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

Florent PAVIN ; Jean-Luc BECHENNEC

December 3, 2009

Contents

1 Introduction

This document contains the test plan for the conformance test of the operating system. This means definition of the test cases, which are used to certify conformance of an OS implementation. For more information about what is a test plan and his link to the conformance methodology previously defined, see OSEK Test Plan 2.0 [1].
Unlike OSEK Test Plan 2.0 which is based from OSEK OS 2.0 [3], this test plan is defined from OSEK OS 2.2.3 [2] and the internal communication of OSEK Communication 3.0.3 [4] .

2 Test cases

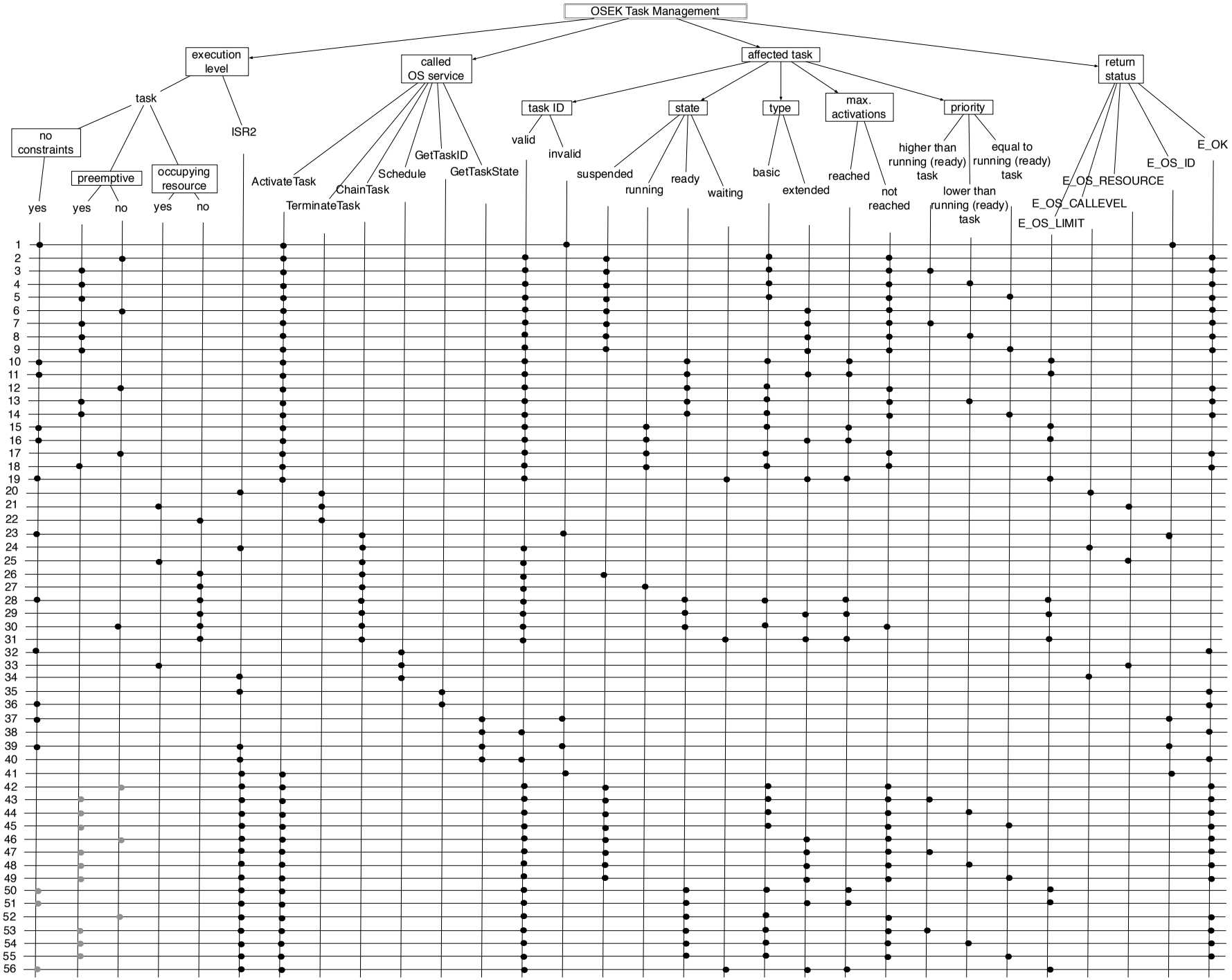
This chapter contains the test cases which will be used to test an implementation of an operating system to be OSEK conform. Thus, they are developed on the basis of the OSEK OS specification, according to figure 12-1 API service restrictions from OSEK/VDX OS v2.2.3. The internal communication comes from CCCB conformance class ([4] p.59).
As we said earlier, this test plan is defined from the OSEK OS version 2.2.3, and to better see the differences between this version and the old one (OSEK Test Plan 2.0), we will explain those differences in each section.
ISR1 does not use an operating system service since after the ISR1 is finished, processing continues exactly at the instruction where the interrupt has occurred, i.e. the interrupt has no influence on task management. Thus, **ISR can't be tested**.
Stack Monitoring, from AUTOSAR OS, is not a functional test. It has to be tested in every target because it's

depending on the portage. *Stack Monitoring* OS Requirements (OS067, OS068, OS396) are therfore not included in this report.
Idem for *Memory Protection* OS Requirements (OS026, OS027, OS044, OS081, OS083, OS086, OS087, OS195, OS196, OS198, OS207, OS208, OS209, OS355, OS356).
Idem for *Protecting the Hardware*.

2.1 Task management

Since Schedule() returns E_OS_RESSOURCE from a task or an interrupt when a resource is occupied, test case 33 appears.
Since GetTaskID returns E_OK from an interrupt, test case 35 appears.
Category 3 interrupts have been removed.

Test Case No.	Action	Expected Result
1	Call ActivateTask() from task-level with invalid task ID (task does not exist)	Service returns E_OS_ID
2	Call ActivateTask() from non-preemptive task on suspended basic task	No preemption of running task. Activated task becomes ready. Service returns E_OK
3	Call ActivateTask() from preemptive task on suspended basic task which has higher priority than running task.	Running task is preempted. Activated task becomes running. Service returns E_OK
4	Call ActivateTask() from preemptive task on suspended basic task which has lower priority than running task.	No preemption of running task. Activated task becomes ready. Service returns E_OK
5	Call ActivateTask() from preemptive task on suspended basic task which has equal priority as running task.	No preemption of running task. Activated task becomes ready. Service returns E_OK
6	Call ActivateTask() from non-preemptive task on suspended extended task	No preemption of running task. Activated task becomes ready and its events are cleared. Service returns E_OK
7	Call ActivateTask() from preemptive task on suspended extended task which has higher priority than running task.	Running task is preempted. Activated task becomes running and its events are cleared. Service returns E_OK
8	Call ActivateTask() from preemptive task on suspended extended task which has lower priority than running task.	No preemption of running task. Activated task becomes ready and its events are cleared. Service returns E_OK
9	Call ActivateTask() from preemptive task on suspended extended task which has equal priority as running task.	No preemption of running task. Activated task becomes ready and its events are cleared. Service returns E_OK
10	Call ActivateTask() on ready basic task which has reached max. number of activations	Service returns E_OS_LIMIT
11	Call ActivateTask() on ready extended task	Service returns E_OS_LIMIT
12	Call ActivateTask() from non-preemptive task on ready basic task which has not reached max. number of activations	No preemption of running task. Activation request is queued in ready list. Service returns E_OK
13	Call ActivateTask() from preemptive task on ready basic task which has not reached max. number of activations and has lower priority than running task1	No preemption of running task. Activation request is queued in ready list. Service returns E_OK


