErrorHook	OSErrorGetServiceId()	OSServiceId_SyncScheduleTable	
ErrorHook	OSServiceId_SyncScheduleTable_ ScheduleTableID()	INVALID_SCHEDULETABLE	
ErrorHook	OSServiceId_SyncScheduleTable_ value()	1	
t1	SyncScheduleTable(sched_implicit, 1)	E_OS_ID	7
ErrorHook	OSErrorGetServiceId()	OSServiceId_SyncScheduleTable	
ErrorHook	OSServiceId_SyncScheduleTable_ ScheduleTableID()	sched_implicit	
ErrorHook	OSServiceId_SyncScheduleTable_ value()	1	
t1	SyncScheduleTable(sched_nosync, 1)	E_OS_ID	7
ErrorHook	OSErrorGetServiceId()	OSServiceId_SyncScheduleTable	
ErrorHook	OSServiceId_SyncScheduleTable_ ScheduleTableID()	sched_nosync	
ErrorHook	OSServiceId_SyncScheduleTable_ value()	1	
t1	SyncScheduleTable(sched_explicit, 1)	E_OS_STATE	9
ErrorHook	OSErrorGetServiceId()	OSServiceId_SyncScheduleTable	
ErrorHook	OSServiceId_SyncScheduleTable_ ScheduleTableID()	sched_explicit	
ErrorHook	OSServiceId_SyncScheduleTable_ value()	1	

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableSynchron(sched_explicit)	E_OK	
t1	NextScheduleTable(sched_explicit, sched_explicit_next)	E_OK	
t1	SyncScheduleTable(sched_explicit_next, 1)	E_OS_STATE	10
ErrorHook	OSErrorGetServiceId()	OSServiceId_SyncScheduleTable	
ErrorHook	OSServiceId_SyncScheduleTable_ ScheduleTableID()	sched_explicit_next	
ErrorHook	OSServiceId_SyncScheduleTable_ value()	1	
t1	SyncScheduleTable(sched_explicit, 11)	E_OS_VALUE	8
ErrorHook	OSErrorGetServiceId()	OSServiceId_SyncScheduleTable	
ErrorHook	OSServiceId_SyncScheduleTable_ ScheduleTableID()	sched_explicit	
ErrorHook	OSServiceId_SyncScheduleTable_ value()	11	
t1	SetScheduleTableAsync(INVALID_ SCHEDULETABLE)	E_OS_ID	14
ErrorHook	OSErrorGetServiceId()	OSServiceId_SetScheduleTableAsync	
ErrorHook	OSServiceId_SetScheduleTableAsync_ ScheduleTableID()	INVALID_SCHEDULETABLE	
t1	SetScheduleTableAsync(sched_implicit)	E_OS_ID	13
ErrorHook	OSErrorGetServiceId()	OSServiceId_SetScheduleTableAsync	
ErrorHook	OSServiceId_SetScheduleTableAsync_ ScheduleTableID()	sched_implicit	
t1	SetScheduleTableAsync(sched_nosync)	E_OS_ID	13
ErrorHook	OSErrorGetServiceId()	OSServiceId_SetScheduleTableAsync	
ErrorHook	OSServiceId_SetScheduleTableAsync_ ScheduleTableID()	sched_nosync	
t1	StartScheduleTableRel(sched_implicit, 1)	E_OS_ID	18
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableRel	
ErrorHook	OSServiceId_StartScheduleTableRel_ ScheduleTableID()	sched_implicit	
ErrorHook	OSServiceId_StartScheduleTableRel_ offset()	1	
t1	TerminateTask()	E_OK	

Test Sequence 3 :

```
TEST CASES:          1, 4, 5, 15, 17, 20, 23, 26
RETURN STATUS:       STANDARD, EXTENDED
SCHEDULING POLICY:   FULL_PREEMPTIVE
HOOKS:       ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
```

```
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 4;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_explicit {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 0;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 5;
  EXPIRY_POINT offset1 {
    OFFSET = 1;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
  };
  EXPIRY_POINT offset3 {
    OFFSET = 3;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
  };
};
EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableSynchron(sched_explicit)	E_OK	1
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_WAITING	15
t1	StartScheduleTableSynchron(sched_explicit)	E_OS_STATE	4
t1	IncrementCounter(Software_Counter1)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_WAITING	
t1	StartScheduleTableRel(sched_explicit)	E_OS_STATE	20
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableRel	
ErrorHook	OSServiceId_StartScheduleTableRel_ ScheduleTableID()	sched_explicit	
ErrorHook	OSServiceId_StartScheduleTableRel_ offset()	1	
t1	StartScheduleTableAbs(sched_explicit)	E_OS_STATE	23
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableAbs	
ErrorHook	OSServiceId_StartScheduleTableAbs_ ScheduleTableID()	sched_explicit	
ErrorHook	OSServiceId_StartScheduleTableAbs_ value()	1	
t1	SyncScheduleTable(sched_explicit, 3)	E_OK	5
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	17
t1	StartScheduleTableRel(sched_explicit)	E_OS_STATE	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableRel	
ErrorHook	OSServiceId_StartScheduleTableRel_ ScheduleTableID()	sched_explicit	
ErrorHook	OSServiceId_StartScheduleTableRel_ offset()	1	
t1	StartScheduleTableAbs(sched_explicit)	E_OS_STATE	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableAbs	
ErrorHook	OSServiceId_StartScheduleTableAbs_ ScheduleTableID()	sched_explicit	
ErrorHook	OSServiceId_StartScheduleTableAbs_ value()	1	
t1	StartScheduleTableSynchron(sched_explicit)	E_OS_STATE	4
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableSynchron	
ErrorHook	OSServiceId_ StartScheduleTableSynchron_ ScheduleTableID()	sched_explicit	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	TerminateTask()	E_OK	

Test Sequence 4 :

```
TEST CASES:             5, 16, 27
RETURN STATUS:          STANDARD, EXTENDED
SCHEDULING POLICY:      FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_explicit {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 0;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 10;
  EXPIRY_POINT sched_explicit_offset2 {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
```

```
      ADJUSTABLE = TRUE {
        MAX_RETARD = 2;
        MAX_ADVANCE = 0;
      };
    };
    EXPIRY_POINT sched_explicit_offset5 {
      OFFSET = 5;
      ACTION = SETEVENT {
        TASK = t2;
        EVENT = Event1;
      };
      ADJUSTABLE = TRUE {
        MAX_RETARD = 2;
        MAX_ADVANCE = 0;
      };
    };
    EXPIRY_POINT sched_explicit_offset8 {
      OFFSET = 8;
      ACTION = SETEVENT {
        TASK = t2;
        EVENT = Event1;
      };
      ADJUSTABLE = TRUE {
        MAX_RETARD = 2;
        MAX_ADVANCE = 0;
      };
    };
  };
  EVENT Event1 {
    MASK = AUTO;
  };
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableSynchron(sched_explicit)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_WAITING	
t1	SyncScheduleTable(sched_explicit, 8)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	SyncScheduleTable(sched_explicit, 4)	E_OK	5
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	16
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	ClearEvent(Event1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	TerminateTask()	E_OK	

Test Sequence 5 :

```
TEST CASES:              28
RETURN STATUS:           STANDARD, EXTENDED
SCHEDULING POLICY:       FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
```

```
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_explicit {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 0;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 10;
  EXPIRY_POINT sched_explicit_offset2 {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
    ADJUSTABLE = TRUE {
      MAX_RETARD = 3;
      MAX_ADVANCE = 0;
    };
  };
  EXPIRY_POINT sched_explicit_offset5 {
    OFFSET = 5;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
    ADJUSTABLE = TRUE {
      MAX_RETARD = 2;
      MAX_ADVANCE = 0;
    };
  };
  EXPIRY_POINT sched_explicit_offset8{
    OFFSET = 8;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
```

```
      };
      ADJUSTABLE = TRUE {
        MAX_RETARD = 2;
        MAX_ADVANCE = 0;
      };
    };
  };
  EVENT Event1 {
    MASK = AUTO;
  };
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableSynchron(sched_explicit)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_WAITING	
t1	SyncScheduleTable(sched_explicit, 8)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	SyncScheduleTable(sched_explicit, 9)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	ClearEvent(Event1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	ClearEvent(Event1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	TerminateTask()	E_OK	

Test Sequence 6 :

```
TEST CASES:             29
RETURN STATUS:          STANDARD, EXTENDED
SCHEDULING POLICY:      FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_explicit {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 0;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
```

```
    LENGTH = 5;
    EXPIRY_POINT sched_explicit_offset0 {
      OFFSET = 0;
      ACTION = ACTIVATETASK {
        TASK = t2;
      };
      ADJUSTABLE = TRUE{
        MAX_RETARD = 3;
        MAX_ADVANCE = 0;
      };
    };
  };
};
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableSynchron(sched_explicit)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_WAITING	
t1	SyncScheduleTable(sched_explicit, 3)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	SyncScheduleTable(sched_explicit, 5)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	

Running task	Called OS service	Return Status	Test case
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	TerminateTask()	E_OK	

Test Sequence 7 :

```
TEST CASES:             31
RETURN STATUS:          STANDARD, EXTENDED
SCHEDULING POLICY:      FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_explicit {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 0;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
```

```
  LENGTH = 10;
  EXPIRY_POINT sched_explicit_offset2 {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
    ADJUSTABLE = TRUE {
      MAX_RETARD = 0;
      MAX_ADVANCE = 6;
    };
  };
  EXPIRY_POINT sched_explicit_offset5 {
    OFFSET = 5;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
    ADJUSTABLE = TRUE {
      MAX_RETARD = 0;
      MAX_ADVANCE = 2;
    };
  };
  EXPIRY_POINT sched_explicit_offset8 {
    OFFSET = 8;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
    ADJUSTABLE = TRUE {
      MAX_RETARD = 0;
      MAX_ADVANCE = 2;
    };
  };
};
EVENT Event1{
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableSynchron(sched_explicit)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_WAITING	
t1	SyncScheduleTable(sched_explicit, 9)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	ClearEvent(Event1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	SyncScheduleTable(sched_explicit, 0)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	TerminateTask()	E_OK	

Test Sequence 8 :

```
TEST CASES:              30
RETURN STATUS:           STANDARD, EXTENDED
SCHEDULING POLICY:       FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_explicit {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
```

```
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
      PRECISION = 0;
      SYNC_STRATEGY = EXPLICIT;
    };
    PERIODIC = TRUE;
    LENGTH = 10;
    EXPIRY_POINT sched_explicit_offset2 {
      OFFSET = 2;
      ACTION = ACTIVATETASK {
        TASK = t2;
      };
      ADJUSTABLE = TRUE {
        MAX_RETARD = 0;
        MAX_ADVANCE = 6;
      };
    };
    EXPIRY_POINT sched_explicit_offset5 {
      OFFSET = 5;
      ACTION = SETEVENT {
        TASK = t2;
        EVENT = Event1;
      };
      ADJUSTABLE = TRUE {
        MAX_RETARD = 0;
        MAX_ADVANCE = 2;
      };
    };
    EXPIRY_POINT sched_explicit_offset8 {
      OFFSET = 8;
      ACTION = SETEVENT {
        TASK = t2;
        EVENT = Event1;
      };
      ADJUSTABLE = TRUE {
        MAX_RETARD = 0;
        MAX_ADVANCE = 4;
      };
    };
  };
  EVENT Event1 {
    MASK = AUTO;
  };
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableSynchron(sched_explicit)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_WAITING	

Running task	Called OS service	Return Status	Test case
t1	SyncScheduleTable(sched_explicit, 9)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	SyncScheduleTable(sched_explicit, 0)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	ClearEvent(Event1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	TerminateTask()	E_OK	

Test Sequence 9 :

```
TEST CASES:              32
RETURN STATUS:           STANDARD, EXTENDED
SCHEDULING POLICY:       FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_explicit {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 0;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 5;
  EXPIRY_POINT sched_explicit_offset0 {
    OFFSET = 0;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
    ADJUSTABLE = TRUE {
      MAX_RETARD = 0;
      MAX_ADVANCE = 3;
```

```
    };
  };
};
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableSynchron(sched_explicit)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_WAITING	
t1	SyncScheduleTable(sched_explicit, 4)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	SyncScheduleTable(sched_explicit, 0)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	TerminateTask()	E_OK	

Test Sequence 10 :

```
TEST CASES:              11, 12, 36
RETURN STATUS:           STANDARD, EXTENDED
SCHEDULING POLICY:       FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_explicit {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 0;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 10;
```

```
  EXPIRY_POINT sched_explicit_offset2 {
   OFFSET = 2;
   ACTION = ACTIVATETASK {
    TASK = t2;
   };
   ADJUSTABLE = TRUE {
     MAX_RETARD = 2;
     MAX_ADVANCE = 0;
   };
  };
  EXPIRY_POINT sched_explicit_offset5 {
    OFFSET = 5;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
    ADJUSTABLE = TRUE {
      MAX_RETARD = 2;
      MAX_ADVANCE = 0;
    };
  };
  EXPIRY_POINT sched_explicit_offset8 {
    OFFSET = 8;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
    ADJUSTABLE = TRUE {
      MAX_RETARD = 2;
      MAX_ADVANCE = 0;
    };
  };
};
EVENT Event1 {
  MASK = AUTO;
};
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableSynchron(sched_explicit)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_WAITING	
t1	SyncScheduleTable(sched_explicit, 8)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	SetScheduleTableAsync(sched_explicit)	E_OK	11
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	ClearEvent(Event1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	SyncScheduleTable(sched_explicit, 8)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	

Running task	Called OS service	Return Status	Test case
t1	SyncScheduleTable(sched_explicit, 4)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	SetScheduleTableAsync(sched_explicit)	E_OK	12
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	ClearEvent(Event1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	TerminateTask()	E_OK	