OSEK Task Management
execution level
task
no constraints
preemptive
occupying resource
yes
yes
no
yes
no
ISR2
called OS service
ActivateTask
TerminateTask
ChainTask
Schedule
GetTaskID
GetTaskState
affected task
task ID
valid
invalid
state
suspended
running
ready
waiting
type
basic
extended
max. activations
reached
not reached
priority
higher than running (ready) task
lower than running (ready) task
equal to running (ready) task
return status
E_OK
E_OS_ID
E_OS_RESOURCE
E_OS_CALLEVEL
E_OS_LIMIT
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

Test Case No.	Action	Expected Result
14	Call ActivateTask() from preemptive task on ready basic task which has not reached max. number of activations and has equal priority as running task	No preemption of running task. Activation request is queued in ready list. Service returns E_OK
15	Call ActivateTask() on running basic task which has reached max. number of activations	Service returns E_OS_LIMIT
16	Call ActivateTask() on running extended task	Service returns E_OS_LIMIT
17	Call ActivateTask() from non-preemptive task on running basic task which has not reached max. number of activations	No preemption of running task. Activation request is queued in ready list. Service returns E_OK
18	Call ActivateTask() from preemptive task on running basic task which has not reached max. number of activations	No preemption of running task. Activation request is queued in ready list. Service returns E_OK
19	Call ActivateTask() on waiting extended task	Service returns E_OS_LIMIT
20	Call TerminateTask() from ISR category 2	Service returns E_OS_CALLEVEL
21	Call TerminateTask() while still occupying a resource Running task is not terminated.	Service returns E_OS_RESOURCE
22	Call TerminateTask()	Running task is terminated and ready task with highest priority is executed
23	Call ChainTask() from task-level. Task-ID is invalid (does not exist).	Service returns E_OS_ID
24	Call ChainTask() from ISR category 2	Service returns E_OS_CALLEVEL
25	Call ChainTask() while still occupying a resource	Running task is not terminated. Service returns E_OS_RESOURCE
26	Call ChainTask() on suspended task	Running task is terminated, chained task becomes ready and ready task with highest priority is executed
27	Call ChainTask() on running task	Running task is terminated, chained task becomes ready and ready task with highest priority is executed
28	Call ChainTask() on ready basic task which has reached max. number of activations	Running task is not terminated. Service returns E_OS_LIMIT
29	Call ChainTask() on ready extended task	Running task is not terminated. Service returns E_OS_LIMIT
30	Call ChainTask() from non-preemptive task on ready basic task which has not reached max. number of activations	Running task is terminated, activation request is queued in ready list and ready task with highest priority is executed
31	Call ChainTask() on waiting extended task	Service returns E_OS_LIMIT
32	Call Schedule() from task.	Ready task with highest priority is executed. Service returns E_OK
33	Call Schedule() while still occupying a resource	Service returns E_OS_RESOURCE
34	Call Schedule() from ISR category 2	Service returns E_OS_CALLEVEL
35	Call GetTaskID() from ISR category 2	Service returns E_OK
36	Call GetTaskID() from task	Return task ID of currently running task. Service returns E_OK
37	Call GetTaskState() with invalid task ID (task does not exist)	Service returns E_OS_ID
38	Call GetTaskState() Return state of queried task.	Service returns E_OK
39	Call GetTaskState() from ISR2 with invalid task ID (task does not exist)	Service returns E_OS_ID
40	Call GetTaskState() from ISR2. Return state of queried task.	Service returns E_OK

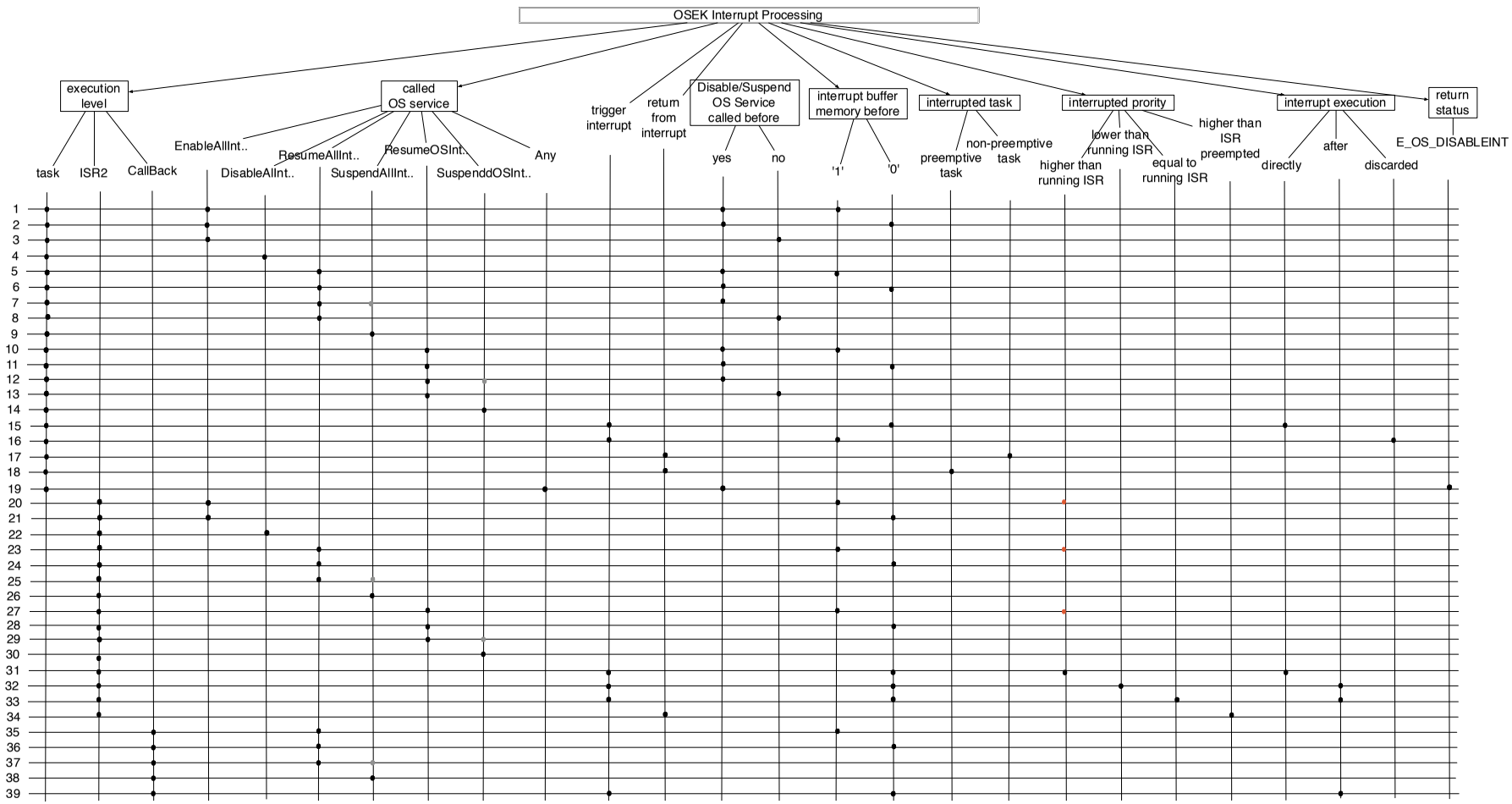
Test Case No.	Action	Expected Result
41	Call ActivateTask() from ISR2 with invalid task ID (task does not exist)	Service returns E_OS_ID
42	Call ActivateTask() from ISR2 (in non-preemptive mode) on suspended basic task.	Activated task becomes ready. Service returns E_OK
43	Call ActivateTask() from ISR2 (in preemptive mode) on suspended basic task which has higher priority than last running task.	Activated task becomes ready and first. Service returns E_OK
44	Call ActivateTask() from ISR2 (in preemptive mode) on suspended basic task which has lower priority than last running task.	Activated task becomes ready. Service returns E_OK
45	Call ActivateTask() from ISR2 (in preemptive mode) on suspended basic task which has equal priority as last running task.	Activated task becomes ready. Service returns E_OK
46	Call ActivateTask() from ISR2 (in non-preemptive mode) on suspended extended task	Activated task becomes ready and its events are cleared. Service returns E_OK
47	Call ActivateTask() from ISR2 (in preemptive mode) on suspended extended task which has higher priority than last running task.	Activated task becomes ready and first and its events are cleared. Service returns E_OK
48	Call ActivateTask() from ISR2 (in preemptive mode) on suspended extended task which has lower priority than last running task.	Activated task becomes ready and its events are cleared. Service returns E_OK
49	Call ActivateTask() from ISR2 (in preemptive mode) on suspended extended task which has equal priority as last running task.	Activated task becomes ready and its events are cleared. Service returns E_OK
50	Call ActivateTask() from ISR2 on ready basic task which has reached max. number of activations	Service returns E_OS_LIMIT
51	Call ActivateTask() from ISR2 on ready extended task	Service returns E_OS_LIMIT
52	Call ActivateTask() from ISR2 (in non-preemptive mode) on ready basic task which has not reached max. number of activations	Activation request is queued in ready list. Service returns E_OK
53	Call ActivateTask() from ISR2 (in preemptive mode) on ready basic task which has not reached max. number of activations and has higher priority than last running	Activation request is queued in ready list on first place. Service returns E_OK
54	Call ActivateTask() from ISR2 (in preemptive mode) on ready basic task which has not reached max. number of activations and has lower priority than last running task1	Activation request is queued in ready list. Service returns E_OK
55	Call ActivateTask() from ISR2 (in preemptive mode) on ready basic task which has not reached max. number of activations and has equal priority as last running task	Activation request is queued in ready list. Service returns E_OK
56	Call ActivateTask() from ISR2 on waiting extended task	Service returns E_OS_LIMIT

2.2 Interrupt processing

New routines appear (EnableAllInterrupts, DisableAllInterrupts, SuspendAllInterrupts, ResumeAllInterrupts, SuspendOSInterrupts, ResumeOSInterrupts), test cases 1 to 19 are new ones.
Category 3 interrupts have been removed.

Maximum number of activation of ISR2 can't be more than 1.
EnableAllInterrupts, ResumeAllInterrupts and ResumeOSInterrupts from ISR2 are only tested with an interrupt trigged with a priority higher than running ISR2.
SuspendAllInterrupts and ResumeAllInterrupts are the only ones functions allowed in callback routines.

Test Case No.	Action	Expected Result
1	Call EnableAllInterrupts() from task. An interrupt has been trigged in disable mode	The Interrupt is executed. Running task become ready
2	Call EnableAllInterrupts() from task	Enable all interrupts
3	Call EnableAllInterrupts() from task without calling DisableAllInterrupts()	The service is not performed
4	Call DisableAllInterrupts() from task	Disable all interrupts
5	Call ResumeAllInterrupts() from task. An interrupt has been trigged in disable mode	The Interrupt is executed. Running task become ready
6	Call ResumeAllInterrupts() from task	Resume all interrupts
7	Call ResumeAllInterrupts() from task as many times as SuspendAllInterrupts() is previously called	Resume all interrupts
8	Call ResumeAllInterrupts() from task without calling SuspendAllInterrupts()	The service is not performed
9	Call SuspendAllInterrupts() from task	Suspend all interrupts
10	Call ResumeOSInterrupts() from task. An interrupt has been trigged in disable mode	The Interrupt is executed. Running task become ready
11	Call ResumeOSInterrupts() from task	Resume OS interrupts
12	Call ResumeOSInterrupts() from task as many times as SuspendOSInterrupts() is previously called	Resume OS interrupts
13	Call ResumeOSInterrupts() from task without calling SuspendOSInterrupts()	The service is not performed
14	Call SuspendOSInterrupts() from task	Suspend OS interrupts
15	Interruption of running task	Interrupt is executed
16	Interruption of running task with the same interrupt already trigged (activation count = activation max)	Interrupt is discarded
17	Return from ISR2. Interrupted task is non-preemptive	Execution of interrupted task is continued
18	Return from ISR2. Interrupted task is preemptive	Ready task with highest priority is executed (Rescheduling)
19	Call any OS service between Suspend/Disable- and Resume/Enable-pairs	Service returns E_OS_DISABLEINT and not perform the service (see AUTOSAR OS092), even Disable and Enable pairs (see OSEK p26)
20	Call EnableAllInterrupts() from ISR2. An interrupt has been trigged in disable mode with a higher priority than running ISR2	The Interrupt is executed. Running ISR2 becomes ready
21	Call EnableAllInterrupts() from ISR2	Enable all interrupts
22	Call DisableAllInterrupts() from ISR2	Disable all interrupts
23	Call ResumeAllInterrupts() from ISR2. An interrupt has been trigged in disable mode with a higher priority than running ISR2	The Interrupt is executed. Running ISR2 becomes ready