Test Sequence 11 :

```
TEST CASES:            33
RETURN STATUS:         STANDARD, EXTENDED
SCHEDULING POLICY:     FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
```

```
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_explicit {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 1;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 10;
  EXPIRY_POINT sched_explicit_offset2 {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
    ADJUSTABLE = TRUE {
      MAX_RETARD = 2;
      MAX_ADVANCE = 0;
    };
  };
  EXPIRY_POINT sched_explicit_offset5 {
    OFFSET = 5;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
    ADJUSTABLE = TRUE {
      MAX_RETARD = 2;
      MAX_ADVANCE = 0;
    };
  };
  EXPIRY_POINT sched_explicit_offset8 {
    OFFSET = 8;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
    ADJUSTABLE = TRUE {
      MAX_RETARD = 2;
      MAX_ADVANCE = 0;
    };
  };
};
EVENT Event1 {
  MASK = AUTO;
```

};

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableSynchron(sched_explicit)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_WAITING	
t1	SyncScheduleTable(sched_explicit, 8)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	SyncScheduleTable(sched_explicit, 4)	E_OK	5
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	16
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	ClearEvent(Event1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	TerminateTask()	E_OK	

Test Sequence 12 :

```
TEST CASES:            34
RETURN STATUS:         STANDARD, EXTENDED
SCHEDULING POLICY:     FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_explicit {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 1;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 10;
  EXPIRY_POINT sched_explicit_offset2 {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t2;
```

```
      };
      ADJUSTABLE = TRUE {
        MAX_RETARD = 3;
        MAX_ADVANCE = 0;
      };
    };
    EXPIRY_POINT sched_explicit_offset5 {
      OFFSET = 5;
      ACTION = SETEVENT {
        TASK = t2;
        EVENT = Event1;
      };
      ADJUSTABLE = TRUE {
        MAX_RETARD = 2;
        MAX_ADVANCE = 0;
      };
    };
    EXPIRY_POINT sched_explicit_offset8 {
      OFFSET = 8;
      ACTION = SETEVENT {
        TASK = t2;
        EVENT = Event1;
      };
      ADJUSTABLE = TRUE {
        MAX_RETARD = 2;
        MAX_ADVANCE = 0;
      };
    };
  };
  EVENT Event1 {
    MASK = AUTO;
  };
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableSynchron(sched_explicit)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_WAITING	
t1	SyncScheduleTable(sched_explicit, 8)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	SyncScheduleTable(sched_explicit, 9)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	ClearEvent(Event1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	ClearEvent(Event1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	

Running task	Called OS service	Return Status	Test case
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	TerminateTask()	E_OK	

Test Sequence 13 :

```
TEST CASES:              35, 37
RETURN STATUS:           STANDARD, EXTENDED
SCHEDULING POLICY:       FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_explicit {
  COUNTER = Software_Counter1;
  AUTOSTART = SYNCHRON { APPMODE = std; };
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 1;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 5;
  EXPIRY_POINT sched_explicit_offset0 {
    OFFSET = 0;
    ACTION = ACTIVATETASK {
      TASK = t2;
    };
    ADJUSTABLE = TRUE {
      MAX_RETARD = 3;
      MAX_ADVANCE = 0;
    };
  };
```

};

Running task	Called OS service	Return Status	Test case
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_WAITING	
t1	SyncScheduleTable(sched_explicit, 3)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	SyncScheduleTable(sched_explicit, 5)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched_explicit, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	TerminateTask()	E_OK	

Test Sequence 14 :

```
TEST CASES:              19, 22, 27
RETURN STATUS:           STANDARD, EXTENDED
SCHEDULING POLICY:       FULL-PREEMPTIVE
HOOKS:       ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched1 {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 0;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 3;
  EXPIRY_POINT sched1_offset1 {
```

```
      OFFSET = 1;
      ACTION = ACTIVATETASK {
        TASK = t2;
      };
      ADJUSTABLE = TRUE {
        MAX_RETARD = 2;
        MAX_ADVANCE = 0;
      };
    };
  };
  SCHEDULETABLE sched2 {
    COUNTER = Software_Counter1;
    AUTOSTART = NONE;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
      PRECISION = 0;
      SYNC_STRATEGY = EXPLICIT;
    };
    PERIODIC = TRUE;
    LENGTH = 5;
    EXPIRY_POINT sched2_offset0 {
      OFFSET = 0;
      ACTION = SETEVENT {
        TASK = t2;
        EVENT = Event1;
      };
    };
  };
  EVENT Event1 {
    MASK = AUTO;
  };
```

Running task	Called OS service	Return Status	Test case
t1	StartScheduleTableAbs(sched1, 1)	E_OK	22
t1	StartScheduleTableRel(sched2, 1)	E_OK	19
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
ErrorHook	OSErrorGetServiceId()	OSServiceId_SetEvent	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	WaitEvent(Event1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	

Running task	Called OS service	Return Status	Test case
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ActivateTask	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	StopScheduleTable(sched2)	E_OK	
t1	SyncScheduleTable(sched1, 3)	E_OK	
t1	GetScheduleTableStatus(sched2 &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_STOPPED	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	IncrementCounter(Software_Counter1)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING	
t1	IncrementCounter(Software_Counter1)	E_OK	
t2	TerminateTask()	E_OK	
t1	GetScheduleTableStatus(sched2, &STStatusType)	E_OK, STStatusType = SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS	
t1	TerminateTask()	E_OK	

2.12 AUTOSAR - OS-Application

2.12.1 API Service Calls for OS objects

Goil 2b50 and after emits an error when the scalability class is 2 or 4 and the objects are not all in an OS Application. This renders test sequence 2 obsolete. This test sequence is no longer included in the test suite.
CheckTaskMemoryAccess() and CheckISRMemoryAccess()...

Test Sequence 1 :

```
TEST CASES:             38, 39, 40, 42
RETURN STATUS:          EXTENDED
SCHEDULING POLICY:      FULL_PREEMPTIVE
HOOK:                   ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = Resource1;
};
TASK t2 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
ALARM Alarm1 {
  COUNTER = Software_Counter;
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = FALSE;
};
SCHEDULETABLE sched1 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 3;
  EXPIRY_POINT first_point {
    OFFSET = 0;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
RESOURCE Resource1 {
  RESOURCEPROPERTY = STANDARD;
};
APPLICATION app1 {
```

```
    TASK = t1;
    ALARM = Alarm1;
    RESOURCE = Resource1;
    COUNTER = Software_Counter;
    SCHEDULETABLE = sched1;
    HAS_RESTARTTASK = TRUE { RESTARTTASK = t1; };
  };
  APPLICATION app2 {
    TASK = t2;
    TRUSTED = TRUE;
  };
```

Running task	Called OS service	Return Status	Test case
t1	GetApplicationID()	app1	42
t1	SetRelAlarm(Alarm1, 1, 0)	E_OK	
t1	GetAlarm(Alarm1, &AlarmBaseType)	E_OK	
t1	StartScheduleTableAbs(sched1, 0)	E_OK	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType=RUNNINNG	
t1	GetResource()	E_OK	
t1	TerminateApplication(INVALID_RESTART)	E_OS_VALUE	38
ErrorHook	OSErrorGetServiceId()	OSServiceId_ TerminateApplication	
t1	TerminateApplication(RESTART)		40
t1	GetApplicationID()	app1	
t1	GetAlarm(Alarm1, &AlarmBaseType)	E_OS_NOFUNC	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OK, STStatusType=STOPPED	
t1	ReleaseResource()	E_OS_NOFUNC	
t1	TerminateApplication(NO_RESTART)		39
t2	GetApplicationID()	app2	
t2	TerminateTask()	E_OK	

Test Sequence 2 :

TEST CASES: 41
RETURN STATUS: EXTENDED
SCHEDULING POLICY: FULL–PREEMPTIVE

Running task	Called OS service	Return Status	Test case
t1	GetApplicationID()	INVALID_OSAPPLICATION	41
t1	TerminateApplication()	E_OS_CALLEVEL	
t1	TerminateTask()	E_OK	

2.12.2 Access Rights for objects in API services

Test Sequence 3 :

```
TEST CASES:             1 to 37
RETURN STATUS:          EXTENDED
SCHEDULING POLICY:      FULL_PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = Resource1;
  ACCESSING_APPLICATION = app2;
};

TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

ISR isr1 {
  CATEGORY = 2;
  PRIORITY = 1;
  SOURCE = SIGTERM;
  STACKSIZE = 32768;
};

COUNTER Software_Counter {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
COUNTER Software_Counter2 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};

ALARM Alarm1{
  COUNTER = Software_Counter;
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = FALSE;
};

SCHEDULETABLE sched1 {
  COUNTER = Software_Counter;
```

```
    AUTOSTART = NONE;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
    PERIODIC = FALSE;
    LENGTH = 3;
    EXPIRY_POINT first_point {
      OFFSET = 0;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };

  SCHEDULETABLE sched2 {
    COUNTER = Software_Counter2;
    AUTOSTART = NONE;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
    PERIODIC = FALSE;
    LENGTH = 1;
    EXPIRY_POINT many_things {
      OFFSET = 1;
      ACTION = ACTIVATETASK {
        TASK = t2;
      };
    };
  };

  RESOURCE Resource1{
    RESOURCEPROPERTY = STANDARD;
  };
  APPLICATION app1 {
    ALARM = Alarm1;
    COUNTER = Software_Counter;
    RESOURCE = Resource1;
    SCHEDULETABLE = sched1;
    TASK = t1;
    TRUSTED = TRUE;
  };

  APPLICATION app2 {
    COUNTER = Software_Counter2;
    ISR = isr1;
    SCHEDULETABLE = sched2;
    TASK = t2;
  };
```

Running task	Called OS service	Return Status	Test case
t1	CheckObjectAccess(INVALID_OSAPPLICATION, OBJECT_TASK, t1)	NO_ACCESS	1

Running task	Called OS service	Return Status	Test case
t1	CheckObjectAccess(app1, OBJECT_TYPE_COUNT, t1)	NO_ACCESS	2
t1	CheckObjectAccess(app1, OBJECT_TASK, INVALID_TASK)	NO_ACCESS	3
t1	CheckObjectAccess(app1, OBJECT_TASK, t1)	ACCESS	4
t1	CheckObjectAccess(app1, OBJECT_TASK, t2)	NO_ACCESS	5
t1	CheckObjectAccess(app2, OBJECT_ISR, INVALID_ISR)	NO_ACCESS	6
t1	CheckObjectAccess(app2, OBJECT_ISR, isr1)	ACCESS	7
t1	CheckObjectAccess(app1, OBJECT_ISR, isr1)	NO_ACCESS	8
t1	CheckObjectAccess(app2, OBJECT_ALARM, INVALID_ALARM)	NO_ACCESS	9
t1	CheckObjectAccess(app1, OBJECT_ALARM, Alarm1)	ACCESS	10
t1	CheckObjectAccess(app2, OBJECT_ALARM, Alarm1)	NO_ACCESS	11
t1	CheckObjectAccess(app1, OBJECT_RESOURCE, INVALID_RESOURCE)	NO_ACCESS	12
t1	CheckObjectAccess(app1, OBJECT_RESOURCE, Resource1)	ACCESS	13
t1	CheckObjectAccess(app2, OBJECT_RESOURCE, Resource1)	NO_ACCESS	14
t1	CheckObjectAccess(app1, OBJECT_RESOURCE, RES_SCHEDULER)	ACCESS	15
t1	CheckObjectAccess(app2, OBJECT_RESOURCE, RES_SCHEDULER)	ACCESS	15
t1	CheckObjectAccess(app1, OBJECT_ SCHEDULETABLE, INVALID_SCHEDULETABLE)	NO_ACCESS	16
t1	CheckObjectAccess(app1, OBJECT_ SCHEDULETABLE, sched1)	ACCESS	17
t1	CheckObjectAccess(app1, OBJECT_ SCHEDULETABLE, sched2)	NO_ACCESS	18
t1	CheckObjectAccess(app1, OBJECT_COUNTER, INVALID_COUNTER)	NO_ACCESS	19
t1	CheckObjectAccess(app1, OBJECT_COUNTER, Software_Counter)	ACCESS	20
t1	CheckObjectAccess(app2, OBJECT_COUNTER, Software_Counter)	NO_ACCESS	21
t1	CheckObjectAccess(app1, OBJECT_COUNTER, SystemCounter)	NO_ACCESS	22

Running task	Called OS service	Return Status	Test case
t1	CheckObjectOwnerShip(OBJECT_TYPE_COUNT, t1)	INVALID_OSAPPLICATION	23
t1	CheckObjectOwnerShip(OBJECT_TASK, INVALID_TASK)	INVALID_OSAPPLICATION	24
t1	CheckObjectOwnerShip(OBJECT_TASK, t1)	app1	25
t1	CheckObjectOwnerShip(OBJECT_ISR, INVALID_ISR)	INVALID_OSAPPLICATION	26
t1	CheckObjectOwnerShip(OBJECT_ISR, isr1)	app2	27
t1	CheckObjectOwnerShip(OBJECT_ALARM, INVALID_ALARM)	INVALID_OSAPPLICATION	28
t1	CheckObjectOwnerShip(OBJECT_ALARM, Alarm1)	app1	29
t1	CheckObjectOwnerShip(OBJECT_RESOURCE, INVALID_RESOURCE)	INVALID_OSAPPLICATION	30
t1	CheckObjectOwnerShip(OBJECT_RESOURCE, Resource1)	app1	31
t1	CheckObjectOwnerShip(OBJECT_RESOURCE, RES_SCHEDULER)	INVALID_OSAPPLICATION	32
t1	CheckObjectOwnerShip(OBJECT_ SCHEDULETABLE, INVALID_SCHEDULETABLE)	INVALID_OSAPPLICATION	33
t1	CheckObjectOwnerShip(OBJECT_ SCHEDULETABLE, sched1)	app1	34
t1	CheckObjectOwnerShip(OBJECT_COUNTER, INVALID_COUNTER)	INVALID_OSAPPLICATION	35
t1	CheckObjectOwnerShip(OBJECT_COUNTER, Resource1)	app1	36
t1	CheckObjectOwnerShip(OBJECT_COUNTER, SystemCounter)	INVALID_OSAPPLICATION	37
t1	TerminateTask()	E_OK	

Test Sequence 4 :