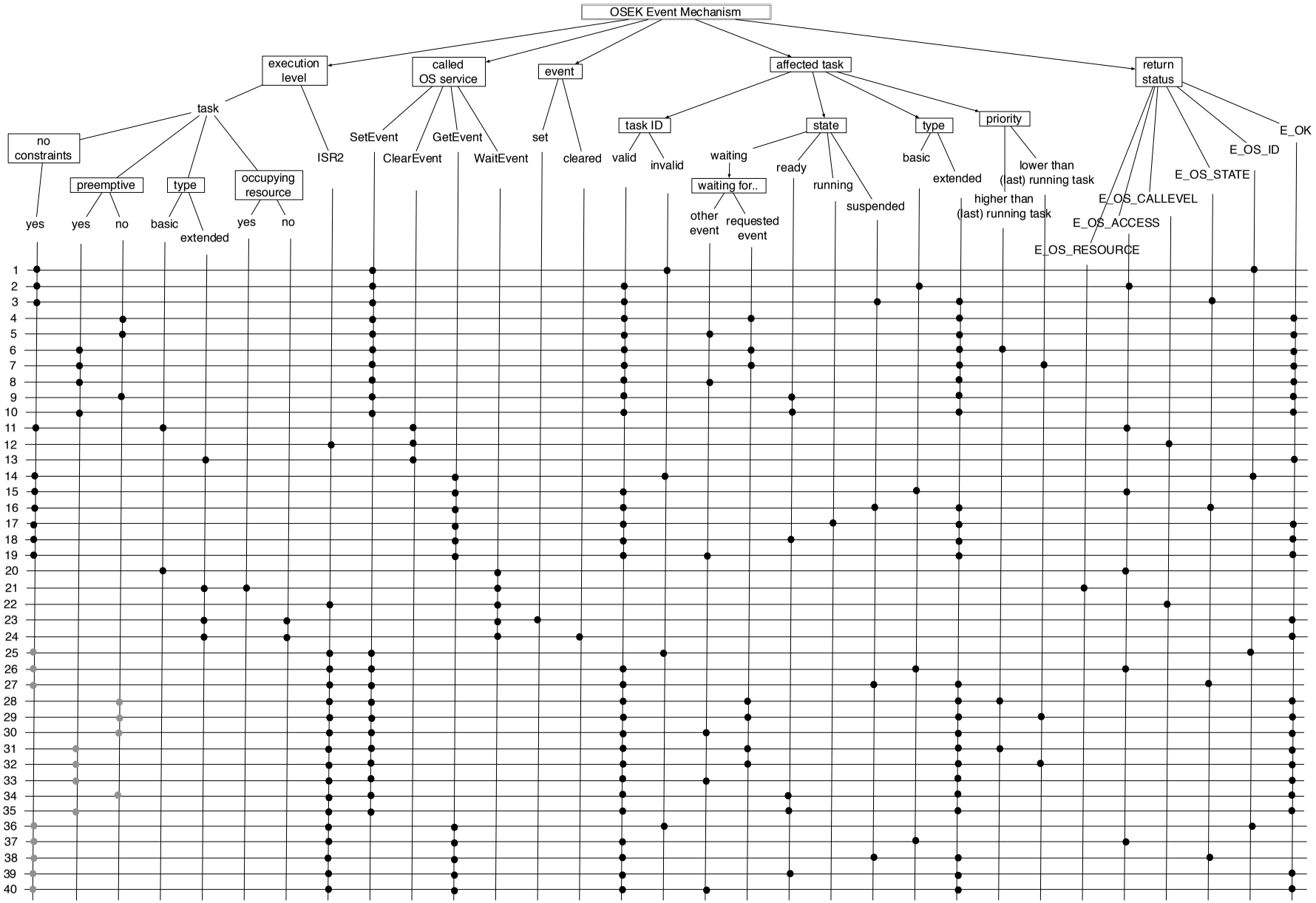

OSEK Interrupt Processing
execution level
task
ISR2
CallBack
called OS service
EnableAllInt..
DisableAllInt..
ResumeAllInt..
SuspendAllInt..
ResumeOSInt..
SuspenddOSInt..
Any
trigger interrupt
return from interrupt
Disable/Suspend OS Service called before
yes
no
interrupt buffer memory before
'1'
'0'
interrupted task
preemptive task
non-preemptive task
interrupted prority
higher than running ISR
lower than running ISR
equal to running ISR
higher than ISR preempted
interrupt execution
directly
after
discarded
return status
E_OS_DISABLEINT
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39

Test Case No.	Action	Expected Result
24	Call ResumeAllInterrupts() from ISR2	Resume all interrupts
25	Call ResumeAllInterrupts() from ISR2 as many times as SuspendAllInterrupts() is previously called	Resume all interrupts
26	Call SuspendAllInterrupts() from ISR2	Suspend all interrupts
27	Call ResumeOSInterrupts() from ISR2. An interrupt has been trigged in disable mode with a higher priority than running ISR2	The Interrupt is executed. Running ISR2 becomes ready
28	Call ResumeOSInterrupts() from ISR2	Resume OS interrupts
29	Call ResumeOSInterrupts() from ISR2 as many times as SuspendOSInterrupts() is previously called	Resume OS interrupts
30	Call SuspendOSInterrupts() from ISR2	Suspend OS interrupts
31	Interruption of running ISR2 on interrupt which has higher priority than running interrupt	Running Interrupt is preempted. Executed interrupt becomes running
32	Interruption of running ISR2 on interrupt which has lower priority than running interrupt	No preemption of running interrupt. Executed interrupt becomes ready
33	Interruption of running ISR2 on interrupt which has equal priority as running interrupt	No preemption of running interrupt. Executed interrupt becomes ready
34	Return from ISR2 to an ISR2 which has higher priority than ISR2 preempted	ISR2 with the highest priority is executed
35	Call ResumeAllInterrupts() from callback routine. An interrupt has been trigged in disable mode	No preemption of callback routine because ISR2 are disabled in callback routines
36	Call ResumeAllInterrupts() from callback routine	Resume all interrupts
37	Call ResumeAllInterrupts() from callback routine as many times as SuspendAllInterrupts() is previously called	Resume all interrupts
38	Call SuspendAllInterrupts() from callback routine	Suspend all interrupts
39	Interruption in callback routines	Interrupt is executed after callback routines

2.3 Event mechanism

Category 3 interrupts have been removed.
Test cases 9 and 10 have to be tested with a simple ready task and with a READY_AND_NEW task (a task which juste came to be ready).
Test cases 41 to 43 are GOIL test cases.

Test Case No.	Action	Expected Result
1	Call SetEvent() with invalid Task ID	Service returns E_OS_ID
2	Call SetEvent() for basic task	Service returns E_OS_ACCESS
3	Call SetEvent() for suspended extended task	Service returns E_OS_STATE
4	Call SetEvent() from non-preemptive task on waiting extended task which is waiting for at least one of the requested events	Requested events are set. Running task is not preempted. Waiting task becomes ready. Service returns E_OK


OSEK Event Mechanism
execution level
called OS service
event
affected task
return status
task
ISR2
SetEvent
ClearEvent
GetEvent
WaitEvent
set
cleared
task ID
state
type
priority
E_OK
E_OS_ID
E_OS_STATE
E_OS_CALLEVEL
E_OS_ACCESS
E_OS_RESOURCE
no constraints
preemptive
type
occupying resource
yes
yes
no
basic
extended
yes
no
valid
invalid
waiting
waiting for..
other event
requested event
ready
running
suspended
basic
extended
higher than (last) running task
lower than (last) running task
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40

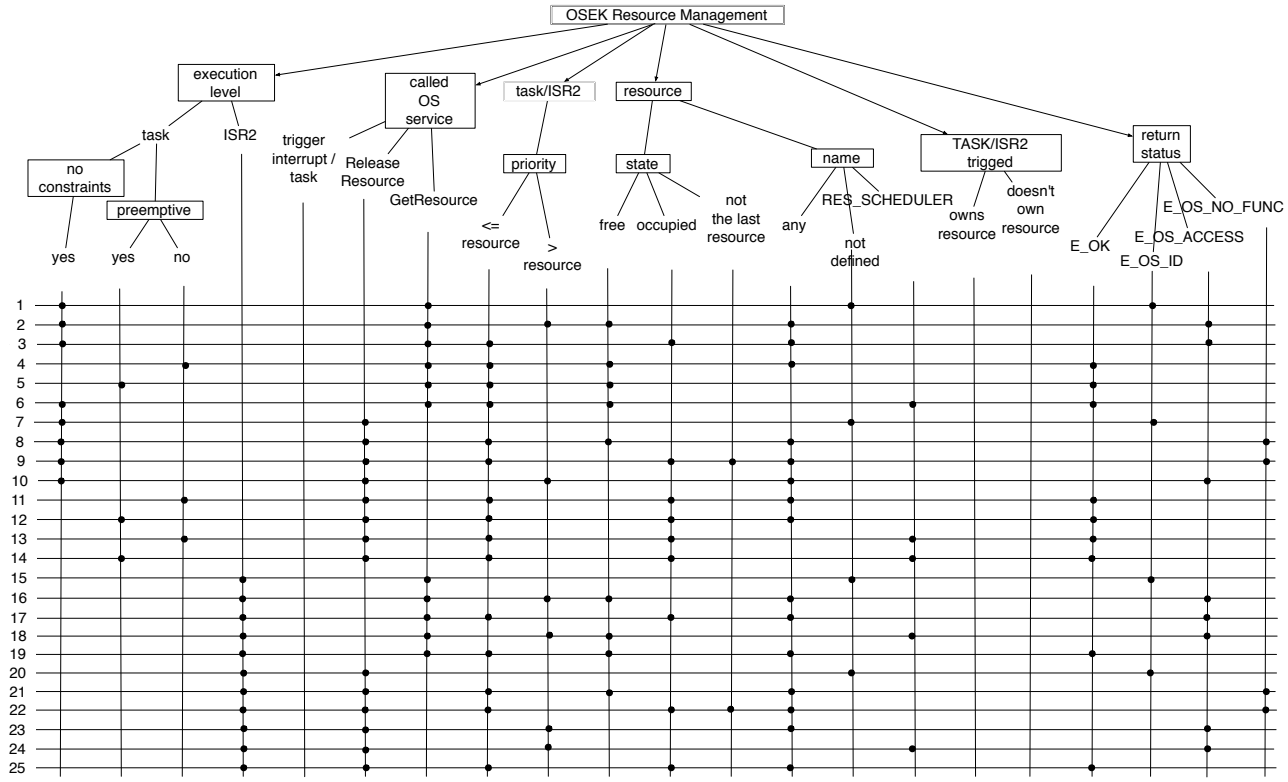
Test Case No.	Action	Expected Result
5	Call SetEvent() from non-preemptive task on waiting extended task which is not waiting for any of the requested events	Requested events are set. Running task is not preempted. Waiting task doesn't become ready. Service returns E_OK
6	Call SetEvent() from preemptive task on waiting extended task which is waiting for at least one of the requested events and has higher priority than running task	Requested events are set. Running task becomes ready (is preempted). Waiting task becomes running. Service returns E_OK
7	Call SetEvent() from preemptive task on waiting extended task which is waiting for at least one of the requested events and has equal or lower priority than running task	Requested events are set. Running task is not preempted. Waiting task becomes ready. Service returns E_OK
8	Call SetEvent() from preemptive task on waiting extended task which is not waiting for any of the requested events	Requested events are set. Running task is not preempted. Waiting task doesn't become ready. Service returns E_OK
9	Call SetEvent() from non-preemptive task on ready extended task	Requested events are set. Running task is not preempted. Service returns E_OK
10	Call SetEvent() from preemptive task on ready extended task	Requested events are set. Running task is not preempted. Service returns E_OK
11	Call ClearEvent() from basic task	Service returns E_OS_ACCESS
12	Call ClearEvent() from ISR2	Service returns E_OS_CALLEVEL
13	Call ClearEvent() from extended task	Requested events are cleared. Service returns E_OK
14	Call GetEvent() with invalid Task ID	Service returns E_OS_ID
15	Call GetEvent() for basic task	Service returns E_OS_ACCESS
16	Call GetEvent() for suspended extended task	Service returns E_OS_STATE
17	Call GetEvent() for running extended task	Return current state of all event bits. Service returns E_OK
18	Call GetEvent() for ready extended task	Return current state of all event bits. Service returns E_OK
19	Call GetEvent() for waiting extended task	Return current state of all event bits. Service returns E_OK
20	Call WaitEvent() from basic task	Service returns E_OS_ACCESS
21	Call WaitEvent() from extended task which occupies a resource	Service returns E_OS_RESOURCE
22	Call WaitEvent() from ISR2	Service returns E_OS_CALLEVEL
23	Call WaitEvent() from extended task. None of the events waited for is set	Running task becomes waiting and ready task with highest priority is executed Service returns E_OK
24	Call WaitEvent() from extended task. At least one event waited for is already set	No preemption of running task Service returns E_OK
25	Call SetEvent() from ISR2 with invalid Task ID	Service returns E_OS_ID
26	Call SetEvent() from ISR2 for basic task	Service returns E_OS_ACCESS
27	Call SetEvent() from ISR2 for suspended extended task	Service returns E_OS_STATE
28	Call SetEvent() from ISR2 (in non-preemptive mode) on waiting extended task which is waiting for at least one of the requested events and has higher priority than last running task	Requested events are set. Waiting task becomes ready. Service returns E_OK
29	Call SetEvent() from ISR2 (in non-preemptive mode) on waiting extended task which is waiting for at least one of the requested events and has lower priority than last running task	Requested events are set. Waiting task becomes ready. Service returns E_OK