```
TEST CASES:              1 to 46
RETURN STATUS:           EXTENDED
SCHEDULING POLICY:       FULL-PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = Resource1;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
```

```
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event2;
  RESOURCE = Resource2;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event3;
  ACCESSING_APPLICATION = app1;
};
COUNTER Software_Counter1 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
COUNTER Software_Counter2 {
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
ALARM Alarm1 {
  COUNTER = Software_Counter1;
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = FALSE;
};
ALARM Alarm2 {
  COUNTER = Software_Counter2;
  ACTION = ACTIVATETASK {
    TASK = t2;
  };
  AUTOSTART = FALSE;
};
SCHEDULETABLE sched1 {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 1;
    SYNC_STRATEGY = EXPLICIT;
  };
  PERIODIC = TRUE;
  LENGTH = 3;
  EXPIRY_POINT first_point {
    OFFSET = 0;
```

```
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  SCHEDULETABLE sched2 {
    COUNTER = Software_Counter2;
    AUTOSTART = NONE;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
      PRECISION = 1;
      SYNC_STRATEGY = EXPLICIT;
    };
    PERIODIC = TRUE;
    LENGTH = 3;
    EXPIRY_POINT first_point {
      OFFSET = 0;
      ACTION = ACTIVATETASK {
        TASK = t2;
      };
    };
  };
  SCHEDULETABLE sched3 {
    COUNTER = Software_Counter1;
    AUTOSTART = NONE;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
      PRECISION = 1;
      SYNC_STRATEGY = EXPLICIT;
    };
    PERIODIC = TRUE;
    LENGTH = 3;
    EXPIRY_POINT first_point {
      OFFSET = 0;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  RESOURCE Resource1 {
    RESOURCEPROPERTY = STANDARD;
  };
  RESOURCE Resource2 {
    RESOURCEPROPERTY = STANDARD;
  };
  APPLICATION app1 {
    TASK = t1;
    COUNTER = Software_Counter1;
    ALARM = Alarm1;
    RESOURCE = Resource1;
    SCHEDULETABLE = sched1;
    SCHEDULETABLE = sched3;
```

```
};
APPLICATION app2 {
  TASK = t2;
  COUNTER = Software_Counter2;
  ALARM = Alarm2;
  RESOURCE = Resource2;
  SCHEDULETABLE = sched2;
};
APPLICATION app3 {
  TASK = t3;
  TRUSTED = TRUE;
};
EVENT Event2 {
  MASK = 2;
};
EVENT Event3{
  MASK = 4;
};
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OS_ACCESS	1
t1	ActivateTask(t3)	E_OK	2
t3	WaitEvent(Event3)	E_OK	
t1	GetTaskState(t2, &TaskStateType)	E_OS_ACCESS	5
t1	GetTaskState(t3, &TaskStateType)	E_OK, State=WAITING	6
t1	SetEvent(t2, Event2)	E_OS_ACCESS	11
t1	SetEvent(t3, Event3)	E_OK	12
t3	WaitEvent(Event2)	E_OK	
t1	GetEvent(t2, &EventMaskType)	E_OS_ACCESS	13
t1	GetEvent(t3, &EventMaskType)	E_OK, Event=Event3	14
t1	GetResource(Resource2)	E_OS_ACCESS	7
t1	GetResource(Resource1)	E_OK	8
t1	ReleaseResource(Resource1)	E_OK	10
t1	ReleaseResource(Resource2)	E_OS_ACCESS	9
t1	SetAbsAlarm(Alarm2, 1, 0)	E_OS_ACCESS	21
t1	SetAbsAlarm(Alarm1, 1, 0)	E_OK	22
t1	GetAlarmBase(Alarm2, &AlarmBaseType)	E_OS_ACCESS	15
t1	GetAlarmBase(Alarm1, &AlarmBaseType)	E_OK	16
t1	GetAlarm(Alarm2, &TickType)	E_OS_ACCESS	17
t1	GetAlarm(Alarm1, &TickType)	E_OK	18
t1	CancelAlarm(Alarm2)	E_OS_ACCESS	23
t1	CancelAlarm(Alarm1)	E_OK	24
t1	SetRelAlarm(Alarm2, 1, 0)	E_OS_ACCESS	19

Running task	Called OS service	Return Status	Test case
t1	SetRelAlarm(Alarm1, 1, 0)	E_OK	20
t1	CancelAlarm(Alarm1)	E_OK	
t1	StartScheduleTableAbs(sched2, 0)	E_OS_ACCESS	27
t1	StartScheduleTableAbs(sched1, 0)	E_OK	28
t1	StopScheduleTable(sched2)	E_OS_ACCESS	29
t1	StopScheduleTable(sched1)	E_OK	30
t1	StartScheduleTableRel(sched2, 0)	E_OS_ACCESS	25
t1	StartScheduleTableRel(sched1, 0)	E_OK	26
t1	NextScheduleTable(sched1, sched2)	E_OS_ACCESS	31
t1	NextScheduleTable(sched2, sched1)	E_OS_ACCESS	31
t1	NextScheduleTable(sched1, sched3)	E_OK	32
t1	StopScheduleTable(sched1)	E_OK	
t1	StartScheduleTableSynchron(sched2)	E_OS_ACCESS	33
t1	StartScheduleTableSynchron(sched1)	E_OK	34
t1	SyncScheduleTable(sched2, 0)	E_OS_ACCESS	35
t1	SyncScheduleTable(sched1, 0)	E_OK	36
t1	SetScheduleTableAsync(sched2)	E_OS_ACCESS	37
t1	SetScheduleTableAsync(sched1)	E_OK	38
t1	GetScheduleTableStatus(sched2)	E_OS_ACCESS	39
t1	GetScheduleTableStatus(sched1)	E_OK, State= SCHEDULETABLE_ RUNNING	40
t1	IncrementCounter(Software_Counter2)	E_OS_ACCESS	41
t1	IncrementCounter(Software_Counter1)	E_OK	42
t1	GetCounterValue(Software_Counter2, &TickType)	E_OS_ACCESS	43
t1	GetCounterValue(Software_Counter1, &TickType)	E_OK, TickType=1	44
t1	GetElapsedCounterValue(Software_Counter2, 0, &TickType)	E_OS_ACCESS	45
t1	GetElapsedCounterValue(Software_Counter1, 0, &TickType)	E_OK, TickType=1	46
t1	SetEvent(t3, Event2)	E_OK	
t3			
t1	ChainTask(t2)	E_OS_ACCESS	3
t1	ChainTask(t3)	E_OK	4
t3	TerminateTask()	E_OK	

2.13 AUTOSAR - Service Protection

Test case 10 has already been tested in *interrupts_s6* because OSEK OS specify that no service is allowed afeter disabling interrupts.
Test case 12 has already been tested in *hook_s2*. If PosttaskHooh were called, an error should have occured.

Test Sequence 1 :

```
TEST CASES:              2, 10
RETURN STATUS:           EXTENDED
SCHEDULING POLICY:       FULL-PREEMPTIVE
HOOKS:                   ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART =  FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
ISR isr1 {
  CATEGORY = 2;
  SOURCE = SIGTERM;
  PRIORITY = 1;
  STACKSIZE = 32768;
};
ISR isr2 {
  CATEGORY = 2;
  SOURCE = SIGUSR2;
  PRIORITY = 1;
  STACKSIZE = 32768;
};
```

Running task	Called OS service	Return Status	Test case
t1	ActivateTask(t2)	E_OK	
t1	DisableAllInterrupts()		
t1	*trigger interrupt isr2*		
t1	TerminateTask()	E_OS_DISABLEDINT	10
ErrorHook	OSErrorGetServiceId	OSServiceId_TerminateTask	
t1	*trigger interrupt isr2*		
t1	TerminateTask()	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId	OSServiceId_TerminateTask	
t1	*trigger interrupt isr2*		
ErrorHook	StatusError	E_OS_MISSINGEND	2
isr2			
t2	*trigger interrupt isr1*		
isr1			
t2	TerminateTask()		

Running task	Called OS service	Return Status	Test case

Test Sequence 2 :

```
TEST CASES:             6
RETURN STATUS:          EXTENDED
SCHEDULING POLICY:      FULL_PREEMPTIVE
HOOKS:                  ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
ISR isr1 {
  CATEGORY = 2;
  SOURCE = SIGTERM;
  PRIORITY = 1;
  STACKSIZE = 32768;
};
ISR isr2 {
  CATEGORY = 2;
  SOURCE = SIGUSR2;
  PRIORITY = 2;
  STACKSIZE = 32768;
};
```

Running task	Called OS service	Return Status	Test case
t1	*trigger interrupt isr1*		
isr1	DisableAllInterrupts()		
isr1	*trigger interrupt isr2*		
ErrorHook	StatusError	E_OS_DISABLEDINT	6
isr2			
t1	TerminateTask()		

Test Sequence 3 :

```
TEST CASES:             1, 3, 4, 5
RETURN STATUS:          EXTENDED
SCHEDULING POLICY:      FULL_PREEMPTIVE
HOOKS:                  ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = Resource1;
  RESOURCE = Resource2;
```

```
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = Resource1;
  RESOURCE = Resource2;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
RESOURCE Resource1{
  RESOURCEPROPERTY = STANDARD;
};
RESOURCE Resource2{
  RESOURCEPROPERTY = STANDARD;
};
```

Running task	Called OS service	Return Status	Test case
t1	GetResource(Resource1)	E_OK	
t1	ActivateTask(t2)	E_OK	
t1	TerminateTask()	E_OS_RESOURCE	
ErrorHook	OSErrorGetServiceId	OSServiceId_TerminateTask	
ErrorHook	StatusError	E_OS_RESOURCE	
t1	ActivateTask(t3)	E_OK	
t3			
ErrorHook	GetTaskID()	E_OK, TaskID=t3	
ErrorHook	GetTaskState(TaskID, &TaskStateType)	E_OK, TaskState-Type=RUNNING	
ErrorHook	StatusError	E_OS_MISSINGEND	1, 5
PostTaskHook			
t1	TerminateTask()	E_OS_RESOURCE	
ErrorHook	OSErrorGetServiceId	OSServiceId_TerminateTask	
ErrorHook	StatusError	E_OS_RESOURCE	
t1			
ErrorHook	StatusError	E_OS_MISSINGEND	3
t2	GetResource(Resource1)	E_OK	
t2	GetResource(Resource2)	E_OK	
t2	ActivateTask(t1)	E_OK	
ErrorHook	StatusError	E_OS_MISSINGEND	4

Running task	Called OS service	Return Status	Test case
t1	GetResource(Resource2)	E_OK	
t1	GetResource(Resource1)	E_OK	
t1	ReleaseResource(Resource1)	E_OK	
t1	ReleaseResource(Resource2)	E_OK	
t1	TerminateTask()	E_OK	

Test Sequence 4 :

```
TEST CASES:             7, 8, 9
RETURN STATUS:          EXTENDED
SCHEDULING POLICY:      FULL_PREEMPTIVE
HOOKS:                  ErrorHook

TASK t1 {
  AUTOSTART =  TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
ISR isr1 {
  CATEGORY = 2;
  SOURCE = SIGTERM;
  PRIORITY = 1;
  STACKSIZE = 32768;
  RESOURCE = Resource1;
  RESOURCE = Resource2;
};
ISR isr2 {
  CATEGORY = 2;
  SOURCE = SIGUSR2;
  PRIORITY = 2;
  STACKSIZE = 32768;
  RESOURCE = Resource1;
  RESOURCE = Resource2;
};
ISR isr3 {
  CATEGORY = 2;
  SOURCE = SIGPIPE;
  PRIORITY = 3;
  STACKSIZE = 32768;
};
RESOURCE Resource1 {
  RESOURCEPROPERTY = STANDARD;
};
RESOURCE Resource2 {
  RESOURCEPROPERTY = STANDARD;
};
```

Running task	Called OS service	Return Status	Test case
t1	*trigger interrupt isr1*		
isr1	GetResource(Resource1)	E_OK	
isr1	*trigger interrupt isr2*		
isr1	*trigger interrupt isr3*		
isr3			9
isr1			
ErrorHook	StatusError	E_OS_RESOURCE	7
isr2	GetResource(Resource1)	E_OK	
isr2	GetResource(Resource2)	E_OK	
isr2	*trigger interrupt isr1*		
ErrorHook	StatusError	E_OS_RESOURCE	8
isr1	GetResource(Resource2)	E_OK	
isr1	GetResource(Resource1)	E_OK	
isr1	ReleaseResource(Resource1)	E_OK	
isr1	ReleaseResource(Resource2)	E_OK	
t1	TerminateTask()	E_OK	

Test Sequence 5 :

```
TEST CASES:              10
RETURN STATUS:           EXTENDED
SCHEDULING POLICY:       FULL-PREEMPTIVE
HOOKS:                   ErrorHook

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter1{
  MAXALLOWEDVALUE = 100;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched1 {
  COUNTER = Software_Counter1;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 1;
```

```
      SYNC_STRATEGY = EXPLICIT;
    };
    PERIODIC = TRUE;
    LENGTH = 3;
    EXPIRY_POINT first_point {
      OFFSET = 0;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  SCHEDULETABLE sched2 {
    COUNTER = Software_Counter1;
    AUTOSTART = NONE;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
      PRECISION = 1;
      SYNC_STRATEGY = EXPLICIT;
    };
    PERIODIC = TRUE;
    LENGTH = 3;
    EXPIRY_POINT first_point {
      OFFSET = 0;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  APPLICATION app1 {
    TASK = t1;
    TASK = t2;
    COUNTER = Software_Counter1;
    SCHEDULETABLE = sched1;
    SCHEDULETABLE = sched2;
    TRUSTED = TRUE;
  };
```

Running task	Called OS service	Return Status	Test case
t1	DisableAllInterrupts()		
t1	GetApplicationID()	INVALID_OSAPPLICATION	10
ErrorHook	OSErrorGetServiceId()	OSServiceId_GetApplicationID	
t1	GetISRID()	INVALID_ISR	
ErrorHook	OSErrorGetServiceId()	OSServiceId_GetISRID	
t1	ChecObjectAccess()	NO_ACCESS	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ ChecObjectAccess	

Running task	Called OS service	Return Status	Test case
t1	ChecObjectOwnership()	INVALID_OSAPPLICATION	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ ChecObjectOwnership	
t1	StartScheduleTableRel(sched1, 1)	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableRel	
t1	StartScheduleTableAbs(sched1, 0)	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableAbs	
t1	StopScheduleTable(sched1)	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StopScheduleTable	
t1	NextScheduleTable(sched1, sched2)	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ NextScheduleTable	
t1	StartScheduleTableSynchron(sched1)	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ StartScheduleTableSynchron	
t1	SyncScheduleTable(sched1, 0)	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ SyncScheduleTable	
t1	GetScheduleTableStatus(sched1, &STStatusType)	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ GetScheduleTableStatus	
t1	SetScheduleTableAsync(sched1)	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ SetScheduleTableAsync	
t1	IncrementCounter(Software_Counter1)	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ IncrementCounter	
t1	GetCounterValue(Software_Counter1, &TickType)	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ GetCounterValue	
t1	GetElapsedCounterValue(Software_Counter1, &TickType1, &TickType2)	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ GetElapsedCounterValue	
t1	TerminateApplication(NO_RESTART)	E_OS_DISABLEDINT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ TerminateApplication	

Running task	Called OS service	Return Status	Test case
t1	EnableAllInterrupts()		
t1	TerminateTask()	E_OK	

2.14 AUTOSAR - Memory Protection

Test Sequence 1 :

```
TEST CASES:             1, 2, 5, 6, 7
RETURN STATUS:          EXTENDED
SCHEDULING POLICY:      NON-PREEMPTIVE
HOOKS:                  ProtectionHook

  TASK t1_app_nontrusted1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON;
     EVENT = Event1;
};
TASK t1_app_nontrusted2 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
TASK t1_app_trusted1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
     ACCESSING_APPLICATION = app_nontrusted1;
};
TASK t1_app_trusted2 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
APPLICATION app_nontrusted1 {
     TRUSTED = FALSE;
  TASK = t1_app_nontrusted1;
  HAS_RESTARTTASK = TRUE { RESTARTTASK = t1_app_nontrusted1; };
};
APPLICATION app_nontrusted2 {
     TRUSTED = FALSE;
  TASK = t1_app_nontrusted2;
};
APPLICATION app_trusted1 {
```

```
    TRUSTED = TRUE;
  TASK = t1_app_trusted1;
};
APPLICATION app_trusted2 {
    TRUSTED = TRUE;
  TASK = t1_app_trusted2;
};
  EVENT Event1 { MASK = AUTO; };
```

Running task	Called OS service	Return Status	Test case
t1_app_nontrusted1	Read its own OS application datas	Access allowed	5a
t1_app_nontrusted1	Write its own OS application datas	Access allowed	5b
t1_app_nontrusted1	Read trusted OS application datas	Call ProtectionHook with E_OS_PROTECTION_MEMORY	6e
t1_app_nontrusted1	Write trusted OS application datas	Call ProtectionHook with E_OS_PROTECTION_MEMORY	6f
t1_app_nontrusted1	Read other non-trusted OS application datas	Call ProtectionHook with E_OS_PROTECTION_MEMORY	6c
t1_app_nontrusted1	Write other non-trusted OS application datas	Call ProtectionHook with E_OS_PROTECTION_MEMORY	6d
t1_app_nontrusted1	Read OS datas from non-truted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	1a
t1_app_nontrusted1	Write OS datas from non-truted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	1b
t1_app_nontrusted1	WaitEvent(Event1)	E_OK	
t1_app_trusted1	Read its own OS application datas	Access allowed	7a
t1_app_trusted1	Write its own OS application datas	Access allowed	7b
t1_app_trusted1	Read other trusted OS application datas	Access allowed	7e
t1_app_trusted1	Write other trusted OS application datas	Access allowed	7f
t1_app_trusted1	Read non-trusted OS application datas	Access allowed	7c
t1_app_trusted1	Write non-trusted OS application datas	Access allowed	7d
t1_app_trusted1	Read OS datas from trusted OS application	Access allowed	2a
t1_app_trusted1	Write OS datas from trusted OS application	Access allowed	2b
t1_app_trusted1	SetEvent(t1_app_nontrusted1, Event1)	E_OK	
t1_app_trusted1	TerminateTask()	E_OK	
t1_app_nontrusted1	TerminateTask()	E_OK	

Test Sequence 2 :

```
TEST CASES:             8, 9
RETURN STATUS:          EXTENDED
SCHEDULING POLICY:      NON-PREEMPTIVE
HOOKS:                  ProtectionHook

  TASK t1_app_nontrusted1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 2;
  ACTIVATION = 1;
```

```
  SCHEDULE = NON;
      EVENT = Event1;
};
  TASK t2_app_nontrusted1 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
TASK t1_app_nontrusted2 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
TASK t1_app_trusted1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
      ACCESSING_APPLICATION = app_nontrusted1;
};
TASK t2_app_trusted1 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
TASK t1_app_trusted2 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
APPLICATION app_nontrusted1 {
      TRUSTED = FALSE;
  TASK = t1_app_nontrusted1;
  TASK = t2_app_nontrusted1;
};
APPLICATION app_nontrusted2 {
      TRUSTED = FALSE;
  TASK = t1_app_nontrusted2;
};
APPLICATION app_trusted1 {
      TRUSTED = TRUE;
  TASK = t1_app_trusted1;
  TASK = t2_app_trusted1;
};
APPLICATION app_trusted2 {
      TRUSTED = TRUE;
```

```
  TASK = t1_app_trusted2;
};
  EVENT Event1 { MASK = AUTO; };
```

Running task	Called OS service	Return Status	Test case
t1_app_nontrusted1	Read/Write its own Task/OsIsr datas from non-trusted OS app.	Access allowed	
t1_app_nontrusted1	Read Task/OsIsr datas in the same non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	8a
t1_app_nontrusted1	Write Task/OsIsr datas in the same non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	8b
t1_app_nontrusted1	Read Task/OsIsr datas in the other non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	8c
t1_app_nontrusted1	Write Task/OsIsr datas in the other non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	8d
t1_app_nontrusted1	Read Task/OsIsr datas in trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	8e
t1_app_nontrusted1	Write Task/OsIsr datas in trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	8f
t1_app_nontrusted1	WaitEvent(Event1)	E_OK	
t1_app_trusted1	Read/Write its own Task/OsIsr datas from trusted OS app.	Access allowed	
t1_app_trusted1	Read Task/OsIsr datas in the same trusted OS application	Access allowed	9a
t1_app_trusted1	Write Task/OsIsr datas in the same trusted OS application	Access allowed	9b
t1_app_trusted1	Read Task/OsIsr datas in the other trusted OS application	Access allowed	9e
t1_app_trusted1	Write Task/OsIsr datas in the other trusted OS application	Access allowed	9f
t1_app_trusted1	Read Task/OsIsr datas in non-trusted OS application	Access allowed	9c
t1_app_trusted1	Write Task/OsIsr datas in non-trusted OS application	Access allowed	9d
t1_app_trusted1	SetEvent(t1_app_nontrusted1, Event1)	E_OK	
t1_app_trusted1	TerminateTask()	E_OK	
t1_app_nontrusted1	TerminateTask()	E_OK	

Test Sequence 3 :

```
TEST CASES:          3, 4, 10, 11
RETURN STATUS:       EXTENDED
SCHEDULING POLICY:   NON-PREEMPTIVE
HOOKS:               ProtectionHook

  TASK t1_app_nontrusted1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
```

```
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON;
     EVENT = Event1;
};
  TASK t2_app_nontrusted1 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
TASK t1_app_nontrusted2 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
TASK t1_app_trusted1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
     ACCESSING_APPLICATION = app_nontrusted1;
};
TASK t2_app_trusted1 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
TASK t1_app_trusted2 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
};
APPLICATION app_nontrusted1 {
     TRUSTED = FALSE;
  TASK = t1_app_nontrusted1;
  TASK = t2_app_nontrusted1;
};
APPLICATION app_nontrusted2 {
     TRUSTED = FALSE;
  TASK = t1_app_nontrusted2;
};
APPLICATION app_trusted1 {
     TRUSTED = TRUE;
  TASK = t1_app_trusted1;
  TASK = t2_app_trusted1;
};
```

```
APPLICATION app_trusted2 {
      TRUSTED = TRUE;
  TASK = t1_app_trusted2;
};
  EVENT Event1 { MASK = AUTO; };
```

Running task	Called OS service	Return Status	Test case
t1_app_nontrusted1	DisableAllInterrupts()		
t1_app_nontrusted1	Read/Write its own Task/OsIsr stack from non-trusted OS app.	Access allowed	
t1_app_nontrusted1	EnableAllInterrupts()		
t1_app_nontrusted1	Read Task/OsIsr stack in the same non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	10a
t1_app_nontrusted1	Write Task/OsIsr stack in the same non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	10b
t1_app_nontrusted1	Read Task/OsIsr stack in the other non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	10c
t1_app_nontrusted1	Write Task/OsIsr stack in the other non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	10d
t1_app_nontrusted1	Read Task/OsIsr stack in trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	10e
t1_app_nontrusted1	Write Task/OsIsr stack in trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	10f
t1_app_nontrusted1	Read OS stack from non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	3a
t1_app_nontrusted1	Write OS stack from non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	3b
t1_app_nontrusted1	WaitEvent(Event1)	E_OK	
t1_app_trusted1	DisableAllInterrupts()		
t1_app_trusted1	Read/Write its own Task/OsIsr stack from trusted OS app.	Access allowed	
t1_app_trusted1	EnableAllInterrupts()		
t1_app_trusted1	DisableAllInterrupts()		
t1_app_trusted1	Read Task/OsIsr stack in the same trusted OS application	Access allowed	11a
t1_app_trusted1	Write Task/OsIsr stack in the same trusted OS application	Access allowed	11b
t1_app_trusted1	Read Task/OsIsr stack in the other trusted OS application	Access allowed	11e
t1_app_trusted1	Write Task/OsIsr stack in the other trusted OS application	Access allowed	11f

Running task	Called OS service	Return Status	Test case
t1_app_trusted1	Read Task/OsIsr stack in non-trusted OS application	Access allowed	11c
t1_app_trusted1	Write Task/OsIsr stack in non-trusted OS application	Access allowed	11d
t1_app_trusted1	EnableAllInterrupts()		11
t1_app_trusted1	DisableAllInterrupts()		
t1_app_trusted1	Read OS stack from truted OS application	Access allowed	4a
t1_app_trusted1	Write OS stack from truted OS application	Access allowed	4b
t1_app_trusted1	EnableAllInterrupts()		
t1_app_trusted1	SetEvent(t1_app_nontrusted1, Event1)	E_OK	
t1_app_trusted1	TerminateTask()	E_OK	
t1_app_nontrusted1	TerminateTask()	E_OK	

Test Sequence 4 :

```
TEST CASES:            12, 13, 14, 16, 15, 17, 18
RETURN STATUS:         EXTENDED
SCHEDULING POLICY:     NON-PREEMPTIVE
HOOKS:                 ProtectionHook

  TASK t1_app_nontrusted1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON;
      EVENT = Event1;
};
TASK t1_app_trusted1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
      ACCESSING_APPLICATION = app_nontrusted1;
};
APPLICATION app_nontrusted1 {
      TRUSTED = FALSE;
  TASK = t1_app_nontrusted1;
};
APPLICATION app_trusted1 {
      TRUSTED = TRUE;
  TASK = t1_app_trusted1;
};
  EVENT Event1 { MASK = AUTO; };
```

Running task	Called OS service	Return Status	Test case
t1_app_nontrusted1	Execute protected code from non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	(not tested yet) 14
t1_app_nontrusted1	Execute shared library code from non-trusted OS application	Access allowed	12
t1_app_nontrusted1	Read access to peripheral from non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	17c
t1_app_nontrusted1	Write access to peripheral from non-trusted OS application	Call ProtectionHook with E_OS_PROTECTION_MEMORY	17d
t1_app_nontrusted1	Read access to its assigned peripheral from non-trusted OS application	Access allowed	(not tested yet) 16a
t1_app_nontrusted1	Write access to its assigned peripheral from non-trusted OS application	Access allowed	(not tested yet) 16b
t1_app_nontrusted1	WaitEvent(Event1)	E_OK	
t1_app_trusted1	Execute protected code from trusted OS application	Access allowed	(not tested yet) 15
t1_app_trusted1	Execute shared library code from trusted OS application	Access allowed	13
t1_app_trusted1	Read access to peripheral from trusted OS application	Access allowed	18c
t1_app_trusted1	Write access to peripheral from trusted OS application	Access allowed	18d
t1_app_trusted1	Read access to its assigned peripheral from trusted OS application	Access allowed	(not tested yet) 18a
t1_app_trusted1	Write access to its assigned peripheral from trusted OS application	Access allowed	(not tested yet) 18b

Running task	Called OS service	Return Status	Test case
t1_app_trusted1	SetEvent(t1_app_nontrusted1, Event1)	E_OK	
t1_app_trusted1	TerminateTask()	E_OK	
t1_app_nontrusted1	TerminateTask()	E_OK	