Test Case No.	Action	Expected Result
30	Call SetEvent() from ISR2 (in non-preemptive mode) on waiting extended task which is not waiting for any of the requested events	Requested events are set. Waiting task doesn't become ready. Service returns E_OK
31	Call SetEvent() from ISR2 (in preemptive mode) on waiting extended task which is waiting for at least one of the requested events and has higher priority than running task	Requested events are set. Waiting task becomes ready and first. Service returns E_OK
32	Call SetEvent() from ISR2 (in preemptive mode) on waiting extended task which is waiting for at least one of the requested events and has equal or lower priority than running task	Requested events are set. Waiting task becomes ready. Service returns E_OK
33	Call SetEvent() from ISR2 (in preemptive mode) on waiting extended task which is not waiting for any of the requested events	Requested events are set. Waiting task doesn't become ready. Service returns E_OK
34	Call SetEvent() from ISR2 (in non-preemptive mode) on ready extended task	Requested events are set. Service returns E_OK
35	Call SetEvent() from ISR2 (in preemptive mode) on ready extended task	Requested events are set. Service returns E_OK
36	Call GetEvent() from ISR2 with invalid Task ID	Service returns E_OS_ID
37	Call GetEvent() from ISR2 for basic task	Service returns E_OS_ACCESS
38	Call GetEvent() from ISR2 for suspended extended task	Service returns E_OS_STATE
39	Call GetEvent() from ISR2 for ready extended task	Return current state of all event bits. Service returns E_OK
40	Call GetEvent() from ISR2 for waiting extended task	Return current state of all event bits. Service returns E_OK
41	Creating an event with a MASK using more than one bit	Warning : Event Mask uses more than one bit
42	Creating an event with a MASK already used	Error : Mask already used
43	Creating an event with an automatic MASK but all the MASK are already used	Error : All mask bits are already used, the last event can't be created

2.4 Resource management

An ISR2 is like a task, it can get and release resources if it's allowed (if it owns the resource). See test cases 3, 4, 9 and 10.
GetResource() returns E_OS_ACCESS if the resource's priority is inferior to the task's priority (it means the task doesn't use it so if it gets the resource, the resource is not well shared). Otherwise, a task is allowed to get a Resource with a priority higher than itself.
There's no more maximum number of nested resources reachable.
Category 3 interrupts have been removed.

Test Case No.	Action	Expected Result
1	Call GetResource() from task with invalid resource ID	Service returns E_OS_ID
2	Call GetResource() from task with priority of the calling task higher than the calculated ceiling priority	Service returns E_OS_ACCESS



Test Case No.	Action	Expected Result
3	Call GetResource() from task with occupied resource	Service returns E_OS_ACCESS
4	Test Priority Ceiling Protocol: Call GetResource() from non-preemptive task, activate task/ISR2 with priority higher than running task but lower than ceiling priority, and force rescheduling	Resource is occupied and running task's priority is set to resource's ceiling priority. Service returns E_OK. No preemption occurs after activating the task with higher priority and rescheduling
5	Test Priority Ceiling Protocol: Call GetResource() from preemptive task, and activate task/ISR2 with priority higher than running task but lower than ceiling priority	Resource is occupied and running task's priority is set to resource's ceiling priority. Service returns E_OK. No preemption occurs after activating the task with higher priority
6	Call GetResource() from task for resource RES_SCHEDULER	Resource is occupied and running task's priority is set to resource's ceiling priority. Service returns E_OK
7	Call ReleaseResource() from task with invalid resource ID	Service returns E_OS_ID
8	Call ReleaseResource() from task with resource which is not occupied	Service returns E_OS_NOFUNC
9	Call ReleaseResource() from task when another resource shall be released before	Service returns E_OS_NOFUNC
10	Call ReleaseResource() from task with priority of the calling task higher than the calculated ceiling priority	Service returns E_OS_ACCESS
11	Call ReleaseResource() from non-preemptive task	Resource is released and running task's priority is reset. No preemption of running task. Service returns E_OK

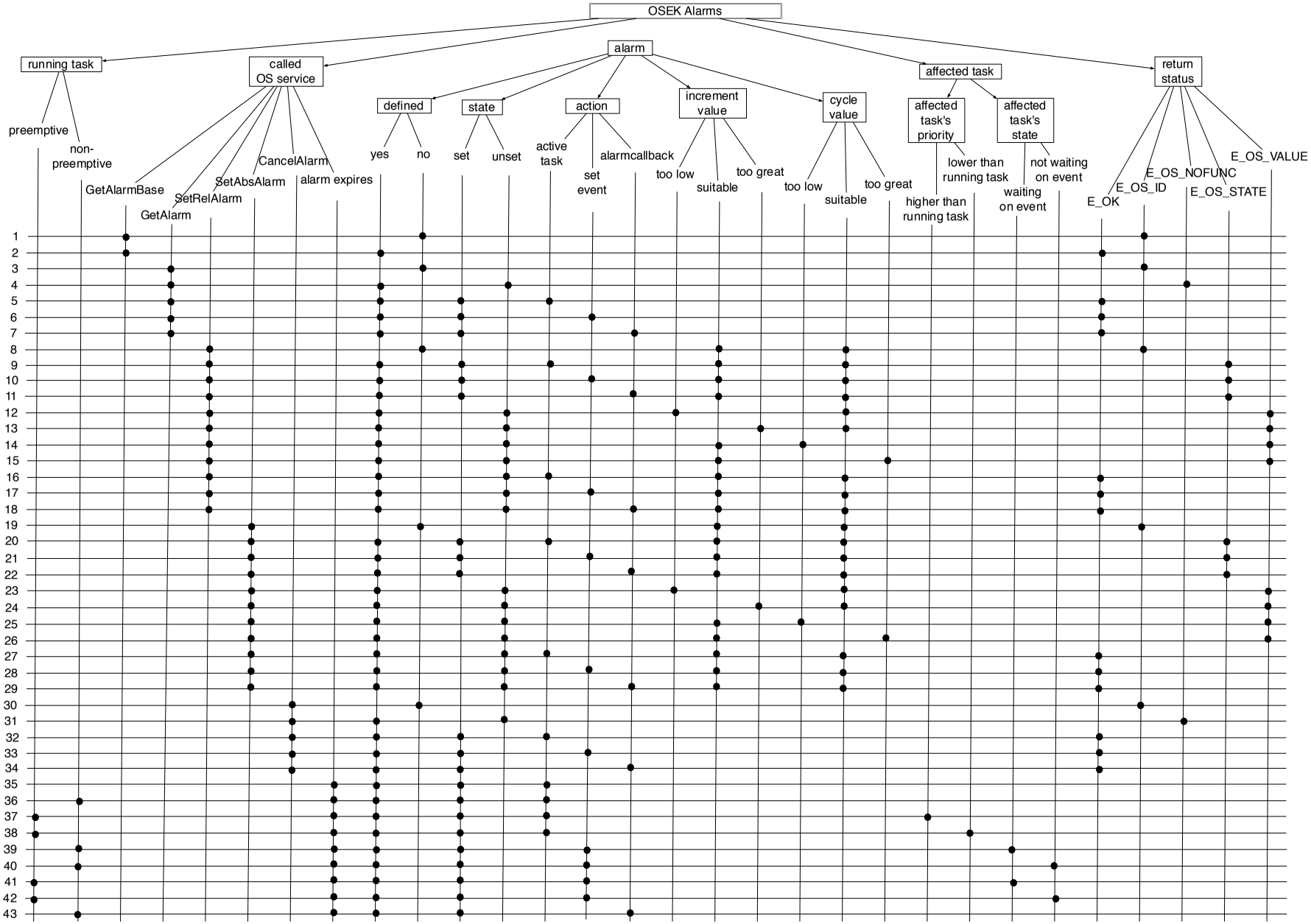
Test Case No.	Action	Expected Result
12	Call ReleaseResource() from preemptive task	Resource is released and running task's priority is reset. Ready task with highest priority is executed (Rescheduling). Service returns E_OK
13	Call ReleaseResource() from non-preemptive task for resource RES_SCHEDULER	Resource is released and running task's priority is reset. No preemption of running task. Service returns E_OK
14	Call ReleaseResource()from preemptive task for resource RES_SCHEDULER	Resource is released and running task's priority is reset. Ready task with highest priority is executed (Rescheduling). Service returns E_OK
15	Call GetResource() from ISR2 with invalid resource ID	Service returns E_OS_ID
16	Call GetResource() from ISR2 with priority of the calling ISR2 higher than the calculated ceiling priority	Service returns E_OS_ACCESS
17	Call GetResource() from ISR2 with occupied resource	Service returns E_OS_ACCESS
18	Call GetResource() from ISR2 for resource RES_SCHEDULER	Service returns E_OS_ACCESS
19	Test Priority Ceiling Protocol: Call GetResource() from ISR2, and activate ISR2 with priority higher than running ISR2 but lower than ceiling priority	Resource is occupied and running ISR2's priority is set to resource's ceiling priority. Service returns E_OK. No preemption occurs after activating the ISR2 with higher priority
20	Call ReleaseResource() from ISR2 with invalid resource ID	Service returns E_OS_ID
21	Call ReleaseResource() from ISR2 with resource which is not occupied	Service returns E_OS_NOFUNC
22	Call ReleaseResource() from ISR2 when another resource shall be released before	Service returns E_OS_NOFUNC
23	Call ReleaseResource() from ISR2 with priority of the calling ISR2 higher than the calculated ceiling priority	Service returns E_OS_ACCESS
24	Call ReleaseResource() from ISR2 for resource RES_SCHEDULER (priority of the calling ISR2 higher than the calculated ceiling priority)	Service returns E_OS_ACCESS
25	Call ReleaseResource() from ISR2	Resource is released and running ISR2's priority is reset. Ready task/ISR2 with highest priority is executed (Rescheduling). Service returns E_OK