2.15 AUTOSAR - Timing Protection

2.15.1 Time Execution Budget

Test Sequence 1 :

```
  TEST CASES:             1, 2, 3, 4
  RETURN STATUS:          EXTENDED
  SCHEDULING POLICY:      PREEMPTIVE
  HOOKS:                  ProtectionHook

TASK t1 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  TIMING_PROTECTION = TRUE/* {
    EXECUTIONBUDGET = 10000;  long enough to let the task execute itself plus
        t2 (to test a task preempted and finishing before the Execution budget)
    TIMEFRAME = 1;
    MAXOSINTERRUPTLOCKTIME = 1;
    MAXALLINTERRUPTLOCKTIME = 1;
  }*/;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t3 {
  AUTOSTART = FALSE;
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  TIMING_PROTECTION = TRUE /*{
    EXECUTIONBUDGET = 10000;
    TIMEFRAME = 1;
    MAXOSINTERRUPTLOCKTIME = 1;
    MAXALLINTERRUPTLOCKTIME = 1;
  }*/;
};
TASK t4 {
  AUTOSTART = FALSE;
```

```
 PRIORITY = 1;
 ACTIVATION = 1;
 SCHEDULE = FULL;
};
COUNTER Hardware_Counter{
 MAXALLOWEDVALUE = 16;
 TICKSPERBASE = 10;
 MINCYCLE = 1;
 TYPE = HARDWARE {};
};
SCHEDULETABLE sched1 {
 COUNTER = Hardware_Counter;
 AUTOSTART = RELATIVE { OFFSET = 1; APPMODE = std; };
 LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
 PERIODIC = TRUE;
 LENGTH = 5;
 EXPIRY_POINT exp1 {
  OFFSET = 0;
  ACTION = ACTIVATETASK {
   TASK = t1;
  };
 };
};
```

Running task	Called OS service	Return Status	Test case
idle	*Wait for the alarm*		
t1	TerminateTask()	E_OK	1
idle	*Wait for the alarm*		
t1	ActivateTask(t2)	E_OK	
t2	TerminateTask()	E_OK	
t1	TerminateTask()	E_OK	3
idle	*Wait for the alarm*		
t1	StopScheduleTable(sched1)	E_OK	
t1	ActivateTask(t3)	E_OK	
t3	*endless loop to active protection hook*		2
ProtectionHook	Fatalerror	E_OS_PROTECTION_TIME	
ProtectionHook	*return PRO_TERMINATETASKISR*		
t1	ActivateTask(t4)	E_OK	
t1	*endless loop to active protection hook*		4
ProtectionHook	Fatalerror	E_OS_PROTECTION_TIME	
ProtectionHook	*return PRO_TERMINATETASKISR*		
t4	TerminateTask()	E_OK	

Test Sequence 2 :

```
TEST CASES:              5, 7, 8, 9, 10
RETURN STATUS:           EXTENDED
SCHEDULING POLICY:       PREEMPTIVE
```

```
 HOOKS:                    ProtectionHook

TASK t1 {
 AUTOSTART = TRUE { APPMODE = std; };
 PRIORITY = 2;
 ACTIVATION = 1;
 SCHEDULE = FULL;
 TIMING_PROTECTION = TRUE {
   EXECUTIONBUDGET = 10000; /* long enough to let the task execute itself
       plus t2 (to test a task preempted and finishing before the Execution
       budget) */
   TIMEFRAME = 1;
   MAXOSINTERRUPTLOCKTIME = 1;
   MAXALLINTERRUPTLOCKTIME = 1;
 };
 EVENT = t1_event1;
};
TASK t2 {
 AUTOSTART = FALSE;
 PRIORITY = 3;
 ACTIVATION = 1;
 SCHEDULE = FULL;
};
TASK t3 {
 AUTOSTART = TRUE { APPMODE = std; };
 PRIORITY = 3;
 ACTIVATION = 1;
 SCHEDULE = FULL;
 TIMING_PROTECTION = TRUE {
   EXECUTIONBUDGET = 10000;
   TIMEFRAME = 1;
   MAXOSINTERRUPTLOCKTIME = 1;
   MAXALLINTERRUPTLOCKTIME = 1;
 };
 EVENT = t3_event1;
};
TASK t4 {
 AUTOSTART = FALSE;
 PRIORITY = 1;
 ACTIVATION = 1;
 SCHEDULE = FULL;
};
COUNTER Hardware_Counter{
 MAXALLOWEDVALUE = 16;
 TICKSPERBASE = 10;
 MINCYCLE = 1;
 TYPE = HARDWARE {};
};
SCHEDULETABLE sched1 {
 COUNTER = Hardware_Counter;
```

```
 AUTOSTART = RELATIVE { OFFSET = 1; APPMODE = std; };
 LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
 PERIODIC = TRUE;
 LENGTH = 5;
 EXPIRY_POINT exp1 {
  OFFSET = 0;
  ACTION = SETEVENT {
   TASK = t1;
   EVENT = t1_event1;
  };
 };
};
EVENT t1_event1{ MASK = AUTO;};
EVENT t3_event1{ MASK = AUTO;};
```

Running task	Called OS service	Return Status	Test case
t3	WaitEvent(t1_event1)	E_OK	5
t1	WaitEvent(t1_event1)	E_OK	
idle	*Wait for the alarm*		
t1	ClearEvent(t1_event1)	E_OK	7
t1	WaitEvent(t1_event1)	E_OK	
idle	*Wait for the alarm*		
t1	ClearEvent(t1_event1)	E_OK	
t1	ActivateTask(t2)	E_OK	
t2	TerminateTask()	E_OK	9
t1	WaitEvent(t1_event1)	E_OK	
idle	*Wait for the alarm*		
t1	ClearEvent(t1_event1)	E_OK	
t1	StopScheduleTable(sched1)	E_OK	
t1	SetEvent(t3,t3_event1)	E_OK	
t3	ClearEvent(t3 _event1)	E_OK	
t3	*endless loop to active protection hook*		8
ProtectionHook	Fatalerror	E_OS_PROTECTION_TIME	
ProtectionHook	*return PRO_TERMINATETASKISR*		
t1	ActivateTask(t4)	E_OK	
t1	*endless loop to active protection hook*	E_OS_PROTECTION_TIME	10
ProtectionHook	*return PRO_TERMINATETASKISR*		
t4	TerminateTask()	E_OK	

Test Sequence 3 :

```
 TEST CASES:            6, 11, 12
 RETURN STATUS:         EXTENDED
 SCHEDULING POLICY:     PREEMPTIVE
 HOOKS:                 ProtectionHook

TASK t1 {
 AUTOSTART = TRUE{ APPMODE = std; };
```

```
  PRIORITY = 3;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  TIMING_PROTECTION = TRUE {
    EXECUTIONBUDGET = 10000; /* long enough to let the task execute itself
        plus t2 (to test a task preempted and finishing before the Execution
        budget) */
    TIMEFRAME = 1;
    MAXOSINTERRUPTLOCKTIME = 1;
    MAXALLINTERRUPTLOCKTIME = 1;
  };
  EVENT = t1_event1;
};
TASK t2 {
  AUTOSTART = TRUE{ APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t3 {
  AUTOSTART = TRUE{ APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
EVENT t1_event1{ MASK = AUTO;};
```

Running task	Called OS service	Return Status	Test case
t1	SetEvent(t1,t1_event1)	E_OK	
t1	WaitEvent(t1_event1)	E_OK	
t1	ClearEvent(t1_event1)	E_OK	
t1	SetEvent(t1,t1_event1)	E_OK	
t1	WaitEvent(t1_event1)	E_OK	
t1	ClearEvent(t1_event1)	E_OK	
t1	SetEvent(t1,t1_event1)	E_OK	
t1	WaitEvent(t1_event1)	E_OK	11
t1	CleanEvent(t1_event1)	E_OK	
t1	SetEvent(t1,t1_event1)	E_OK	
t1	*endless loop to active protection hook*		12
ProtectionHook	Fatalerror	E_OS_PROTECTION_TIME	
ProtectionHook	*return PRO_TERMINATETASKISR*		
t2	*endless loop to active protection hook*		6
ProtectionHook	Fatalerror	E_OS_PROTECTION_TIME	
ProtectionHook	*return PRO_TERMINATETASKISR*		

Running task	Called OS service	Return Status	Test case
t3	TerminateTask1()	E_OK	

Test Sequence 4 :

```
TEST CASES:             13, 14, 15, 16
RETURN STATUS:          EXTENDED
SCHEDULING POLICY:      PREEMPTIVE
HOOKS:                  ProtectionHook

TASK t1 {
 AUTOSTART = TRUE { APPMODE = std; };
 PRIORITY = 2;
 ACTIVATION = 1;
 SCHEDULE = FULL;
};
ISR isr1 {
  CATEGORY = 2;
  PRIORITY = 1;
  SOURCE = SIGTERM;
  TIMING_PROTECTION = TRUE {
    TIMEFRAME = 1;
    EXECUTIONTIME = 1000;
    MAXOSINTERRUPTLOCKTIME = 1;
    MAXALLINTERRUPTLOCKTIME = 1;
  };
};
ISR isr2 {
  CATEGORY = 2;
  PRIORITY = 2;
  SOURCE = SIGUSR2;
  TIMING_PROTECTION = TRUE {
    TIMEFRAME = 1;
    EXECUTIONTIME = 1000;
    MAXOSINTERRUPTLOCKTIME = 1;
    MAXALLINTERRUPTLOCKTIME = 1;
  };
};
```

Running task	Called OS service	Return Status	Test case
t1	*trigger interrupt isr1*		
isr1			13
t1	*trigger interrupt isr1*		
isr1	*trigger interrupt isr2*		
isr2			
isr1			15

Running task	Called OS service	Return Status	Test case
t1	*trigger interrupt isr1*		
isr1	*trigger interrupt isr2*		
isr2	*endless loop to active protection hook*		14
ProtectionHook	Fatalerror	E_OS_PROTECTION_TIME	
ProtectionHook	*return PRO_TERMINATETASKISR*		
isr1	*endless loop to active protection hook*		16
ProtectionHook	Fatalerror	E_OS_PROTECTION_TIME	
ProtectionHook	*return PRO_TERMINATETASKISR*		
t1	TerminateTask()	E_OK	

2.15.2 Time Frame

Test Sequence 5 :

```
  TEST CASES:             1, 2
  RETURN STATUS:          EXTENDED
  SCHEDULING POLICY:      PREEMPTIVE
  HOOKS:                  ProtectionHook

TASK t1 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  TIMING_PROTECTION = TRUE {
    EXECUTIONBUDGET = 1;
    TIMEFRAME = 1; /* inferior than alarm first period (20*10ms) and superior
        than alarm second period (5*10ms) */
    MAXOSINTERRUPTLOCKTIME = 1;
    MAXALLINTERRUPTLOCKTIME = 1;
  };
};
COUNTER Hardware_Counter{
  MAXALLOWEDVALUE = 21;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = HARDWARE {};
};
ALARM alarm1 {
  COUNTER = Hardware_Counter;
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = TRUE {
    ALARMTIME = 1;
    CYCLETIME = 20;
    APPMODE = std;
```

```
  };
};
```

Running task	Called OS service	Return Status	Test case
idle	*Wait for the alarm*		
t1	TerminateTask()	E_OK	
idle	*Wait for the alarm*		1
t1	CancelAlarm(alarm1)	E_OK	
t1	SetRelAlarm(alarm1, 5, 5)	E_OK	
t1	TerminateTask()	E_OK	
idle	*Wait for the alarm*		2
ProtectionHook	Fatalerror	E_OS_PROTECTION_ARRIVAL	
ProtectionHook	*return PRO_TERMINATETASKISR*		
idle	*Wait for the alarm*		
t1	TerminateTask()	E_OK	

Test Sequence 6 :

```
 TEST CASES:              3
 RETURN STATUS:           EXTENDED
 SCHEDULING POLICY:       PREEMPTIVE
 HOOKS:                   ProtectionHook, ErrorHook

TASK t1 {
 AUTOSTART = FALSE;
 PRIORITY = 1;
 ACTIVATION = 2;
 SCHEDULE = FULL;
 TIMING_PROTECTION = TRUE {
   EXECUTIONBUDGET = 1;
   TIMEFRAME = 100; /* superior than alarm period (20*10ms) */
   MAXOSINTERRUPTLOCKTIME = 1;
   MAXALLINTERRUPTLOCKTIME = 1;
 };
};
COUNTER Hardware_Counter{
 MAXALLOWEDVALUE = 21;
 TICKSPERBASE = 1;
 MINCYCLE = 1;
 TYPE = HARDWARE {};
};
ALARM alarm1 {
 COUNTER = Hardware_Counter;
 ACTION = ACTIVATETASK {
   TASK = t1;
 };
 AUTOSTART = TRUE {
   ALARMTIME = 1;
   CYCLETIME = 20;
```

```
    APPMODE = std;
  };
};
```

Running task	Called OS service	Return Status	Test case
t1	*Wait for ever (wait for the alarm)*		
ProtectionHook	Fatalerror	E_OS_PROTECTION_ARRIVAL	
ProtectionHook	*return PRO_TERMINATETASKISR*		
t1	*Continue to wait for ever (wait for the alarm)*		3
ErrorHook	error	E_OS_LIMIT	
ErrorHook	OSErrorGetServiceId()	OSServiceId_ActivateTask	

Test Sequence 7 :

```
TEST CASES:             4, 5, 6
RETURN STATUS:          EXTENDED
SCHEDULING POLICY:      PREEMPTIVE
HOOKS:                  ProtectionHook

TASK t1 {
 AUTOSTART = TRUE { APPMODE = std; };
 PRIORITY = 2;
 ACTIVATION = 1;
 SCHEDULE = FULL;
 TIMING_PROTECTION = TRUE {
    EXECUTIONBUDGET = 1;
    TIMEFRAME = 100;
    MAXOSINTERRUPTLOCKTIME = 1;
    MAXALLINTERRUPTLOCKTIME = 1;
  };
 EVENT = t1_event1;
};
TASK t2 {
 AUTOSTART = TRUE { APPMODE = std; };
 PRIORITY = 1;
 ACTIVATION = 1;
 SCHEDULE = FULL;
 TIMING_PROTECTION = TRUE {
    EXECUTIONBUDGET = 1;
    TIMEFRAME = 100;
    MAXOSINTERRUPTLOCKTIME = 1;
    MAXALLINTERRUPTLOCKTIME = 1;
  };
 EVENT = t2_event1;
};
TASK t3 {
 AUTOSTART = FALSE;
 PRIORITY = 3;
 ACTIVATION = 1;
```

```
 SCHEDULE = FULL;
 TIMING_PROTECTION = TRUE {
   EXECUTIONBUDGET = 1;
   TIMEFRAME = 100;
   MAXOSINTERRUPTLOCKTIME = 1;
   MAXALLINTERRUPTLOCKTIME = 1;
 };
 EVENT = t3_event1;
};
EVENT t1_event1{ MASK = AUTO;};
EVENT t2_event1{ MASK = AUTO;};
EVENT t3_event1{ MASK = AUTO;};
```

Running task	Called OS service	Return Status	Test case
t1	*Wait Time Frame elapsed*		
t1	SetEvent(t1,t1_event1)	E_OK	
t1	WaitEvent(t1_event1)	E_OK	4
t1	TerminateTask()	E_OK	
t2	SetEvent(t2,t2_event1)	E_OK	
t2	WaitEvent(t2_event1)	E_OK	5
t2	ActivateTask(t3)	E_OK	
t3	SetEvent(t3,t3_event1)	E_OK	
t3	WaitEvent(t3_event1)	E_OK	6
ProtectionHook	Fatalerror	E_OS_PROTECTION_ARRIVAL	
ProtectionHook	*return PRO_TERMINATETASKISR*		
t2	TerminateTask()	E_OK	

Test Sequence 8 :

```
 TEST CASES:              7, 8
 RETURN STATUS:           EXTENDED
 SCHEDULING POLICY:       PREEMPTIVE
 HOOKS:                   ProtectionHook

TASK t1 {
 AUTOSTART = TRUE { APPMODE = std; };
 PRIORITY = 2;
 ACTIVATION = 1;
 SCHEDULE = FULL;
 TIMING_PROTECTION = TRUE {
   EXECUTIONBUDGET = 1;
   TIMEFRAME = 100; /* inferior than alarm first period (20*10ms) and
        superior than alarm second period (5*10ms) */
   MAXOSINTERRUPTLOCKTIME = 1;
   MAXALLINTERRUPTLOCKTIME = 1;
 };
 EVENT = t1_event1;
};
COUNTER Hardware_Counter{
```

```
 MAXALLOWEDVALUE = 21;
 TICKSPERBASE = 1;
 MINCYCLE = 1;
 TYPE = HARDWARE {};
};
ALARM alarm1 {
 COUNTER = Hardware_Counter;
 ACTION = SETEVENT {
    TASK = t1;
    EVENT = t1_event1;
  };
 AUTOSTART = TRUE {
    ALARMTIME = 1;
    CYCLETIME = 20;
    APPMODE = std;
  };
};
EVENT t1_event1{ MASK = AUTO;};
```

Running task	Called OS service	Return Status	Test case
t1	*Wait Time Frame elapsed*		
t1	WaitEvent(t1_event1)	E_OK	
idle	*Wait for the alarm if has not set t1_event1 to t1 during previous Time Frame wait*		
t1	ClearEvent(t1_event1)	E_OK	
t1	WaitEvent(t1_event1)	E_OK	
idle	*Wait for the alarm*		7
t1	ClearEvent(t1_event1)	E_OK	
t1	CancelAlarm(alarm1)	E_OK	
t1	SetRelAlarm(alarm1, 5, 5)	E_OK	
t1	WaitEvent(t1_event1)	E_OK	
idle	*Wait for the alarm*		8
ProtectionHook	Fatalerror	E_OS_PROTECTION_ARRIVAL	
ProtectionHook	*return PRO_SHUTDOWN*		

Test Sequence 9 :

```
 TEST CASES:            9, 10, 11, 12
 RETURN STATUS:         EXTENDED
 SCHEDULING POLICY:     PREEMPTIVE
 HOOKS:                 ProtectionHook

TASK t1 {
 AUTOSTART = TRUE { APPMODE = std; };
 PRIORITY = 2;
 ACTIVATION = 1;
 SCHEDULE = FULL;
};
ISR isr1 {
```

```
  CATEGORY = 2;
  PRIORITY = 2;
  SOURCE = SIGTERM;
  TIMING_PROTECTION = TRUE {
    TIMEFRAME = 1;
    EXECUTIONTIME = 1000;
    MAXOSINTERRUPTLOCKTIME = 1;
    MAXALLINTERRUPTLOCKTIME = 1;
  };
};
ISR isr2 {
  CATEGORY = 2;
  PRIORITY = 1;
  SOURCE = SIGUSR2;
  TIMING_PROTECTION = TRUE {
    TIMEFRAME = 1;
    EXECUTIONTIME = 1000;
    MAXOSINTERRUPTLOCKTIME = 1;
    MAXALLINTERRUPTLOCKTIME = 1;
  };
};
```

Running task	Called OS service	Return Status	Test case
t1	*trigger interrupt isr1*		
isr1			
t1	*Wait isr1 Time Frame elapsed*		
t1	*trigger interrupt isr1*		9
isr1			
t1	*trigger interrupt isr1*		11
ProtectionHook	Fatalerror	E_OS_PROTECTION_ARRIVAL	
ProtectionHook	*return PRO_TERMINATETASKISR*		
t1	*Wait isr1 Time Frame elapsed*		
t1	*trigger interrupt isr1*		
isr1	*trigger interrupt isr2*		
isr1	*trigger interrupt isr1*		12
isr1	*Wait isr2 Time Frame elapsed*		
isr2			
t1	*trigger interrupt isr2*		10
isr2			
t1	TerminateTask()	E_OK	

2.15.3 Resource Locking and Interrupt Disabling

3 GOIL Test sequences

This chapter contains the specification of the test sequences that are not allowed for the user. The application in the oil file contains errors and GOIL has to prevent the user of the error (for example, an alarm which send an event to a basic task is forbidden). Each test sequence fulfils the test for one ore more of the test cases defined in Trampoline Test Plan.

3.1 Event mechanism

Test Sequence 1 :

```
TEST CASES:            41, 42, 43

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = NON;
  EVENT = Task1_Event1;
  EVENT = Task1_Event2;
  EVENT = Task1_Event3;
  EVENT = Task1_Event4;
  EVENT = Task1_Event5;
  EVENT = Task1_Event6;
  EVENT = Task1_Event7;
  EVENT = Task1_Event8;
  EVENT = Task1_Event9;
  EVENT = Task1_Event10;
  EVENT = Task1_Event11;
  EVENT = Task1_Event12;
  EVENT = Task1_Event13;
  EVENT = Task1_Event14;
  EVENT = Task1_Event15;
  EVENT = Task1_Event16;
  EVENT = Task1_Event17;
  EVENT = Task1_Event18;
  EVENT = Task1_Event19;
  EVENT = Task1_Event20;
  EVENT = Task1_Event21;
  EVENT = Task1_Event22;
  EVENT = Task1_Event23;
  EVENT = Task1_Event24;
  EVENT = Task1_Event25;
  EVENT = Task1_Event26;
  EVENT = Task1_Event27;
  EVENT = Task1_Event28;
  EVENT = Task1_Event29;
  EVENT = Task1_Event30;
  EVENT = Task1_Event31;
```

```
  EVENT = Task1_Event32;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = NON;
  EVENT = Task2_Event1;
  EVENT = Task2_Event2;
  EVENT = Task2_Event3;
  EVENT = Task2_Event4;
  EVENT = Task2_Event5;
  EVENT = Task2_Event6;
  EVENT = Task2_Event7;
  EVENT = Task2_Event8;
};
EVENT Task1_Event1 { MASK=AUTO; };
EVENT Task1_Event2 { MASK=AUTO; };
EVENT Task1_Event3 { MASK=16; };
EVENT Task1_Event4 { MASK=AUTO; };
EVENT Task1_Event5 { MASK=AUTO; };
EVENT Task1_Event6 { MASK=AUTO; };
EVENT Task1_Event7 { MASK=4; };
EVENT Task1_Event8 { MASK=3; };
EVENT Task1_Event9 { MASK=AUTO; };
EVENT Task1_Event10 { MASK=8; };
EVENT Task1_Event11 { MASK=AUTO; };
EVENT Task1_Event12 { MASK=AUTO; };
EVENT Task1_Event13 { MASK=AUTO; };
EVENT Task1_Event14 { MASK=AUTO; };
EVENT Task1_Event15 { MASK=AUTO; };
EVENT Task1_Event16 { MASK=128; };
EVENT Task1_Event17 { MASK=AUTO; };
EVENT Task1_Event18 { MASK=AUTO; };
EVENT Task1_Event19 { MASK=AUTO; };
EVENT Task1_Event20 { MASK=AUTO; };
EVENT Task1_Event21 { MASK=AUTO; };
EVENT Task1_Event22 { MASK=AUTO; };
EVENT Task1_Event23 { MASK=1024; };
EVENT Task1_Event24 { MASK=AUTO; };
EVENT Task1_Event25 { MASK=AUTO; };
EVENT Task1_Event26 { MASK=AUTO; };
EVENT Task1_Event27 { MASK=AUTO; };
EVENT Task1_Event28 { MASK=AUTO; };
EVENT Task1_Event29 { MASK=AUTO; };
EVENT Task1_Event30 { MASK=AUTO; };
EVENT Task1_Event31 { MASK=AUTO; };
EVENT Task1_Event32 { MASK=AUTO; };

EVENT Task2_Event1 { MASK=4; };
```

```
EVENT Task2_Event2 { MASK=4; };
EVENT Task2_Event3 { MASK=AUTO; };
EVENT Task2_Event4 { MASK=AUTO; };
EVENT Task2_Event5 { MASK=AUTO; };
EVENT Task2_Event6 { MASK=AUTO; };
EVENT Task2_Event7 { MASK=AUTO; };
EVENT Task2_Event8 { MASK=AUTO; };
```

Wrong application	Return Status	Test case
Creating an event with a MASK using more than one bit (*Task1_Event8*)	Warning : Mask attribute uses more than one bit	41
Creating the event *Task2_Event2* with a MASK already used	Error : MASK of event *Task2_Event2* conflicts with previous declarations	42
Creating an event with an automatic MASKs but all the MASK are already used	Error : All event mask bits are already use, event *Task1_Event9* can't be created	43

3.2 AUTOSAR - Alarm

See diagram from Trampoline Test Plan.

Test Sequence 1 :

```
TEST CASES:            3, 5, 7

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 2;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Hardware_Counter{
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = HARDWARE {};
};
COUNTER Software_Counter{
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
```

```
ALARM Alarm_SetEvent_basictask{
  COUNTER = Software_Counter;
  ACTION = SETEVENT {
    TASK = t1;
    EVENT = Event1;
  };
  AUTOSTART = FALSE;
};
ALARM Alarm_IncrementCounter{
  COUNTER = Software_Counter;
  ACTION = INCREMENTCOUNTER {
    COUNTER = Hardware_Counter;
  };
  AUTOSTART = FALSE;
};
APPLICATION app {
  TASK = t1;
  TASK = t2;
  COUNTER = Software_Counter;
  COUNTER = Hardware_Counter;
  ALARM = Alarm_SetEvent_basictask;
  ALARM = Alarm_IncrementCounter;
};
EVENT Event1 {
  MASK = AUTO;
};
```

Wrong application	Return Status	Test case
Alarm's SetEvent action set an event on a basic task	error : An alarm can't set an Event to a basic task (Task t1 is a basic task).	3
Alarm's IncrementCounter, increment an hardware counter	error : OS285 - It is impossible to increment a hardware counter (*Hardware_Counter* is not a software counter).	5, 7

3.3 AUTOSAR - Schedule Table

Test Sequence 1 :

```
TEST CASES:          42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std; };
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter{
  MAXALLOWEDVALUE = 10;
```

```
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
COUNTER Software_Counter_bis{
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched1 {
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  LENGTH = 10;
};
SCHEDULETABLE sched2 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  PERIODIC = TRUE;
  LENGTH = 10;
  EXPIRY_POINT sched2_ep1 {
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
SCHEDULETABLE sched3 {
  COUNTER = Software_Counter;
  COUNTER = Software_Counter_bis;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 10;
  EXPIRY_POINT sched3_ep1 {
    OFFSET = 2;
  };
};
SCHEDULETABLE sched4 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = TRUE;
  LENGTH = 10;
  EXPIRY_POINT sched4_ep1 {
    OFFSET = 2;
    OFFSET = 4;
    ACTION = ACTIVATETASK {
      TASK = t1;
```

```
    };
  };
  EXPIRY_POINT sched4_ep2 {
    OFFSET = 6;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
    ACTION = SETEVENT {
      TASK = t1;
      EVENT = Event1;
    };
  };
};
SCHEDULETABLE sched5 {
  COUNTER = Software_Counter;
  AUTOSTART = NONE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = TRUE;
  LENGTH = 10;
  EXPIRY_POINT sched5_ep1 {
    OFFSET = 4;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
  EXPIRY_POINT sched5_ep2 {
    OFFSET = 6;
    ACTION = SETEVENT {
      TASK = t1;
      EVENT = Event1;
    };
  };
};
EVENT Event1 {
  MASK = AUTO;
};
APPLICATION app {
  TASK = t1;
  COUNTER = Software_Counter;
  COUNTER = Software_Counter_bis;
  SCHEDULETABLE = sched1;
  SCHEDULETABLE = sched2;
  SCHEDULETABLE = sched3;
  SCHEDULETABLE = sched4;
  SCHEDULETABLE = sched5;
};
```

Wrong application	Return Status	Test case
No Expiry point in schedule table *sched1*	error : OS401 - no EXPIRY_POINT found for SCHEDULETABLE *sched1*	42
One expiry point in schedule table *sched3*		43
Two expiry points in schedule table *sched4*		43
No Action in expiry point *sched3_ep1*	error : OS407 - no ACTION found for EXPIRY_POINT *sched3_ep1*	44
One Action in expiry point *sched5_ep1*		45
Two Actions in expiry point *sched4_ep2*		46
No Counter in schedule table *sched1*	error : OS409 - Counter is not defined in *sched1* !	47
One Counter in schedule table *sched2*		48
Multiple Counters in schedule table *sched3*	error : OS409 - COUNTER attribute already defined for Schedule Table *sched3*	49
No Offset in expiry point *sched2_ep1*	error : OS404 - OFFSET is missing for expiry point *sched2_ep1*	50
One Offset in expiry point *sched4_ep2*		51
Multiple Offset in expiry point *sched4_ep1*	error : OS442 - OFFSET Redefinition	52