2.5 Alarm

The behaviour of the OS is not defined by the specification if the action assigned to the expiration of an alarm can not be performed, because

- it would lead to multiple task activation, which is not allowed in the used conformance class or the max. number of activated tasks is already reached, or
- it would set an event for a task which is currently suspended.

The expected behaviour is, that at least the error hook is called. But as this situation is not covered by the specification, it is not part of conformance testing.
Since AlarmCallBack routine have been integrated in OSEK OS Specifications v2.2.3, test cases 7, 11, 18, 22, 29, 34 and 43 appear.


OSEK Alarms
running task
called OS service
alarm
affected task
return status
preemptive
non-preemptive
GetAlarmBase
GetAlarm
SetRelAlarm
SetAbsAlarm
CancelAlarm
alarm expires
defined
state
action
increment value
cycle value
affected task's priority
affected task's state
yes
no
set
unset
active task
set event
alarmcallback
too low
suitable
too great
too low
suitable
too great
higher than running task
lower than running task
waiting on event
not waiting on event
E_OK
E_OS_ID
E_OS_NOFUNC
E_OS_STATE
E_OS_VALUE
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43

Test Case No.	Action	Expected Result
1	Call GetAlarmBase() with invalid alarm ID	Service returns E_OS_ID
2	Call GetAlarmBase() Return alarm base characteristics.	Service returns E_OK
3	Call GetAlarm() with invalid alarm ID	Service returns E_OS_ID
4	Call GetAlarm() for alarm which is currently not in use	Service returns E_OS_NOFUNC
5	Call GetAlarm() for alarm which will activate a task on expiration	Returns number of ticks until expiration. Service returns E_OK
6	Call GetAlarm() for alarm which will set an event on expiration	Returns number of ticks until expiration. Service returns E_OK
7	Call GetAlarm() for alarm which will callback a routine on expiration	Returns number of ticks until expiration. Service returns E_OK
8	Call SetRelAlarm() with invalid alarm ID	Service returns E_OS_ID
9	Call SetRelAlarm() for already activated alarm which will activate a task on expiration	Service returns E_OS_STATE
10	Call SetRelAlarm() for already activated alarm which will set an event on expiration	Service returns E_OS_STATE
11	Call SetRelAlarm() for already activated alarm which will callback a routine on expiration	Service returns E_OS_STATE
12	Call SetRelAlarm() with increment value lower than zero	Service returns E_OS_VALUE
13	Call SetRelAlarm() with increment value greater than maxallowedvalue	Service returns E_OS_VALUE
14	Call SetRelAlarm() with cycle value lower than mincycle	Service returns E_OS_VALUE
15	Call SetRelAlarm() with cycle value greater than maxallowedvalue	Service returns E_OS_VALUE
16	Call SetRelAlarm() for alarm which will activate a task on expiration	Alarm is activated. Service returns E_OK
17	Call SetRelAlarm() for alarm which will set an event on expiration	Alarm is activated.Service returns E_OK
18	Call SetRelAlarm() for alarm which will callback a routine on expiration	Alarm is activated.Service returns E_OK
19	Call SetAbsAlarm() with invalid alarm ID	Service returns E_OS_ID
20	Call SetAbsAlarm() for already activated alarm which will activate a task on expiration	Service returns E_OS_STATE
21	Call SetAbsAlarm() for already activated alarm which will set an event on expiration	Service returns E_OS_STATE
22	Call SetAbsAlarm() for already activated alarm which will callback a routine on expiration	Service returns E_OS_STATE
23	Call SetAbsAlarm() with increment value lower than zero	Service returns E_OS_VALUE
24	Call SetAbsAlarm() with increment value greater than maxallowedvalue	Service returns E_OS_VALUE
25	Call SetAbsAlarm() with cycle value lower than mincycle	Service returns E_OS_VALUE

Test Case No.	Action	Expected Result
26	Call SetAbsAlarm() with cycle value greater than maxallowedvalue	Service returns E_OS_VALUE
27	Call SetAbsAlarm() for alarm which will activate a task on expiration	Alarm is activated. Service returns E_OK
28	Call SetAbsAlarm() for alarm which will set an event on expiration	Alarm is activated. Service returns E_OK
29	Call SetAbsAlarm() for alarm which will callback a routine on expiration	Alarm is activated. Service returns E_OK
30	Call CancelAlarm() with invalid alarm ID	Service returns E_OS_ID
31	Call CancelAlarm() for alarm which is currently not in use	Service returns E_OS_NOFUNC
32	Call CancelAlarm() for already activated alarm which will activate a task on expiration	Alarm is cancelled. Service returns E_OK
33	Call CancelAlarm() for already activated alarm which will set an event on expiration	Alarm is cancelled. Service returns E_OK
34	Call CancelAlarm() for already activated alarm which will callback a routine on expiration	Alarm is cancelled. Service returns E_OK
35	Expiration of alarm which activates a task while no tasks are currently running	Task is activated
36	Expiration of alarm which activates a task while running task is non-preemptive	Task is activated. No preemption of running task
37	Expiration of alarm which activates a task with higher priority than running task while running task is preemptive	Task is activated. Task with highest priority is executed
38	Expiration of alarm which activates a task with lower priority than running task while running task is preemptive	Task is activated. No preemption of running task.
39	Expiration of alarm which sets an event while running task is non-preemptive. Task which owns the event is not waiting for this event and not suspended	Event is set
40	Expiration of alarm which sets an event while running task is non-preemptive. Task which owns the event is waiting for this event	Event is set. Task which is owner of the event becomes ready. No preemption of running task
41	Expiration of alarm which sets an event while running task is preemptive. Task which owns the event is not waiting for this event and not suspended	Event is set
42	Expiration of alarm which sets an event while running task is preemptive. Task which owns the event is waiting for this event	Event is set. Task which is owner of the event becomes ready. Task with highest priority is executed (Rescheduling)
43	Expiration of alarm which callback a routine	Running task becomes ready. Callback routine is activated.