Test Sequence 2 :

```
TEST CASES:          53, 54, 55, 56, 57, 58, 59, 60, 62, 63, 64

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter{
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 4;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched1 {
  COUNTER = Software_Counter;
  AUTOSTART = FALSE ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
  PERIODIC = FALSE;
  LENGTH = 10;
  EXPIRY_POINT sched1_ep1 {
    OFFSET = 0;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
```

```
};
SCHEDULETABLE sched2 {
  COUNTER = Software_Counter;
  AUTOSTART = FALSE ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
  PERIODIC = FALSE;
  LENGTH = 6;
  EXPIRY_POINT sched2_ep1 {
    OFFSET = 3;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
  EXPIRY_POINT sched2_ep2 {
    OFFSET = 6;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
SCHEDULETABLE sched3 {
  COUNTER = Software_Counter;
  AUTOSTART = FALSE ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
  PERIODIC = FALSE;
  LENGTH = 11;
  EXPIRY_POINT sched3_ep1 {
    OFFSET = 4;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
  EXPIRY_POINT sched3_ep2 {
    OFFSET = 8;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
SCHEDULETABLE sched4 {
  COUNTER = Software_Counter;
  AUTOSTART = FALSE ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
  PERIODIC = FALSE;
  LENGTH = 24;
  EXPIRY_POINT sched4_ep1 {
    OFFSET = 10;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
```

```
  };
  EXPIRY_POINT sched4_ep2 {
    OFFSET = 20;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
SCHEDULETABLE sched5 {
  COUNTER = Software_Counter;
  AUTOSTART = FALSE ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
  PERIODIC = FALSE;
  LENGTH = 33;
  EXPIRY_POINT sched5_ep1 {
    OFFSET = 11;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
  EXPIRY_POINT sched5_ep2 {
    OFFSET = 22;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
EVENT Event1{
  MASK = AUTO;
};
APPLICATION app {
  TASK = t1;
  COUNTER = Software_Counter;
  SCHEDULETABLE = sched1;
  SCHEDULETABLE = sched2;
  SCHEDULETABLE = sched3;
  SCHEDULETABLE = sched4;
  SCHEDULETABLE = sched5;
};
```

Wrong application	Return Status	Test case
sched1 First Delay is equal to 0		53
sched2 First Delay is lower than MINCYCLE	error : OS443 - OFFSET of first expiry point is lower than MINCYCLE of the driving counter and not equal to 0.	54

Wrong application	Return Status	Test case
sched3 First Delay is equal to MINCYCLE		55
sched4 First Delay is equal to MAXALLOWEDVALUE		55
sched5 First Delay is greater than MAXALLOWEDVALUE	error : OS443 - OFFSET of first expiry point is greater than MAXALLOWEDVALUE of the driving counter	56
Delay bewteen *sched2_ep1* and *sched2_ep2* is lower than MINCYCLE	error : OS408 - Delay between expiry point number 1 and 2 is lower than MINCYCLE of the driving counter	57
Delay bewteen *sched3_ep1* and *sched3_ep2* is equal to MINCYCLE		58
Delay bewteen *sched4_ep1* and *sched4_ep2* is equal to MAXALLOWEVALUE		58
Delay bewteen *sched5_ep1* and *sched5_ep2* is greater than MAXALLOWEDVALUE	error : OS408 - Delay between expiry point number 1 and 2 is greater than MAXALLOWEDVALUE of the driving counter	59
(Single-shot) *sched2* Final Delay is equal to 0		60
(Single-shot) *sched3* Final Delay is lower than MINCYCLE	error : OS444 - Final delay should be within MINCYCLE and MAXALLOWEDVALUE of the driving counter	62
(Single-shot) *sched4* Final Delay is equal to MINCYCLE		63
(Single-shot) *sched1* Final Delay is equal to MAXALLOWEDVALUE		63
(Single-shot) *sched5* Final Delay is greater than MAXALLOWEDVALUE	error : OS444 - Final delay should be within MINCYCLE and MAXALLOWEDVALUE of the driving counter	64

Test Sequence 3 :

```
TEST CASES:          53, 54, 55, 56, 57, 58, 59, 61, 62, 63, 64

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter{
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 4;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched1 {
  COUNTER = Software_Counter;
```

```
    AUTOSTART = FALSE ;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
    PERIODIC = TRUE;
    LENGTH = 10;
    EXPIRY_POINT sched1_ep1 {
      OFFSET = 0;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  SCHEDULETABLE sched2 {
    COUNTER = Software_Counter;
    AUTOSTART = FALSE ;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
    PERIODIC = TRUE;
    LENGTH = 6;
    EXPIRY_POINT sched2_ep1 {
      OFFSET = 3;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
    EXPIRY_POINT sched2_ep2 {
      OFFSET = 6;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  SCHEDULETABLE sched3 {
    COUNTER = Software_Counter;
    AUTOSTART = FALSE ;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
    PERIODIC = TRUE;
    LENGTH = 11;
    EXPIRY_POINT sched3_ep1 {
      OFFSET = 4;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
    EXPIRY_POINT sched3_ep2 {
      OFFSET = 8;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  SCHEDULETABLE sched4 {
```

```
  COUNTER = Software_Counter;
  AUTOSTART = FALSE ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
  PERIODIC = TRUE;
  LENGTH = 24;
  EXPIRY_POINT sched4_ep1 {
    OFFSET = 10;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
  EXPIRY_POINT sched4_ep2 {
    OFFSET = 20;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
SCHEDULETABLE sched5 {
  COUNTER = Software_Counter;
  AUTOSTART = FALSE ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
  PERIODIC = TRUE;
  LENGTH = 33;
  EXPIRY_POINT sched5_ep1 {
    OFFSET = 11;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
  EXPIRY_POINT sched5_ep2 {
    OFFSET = 22;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
EVENT Event1{
  MASK = AUTO;
};
APPLICATION app {
  TASK = t1;
  COUNTER = Software_Counter;
  SCHEDULETABLE = sched1;
  SCHEDULETABLE = sched2;
  SCHEDULETABLE = sched3;
  SCHEDULETABLE = sched4;
  SCHEDULETABLE = sched5;
};
```

Wrong application	Return Status	Test case
sched1 First Delay is equal to 0		53
sched2 First Delay is lower than MINCYCLE	error : OS443 - OFFSET of first expiry point is lower than MINCYCLE of the driving counter and not equal to 0.	54
sched3 First Delay is equal to MINCYCLE		55
sched4 First Delay is equal to MAXALLOWEDVALUE		55
sched5 First Delay is greater than MAXALLOWEDVALUE	error : OS443 - OFFSET of first expiry point is greater than MAXALLOWEDVALUE of the driving counter	56
Delay bewteen *sched2_ep1* and *sched2_ep2* is lower than MINCYCLE	error : OS408 - Delay between expiry point number 1 and 2 is lower than MINCYCLE of the driving counter	57
Delay bewteen *sched3_ep1* and *sched3_ep2* is equal to MINCYCLE		58
Delay bewteen *sched4_ep1* and *sched4_ep2* is equal to MAXALLOWEVALUE		58
Delay bewteen *sched5_ep1* and *sched5_ep2* is greater than MAXALLOWEDVALUE	error : OS408 - Delay between expiry point number 1 and 2 is greater than MAXALLOWEDVALUE of the driving counter	59
(Repeating) *sched2* Final Delay is equal to 0	error : OS427 - Final delay can be equal to 0 only for single-shot schedule table and *sched2* is a repeating one	61
(Repeating) *sched3* Final Delay is lower than MINCYCLE	error : OS444 - Final delay should be within MINCYCLE and MAXALLOWEDVALUE of the driving counter	62
(Repeating) *sched4* Final Delay is equal to MINCYCLE		63
(Repeating) *sched1* Final Delay is equal to MAXALLOWEDVALUE		63
(Repeating) *sched5* Final Delay is greater than MAXALLOWEDVALUE	error : OS444 - Final delay should be within MINCYCLE and MAXALLOWEDVALUE of the driving counter	64

Test Sequence 4 :

```
TEST CASES:          65, 66, 67, 68, 69, 70

TASK t1 {
  AUTOSTART = FALSE ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter{
  MAXALLOWEDVALUE = 10;
```

```
  TICKSPERBASE = 1;
  MINCYCLE = 4;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched_abs_1 {
  COUNTER = Software_Counter;
  AUTOSTART = ABSOLUTE { START = 0 ; APPMODE = std ; } ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
  PERIODIC = FALSE;
  LENGTH = 10;
  EXPIRY_POINT ep {
    OFFSET = 4;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
SCHEDULETABLE sched_abs_2 {
  COUNTER = Software_Counter;
  AUTOSTART = ABSOLUTE { START = 10 ; APPMODE = std ; } ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
  PERIODIC = FALSE;
  LENGTH = 10;
  EXPIRY_POINT ep {
    OFFSET = 4;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
SCHEDULETABLE sched_abs_3 {
  COUNTER = Software_Counter;
  AUTOSTART = ABSOLUTE { START = 11 ; APPMODE = std ; } ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
  PERIODIC = FALSE;
  LENGTH = 10;
  EXPIRY_POINT ep {
    OFFSET = 4;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
SCHEDULETABLE sched_rel_1 {
  COUNTER = Software_Counter;
  AUTOSTART = RELATIVE { OFFSET = 0 ; APPMODE = std ; } ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
  PERIODIC = FALSE;
  LENGTH = 10;
  EXPIRY_POINT ep {
```

```
      OFFSET = 4;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  SCHEDULETABLE sched_rel_2 {
    COUNTER = Software_Counter;
    AUTOSTART = RELATIVE { OFFSET = 6 ; APPMODE = std ; } ;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
    PERIODIC = FALSE;
    LENGTH = 10;
    EXPIRY_POINT ep {
      OFFSET = 4;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  SCHEDULETABLE sched_rel_3 {
    COUNTER = Software_Counter;
    AUTOSTART = RELATIVE { OFFSET = 7 ; APPMODE = std ; } ;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
    PERIODIC = FALSE;
    LENGTH = 10;
    EXPIRY_POINT ep {
      OFFSET = 4;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  APPLICATION app {
    TASK = t1;
    COUNTER = Software_Counter;
    SCHEDULETABLE = sched_abs_1;
    SCHEDULETABLE = sched_abs_2;
    SCHEDULETABLE = sched_abs_3;
    SCHEDULETABLE = sched_rel_1;
    SCHEDULETABLE = sched_rel_2;
    SCHEDULETABLE = sched_rel_3;
  };
```

Wrong application	Return Status	Test case
Schedule table autostarts in ABSOLUTE mode with <OFFSET> equal to 0		65

Wrong application	Return Status	Test case
Schedule table autostarts in ABSOLUTE mode with <OFFSET> lower than MAXALLOWEDVALUE		66
Schedule table autostarts in ABSOLUTE mode with <OFFSET> greater than MAXALLOWEDVALUE	error: OS349 - *sched_abs_3* autostart's offset is greater than MAXALLOWED VALUE of the driving counter.	67
Schedule table autostarts in RELATIVE mode with <START> equal to 0	error: OS332 - *sched_rel_1* autostart's offset is equal to 0.	68
Schedule table autostarts in RELATIVE mode with <START> lower than MAXALLOWEDVALUE minus the Initial Offset		69
Schedule table autostarts in RELATIVE mode with <START> greater than MAXALLOWEDVALUE minus the Initial Offset	error: OS276 - *sched_rel_3* autostart's offset is greater than MAXALLOWEDVALUE of the driving counter minus the Initial Offset.	70

Test Sequence 5 :

```
TEST CASES:              33

TASK t1 {
  AUTOSTART = FALSE ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
COUNTER Software_Counter{
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched {
  COUNTER = Software_Counter;
  AUTOSTART = FALSE;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE ;
  PERIODIC = FALSE;
  LENGTH = 5;
  EXPIRY_POINT ep {
    OFFSET = 2;
    ACTION = SETEVENT {
      TASK = t1;
```

```
      EVENT = Event1;
    };
  };
};
EVENT Event1{
  MASK = AUTO;
};
APPLICATION app {
  TASK = t1;
  TASK = t2;
  COUNTER = Software_Counter;
  SCHEDULETABLE = sched;
};
```

Wrong application	Return Status	Test case
An action set an event on a basic task	error : An action can't set an Event to a basic task (Task T is a basic task).	33

3.4 AUTOSAR - Schedule Table Synchronization

Test Sequence 1 :

```
TEST CASES:          38, 39, 40, 41, 42, 43

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter{
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched1 {
  COUNTER = Software_Counter;
  AUTOSTART = RELATIVE { OFFSET = 2 ; APPMODE = std ; }  ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    SYNC_STRATEGY = IMPLICIT ;
  };
  PERIODIC = FALSE;
  LENGTH = 10;
  EXPIRY_POINT ep {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
```

```
  };
};
SCHEDULETABLE sched2 {
  COUNTER = Software_Counter;
  AUTOSTART = SYNCHRON { APPMODE = std ; }  ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    SYNC_STRATEGY = IMPLICIT ;
  };
  PERIODIC = TRUE;
  LENGTH = 11;
  EXPIRY_POINT ep {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
SCHEDULETABLE sched3 {
  COUNTER = Software_Counter;
  AUTOSTART = ABSOLUTE { START = 1 ; APPMODE = std ; } ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    SYNC_STRATEGY = IMPLICIT ;
  };
  PERIODIC = TRUE;
  LENGTH = 11;
  EXPIRY_POINT ep {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
  };
};
APPLICATION app {
  TASK = t1;
  COUNTER = Software_Counter;
  SCHEDULETABLE = sched1;
  SCHEDULETABLE = sched2;
  SCHEDULETABLE = sched3;
};
```

Wrong application	Return Status	Test case
IMPLICIT schedule table is single-shot	error : A synchronized schedule table shall be repeating otherwise, synchronisation can't be done.	38
IMPLICIT schedule table is repeating		39
IMPLICIT schedule table autostarts in ABSOLUTE mode		40

Wrong application	Return Status	Test case
IMPLICIT schedule table autostarts in RELATIVE mode	error : OS430 - An IMPLICIT schedule table should be started in Absolute mode only	41
IMPLICIT schedule table autostarts in SYNCHRON mode	error : OS430 - An IMPLICIT schedule table should be started in Absolute mode only	42
IMPLICIT schedule table duration is different to MAXALLOWEDVALUE + 1	error : OS429 - An IMPLICIT schedule table should have a duration equal to OSMAXALLOWEDVALUE + 1 (11) of its counter.	43

Test Sequence 2 :

```
TEST CASES:          44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57,
    58, 59, 60

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter{
  MAXALLOWEDVALUE = 8;
  TICKSPERBASE = 1;
  MINCYCLE = 2;
  TYPE = SOFTWARE;
};
SCHEDULETABLE sched1 {
  COUNTER = Software_Counter;
  AUTOSTART = ABSOLUTE { START = 2 ; APPMODE = std ; } ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
    PRECISION = 11;
    SYNC_STRATEGY = EXPLICIT ;
  };
  PERIODIC = FALSE;
  LENGTH = 9;
  EXPIRY_POINT sched1_ep1 {
    OFFSET = 2;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
    ADJUSTABLE = TRUE{
      MAX_RETARD = 2;
      MAX_ADVANCE = 7;
    };
  };
  EXPIRY_POINT sched1_ep2 {
    OFFSET = 7;
    ACTION = ACTIVATETASK {
      TASK = t1;
    };
```

```
      ADJUSTABLE = TRUE{
        MAX_RETARD = 4;
        MAX_ADVANCE = 5;
      };
    };
  };
  SCHEDULETABLE sched2 {
    COUNTER = Software_Counter;
    AUTOSTART = RELATIVE { OFFSET = 2 ; APPMODE = std ; } ;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
      PRECISION = 10;
      SYNC_STRATEGY = EXPLICIT ;
    };
    PERIODIC = TRUE;
    LENGTH = 9;
    EXPIRY_POINT sched2_ep1 {
      OFFSET = 2;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
      ADJUSTABLE = TRUE{
        MAX_RETARD = 3;
        MAX_ADVANCE = 8;
      };
    };
    EXPIRY_POINT sched2_ep2 {
      OFFSET = 7;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
      ADJUSTABLE = TRUE{
        MAX_RETARD = 3;
        MAX_ADVANCE = 4;
      };
    };
  };
  SCHEDULETABLE sched3 {
    COUNTER = Software_Counter;
    AUTOSTART = SYNCHRON { APPMODE = std ; } ;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = TRUE {
      SYNC_STRATEGY = EXPLICIT ;
    };
    PERIODIC = TRUE;
    LENGTH = 10;
    EXPIRY_POINT sched3_ep1 {
      OFFSET = 2;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
      ADJUSTABLE = TRUE{
```

```
        MAX_RETARD = 1;
        MAX_ADVANCE = 1;
      };
    };
  };
};
APPLICATION app {
  TASK = t1;
  COUNTER = Software_Counter;
  SCHEDULETABLE = sched1;
  SCHEDULETABLE = sched2;
  SCHEDULETABLE = sched3;
};
```

Wrong application	Return Status	Test case
EXPLICIT schedule table is single-shot	error : A synchronized schedule table shall be repeating otherwise, synchronisation can't be done.	44
EXPLICIT schedule table is repeating		45
EXPLICIT schedule table autostarts in ABSOLUTE mode		46
EXPLICIT schedule table autostarts in RELATIVE mode		47
EXPLICIT schedule table autostarts in SYNCHRON mode		48
EXPLICIT schedule table duration is greater than MAXALLOWEDVALUE + 1	error : OS431 - An EXPLICIT schedule table shouldn't have a duration greater than OSMAXALLOWEVALUE + 1 (9) of its counter.	49
EXPLICIT schedule table precision missing	error : PRECISION attribute is missing	50
EXPLICIT schedule table precision lower than duration		51
EXPLICIT schedule table precision greater than duration	error : OS438 - An explicit schedule table shall have a precision in the range 0 to duration.	52
In the first expiry point of an EXPLICIT schedule table, MaxRetard is lower than the maximum value allowed		53
In the first expiry point of an EXPLICIT schedule table, MaxRetard is greater than the maximum value allowed	error : OS436 - In first expiry point, MaxRetard (3) should be inferior to the previous delay (4) minus MINCYCLE of the counter (2).	54
In the first expiry point of an EXPLICIT schedule table, MaxAdvance is lower than the maximum value allowed		55
In the first expiry point of an EXPLICIT schedule table, MaxAdvance is greater than the maximum value allowed	error : OS437 - In first expiry point, MaxAdvance (8) should be inferior to duration (9) minus the first delay(2).	56

Wrong application	Return Status	Test case
In an expiry point of an EXPLICIT schedule table, MaxRetard is lower than the maximum value allowed		57
In an expiry point of an EXPLICIT schedule table, MaxRetard is greater than the maximum value allowed	error : OS436 - In expiry point at offset = 7, MaxRetard (4) should be inferior to the previous delay (5) minus MINCYCLE of the counter (2).	58
In an expiry point of an EXPLICIT schedule table, MaxAdvance is lower than the maximum value allowed		59
In an expiry point of an EXPLICIT schedule table, MaxAdvance is greater than the maximum value allowed	error : OS437 - In expiry point at offset = 7, MaxAdvance (5) should be inferior to duration (9) minus the previous delay(5).	60

3.5 AUTOSAR - OS-Application

3.5.1 API Service Calls for OS objects

Test Sequence 1 :

```
TEST CASES:            43
RETURN STATUS:         EXTENDED
SCHEDULING POLICY:     FULL_PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter{
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
APPLICATION app1 {
  TASK = t1;
};
APPLICATION app2 {
  COUNTER = Software_Counter;
};
```

Wrong application	Return Status	Test case

Wrong application	Return Status	Test case
No Task nor ISR2 in app2	error : OS445 - An application should have at least one Task OR ISR2.	43

3.5.2 Access Rights for objects from OIL file

Test Sequence 2 :

```
TEST CASES:            1 to 7
RETURN STATUS:         EXTENDED
SCHEDULING POLICY:     FULL_PREEMPTIVE

TASK t1 {
  AUTOSTART = TRUE { APPMODE = std ; } ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
TASK t2 {
  AUTOSTART = FALSE ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  EVENT = Event1;
};
TASK t3 {
  AUTOSTART = FALSE ;
  PRIORITY = 1;
  ACTIVATION = 1;
  SCHEDULE = FULL;
};
COUNTER Software_Counter1{
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
COUNTER Software_Counter2{
  MAXALLOWEDVALUE = 10;
  TICKSPERBASE = 1;
  MINCYCLE = 1;
  TYPE = SOFTWARE;
};
ALARM Alarm1{
  COUNTER = Software_Counter2;
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = FALSE;
```

```
};
ALARM Alarm1_1{
  COUNTER = Software_Counter2;
  ACTION = ACTIVATETASK {
    TASK = t1;
  };
  AUTOSTART = FALSE;
  ACCESSING_APPLICATION = app2;
};
ALARM Alarm2{
  COUNTER = Software_Counter1;
  ACTION = ACTIVATETASK {
    TASK = t2;
  };
  AUTOSTART = FALSE;
};
ALARM Alarm2_1{
  COUNTER = Software_Counter1;
  ACTION = ACTIVATETASK {
    TASK = t2;
  };
  AUTOSTART = FALSE;
  ACCESSING_APPLICATION = app2;
};
ALARM Alarm3{
  COUNTER = Software_Counter1;
  ACTION = SETEVENT {
    TASK = t2;
    EVENT = Event1;
  };
  AUTOSTART = FALSE;
};
ALARM Alarm3_1{
  COUNTER = Software_Counter1;
  ACTION = SETEVENT {
    TASK = t2;
    EVENT = Event1;
  };
  AUTOSTART = FALSE;
  ACCESSING_APPLICATION = app2;
};
ALARM Alarm4{
  COUNTER = Software_Counter1;
  ACTION = INCREMENTCOUNTER {
    COUNTER = Software_Counter2;
  };
  AUTOSTART = FALSE;
};
ALARM Alarm4_1{
  COUNTER = Software_Counter1;
```

```
    ACTION = INCREMENTCOUNTER {
      COUNTER = Software_Counter2;
    };
    AUTOSTART = FALSE;
    ACCESSING_APPLICATION = app2;
  };
  SCHEDULETABLE sched1 {
    COUNTER = Software_Counter2;
    AUTOSTART = FALSE ;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
    PERIODIC = FALSE;
    LENGTH = 3;
    EXPIRY_POINT exp1_sched1 {
      OFFSET = 0;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
  };
  SCHEDULETABLE sched1_1 {
    COUNTER = Software_Counter2;
    AUTOSTART = FALSE ;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
    PERIODIC = FALSE;
    LENGTH = 3;
    EXPIRY_POINT exp1_sched1 {
      OFFSET = 0;
      ACTION = ACTIVATETASK {
        TASK = t1;
      };
    };
    ACCESSING_APPLICATION = app2;
  };
  SCHEDULETABLE sched2 {
    COUNTER = Software_Counter1;
    AUTOSTART = FALSE ;
    LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
    PERIODIC = FALSE;
    LENGTH = 3;
    EXPIRY_POINT exp1_sched2 {
      OFFSET = 0;
      ACTION = SETEVENT {
        TASK = t2;
        EVENT = Event1;
      };
    };
    EXPIRY_POINT exp2_sched2 {
      OFFSET = 1;
      ACTION = ACTIVATETASK {
        TASK = t3;
```

```
    };
  };
};
SCHEDULETABLE sched2_1 {
  COUNTER = Software_Counter1;
  AUTOSTART = FALSE ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 3;
  EXPIRY_POINT exp1_sched2 {
    OFFSET = 0;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
  };
  EXPIRY_POINT exp2_sched2 {
    OFFSET = 1;
    ACTION = ACTIVATETASK {
      TASK = t3;
    };
  };
  ACCESSING_APPLICATION = app2;
  ACCESSING_APPLICATION = app3;
};
SCHEDULETABLE sched3 {
  COUNTER = Software_Counter1;
  AUTOSTART = FALSE ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 3;
  EXPIRY_POINT exp1_sched3 {
    OFFSET = 0;
    ACTION = SETEVENT {
      TASK = t2;
      EVENT = Event1;
    };
    ACTION = ACTIVATETASK {
      TASK = t3;
    };
  };
};
SCHEDULETABLE sched3_1 {
  COUNTER = Software_Counter1;
  AUTOSTART = FALSE ;
  LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION = FALSE;
  PERIODIC = FALSE;
  LENGTH = 3;
  EXPIRY_POINT exp1_sched3 {
    OFFSET = 0;
```

```
      ACTION = SETEVENT {
        TASK = t2;
        EVENT = Event1;
      };
      ACTION = ACTIVATETASK {
        TASK = t3;
      };
    };
    ACCESSING_APPLICATION = app2;
    ACCESSING_APPLICATION = app3;
  };
  APPLICATION app1 {
    TASK = t1;
    COUNTER = Software_Counter1;
    ALARM = Alarm1;
    ALARM = Alarm1_1;
    ALARM = Alarm2;
    ALARM = Alarm2_1;
    ALARM = Alarm3;
    ALARM = Alarm3_1;
    ALARM = Alarm4;
    ALARM = Alarm4_1;
    SCHEDULETABLE = sched1;
    SCHEDULETABLE = sched1_1;
    SCHEDULETABLE = sched2;
    SCHEDULETABLE = sched2_1;
    SCHEDULETABLE = sched3;
    SCHEDULETABLE = sched3_1;
  };
  APPLICATION app2 {
    TASK = t2;
    COUNTER = Software_Counter2;
  };
  APPLICATION app3 {
    TASK = t3;
  };
  EVENT Event1 {
    MASK = AUTO;
  };
```

Wrong application	Return Status	Test case
Alarm1's counter doesn't belong to the same application of Alarm1	error : Counter Software_Counter2 doesn't belong to the same application of alarm Alarm1	1
Alarm2 ACTIVATETASK's task doesn't belong to the same application of Alarm2	error : Task t2 doesn't belong to the same application of alarm Alarm2	2
Alarm3 SETEVENT's task doesn't belong to the same application of Alarm3	error : Task t2 doesn't belong to the same application of alarm Alarm3	3

Wrong application	Return Status	Test case
Alarm4 INCREMENTCOUNTER's counter doesn't belong to the same application of Alarm4	error : Counter Software_Counter2 doesn't belong to the same application of alarm Alarm4	4
sched1's counter doesn't belong to the same application of sched1	error : Counter Software_Counter2 doesn't belong to the same application of schedule table sched1	5
sched2 ACTIVATETASK's task doesn't belong to the same application of sched2	error : Task t3 doesn't belong to the same application of schedule table sched2	6
sched2 SETEVENT's task doesn't belong to the same application of sched2	error : Task t2 doesn't belong to the same application of schedule table sched2	7
sched3 ACTIVATETASK's task doesn't belong to the same application of sched3	error : Task t3 doesn't belong to the same application of schedule table sched3	6
sched3 SETEVENT's task doesn't belong to the same application of sched3	error : Task t2 doesn't belong to the same application of schedule table sched3	7

Notes

[1] as said in Trampoline Test Plan - 2.3 Event Mechanism, this test set an event to a READY_AND_NEW task unlike the other test cases 9 and 10.

References

[1] Consortium OSEK/VDX. *OSEK/VDX Operating System*, 2.2.3 edition, 17th February 2005.

[2] Consortium OSEK/VDX. *OSEK/VDX Operating System*, 2.0 edition, 2001.

[3] Consortium OSEK/VDX. *OSEK/VDX OS Test Procedure*, 2.0 edition, 21st April 1999.

[4] Florent PAVIN and Jean-Luc BECHENNEC. *Trampoline (OSEK/VDX) Test Procedure.* IRCCYN, 1.0 edition, April 2009.

[5] Florent PAVIN and Jean-Luc BECHENNEC. *Trampoline (OSEK/VDX) Test Plan.* IRCCYN, 1.0 edition, February 2009.