2.6 Error handling, hook routines (with interrupts) and OS execution control

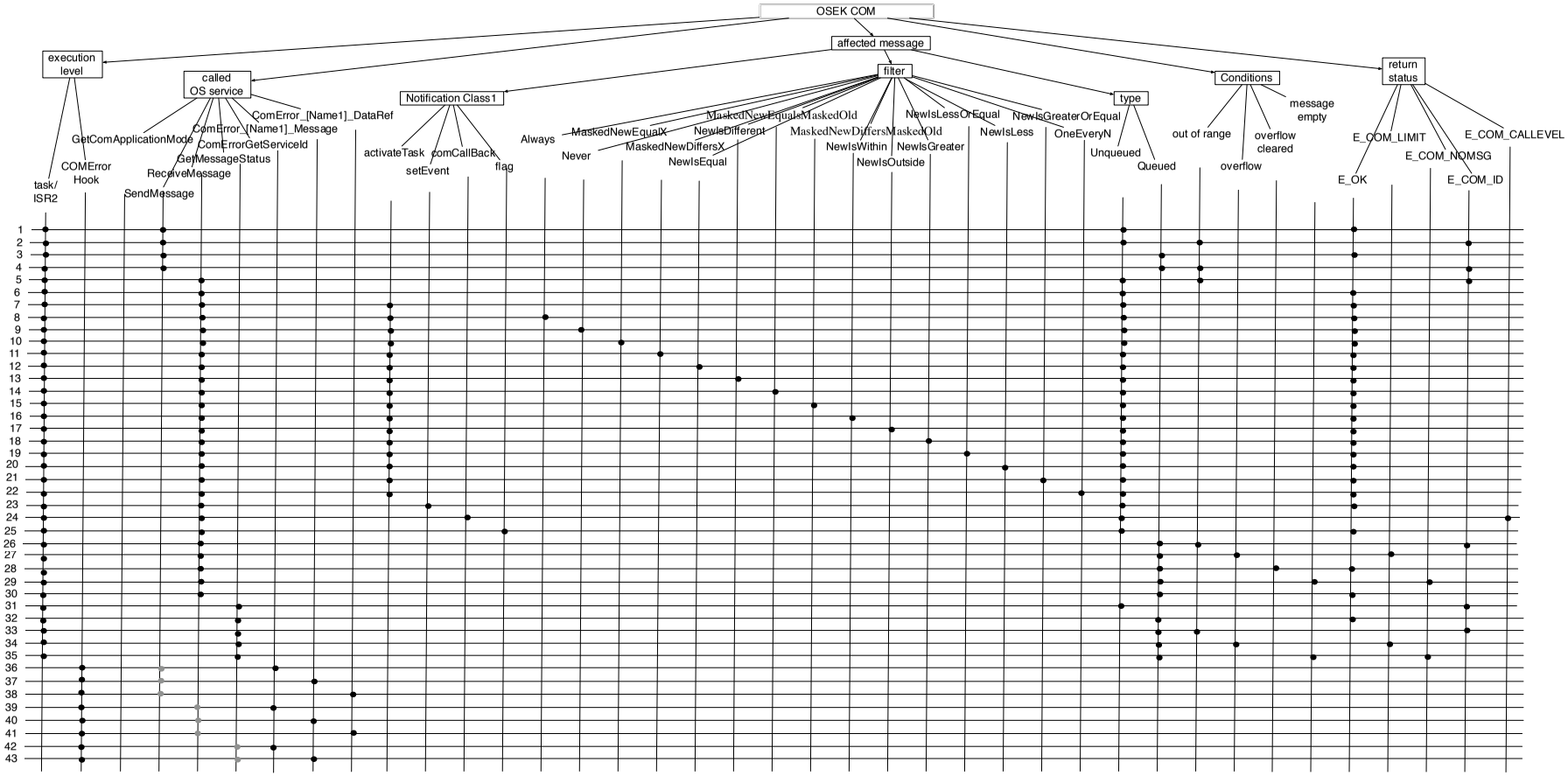
The specification doesn't provide an error status when calling an OS service which is not allowed on hook level from inside a hook routine. It is assumed that the correct behaviour would be to return E_OS_CALLEVEL. As this is not prescribed by the specification, this will not be used as a criteria for the conformance of the implementation. Anyway, the conformance tests will check that restricted OS services return a value not equal E_OK.
Compare to the previous Test Plan 2.0, it's forbidden to call ActivateTask() from StartupHook routine. SuspendAllInterrupts() and ResumeAllInterrupts() are allowed in hook routines.
See Annexe A for more information about interrupt management (test case from 15 to 32).

Test Case No.	Action	Expected Result
1	Call GetActiveApplicationMode()	Return current application mode
2	Call StartOS()	Start operating system
3	Call ShutdownOS()	Shutdown operating system
4	Check PreTaskHook/PostTaskHook: Force rescheduling	PreTaskHook is called before executing the new task, but after the transition to running state. PostTaskHook is called after exiting the current task but before leaving the task's running state
5	Check ErrorHook: Force error	ErrorHook is called at the end of a system service which has a return value not equal E_OK
6	Check StartupHook: Start OS	StartupHook is called after initialisation of OS
7	Check ShutdownHook: Shutdown OS	ShutdownHook is called after the OS shutdown
	Check availability of OS services inside hook routines according to fig 12-1 of OS spec.	*OS services which must not be called from hook routines return status not equal E_OK*
8	Call GetTaskID() from ErrorHook, PreTaskHook and PostTaskHook	Return E_OK
9	Call GetTaskState() from ErrorHook, PreTaskHook and PostTaskHook	Return E_OK if TaskID is valid
10	Call SuspendAllInterrupts() from ErrorHook, PreTaskHook and PostTaskHook	
11	Call ResumeAllInterrupts() from ErrorHook, PreTaskHook and PostTaskHook	
12	Call GetEvent() from ErrorHook, PreTaskHook and PostTaskHook	Return E_OK if TaskID is valid, Referenced task <TaskID> is an extended task and not in suspended state.
13	Call GetAlarmBase() from ErrorHook, PreTaskHook and PostTaskHook	Return E_OK if AlarmID is valid
14	Call GetAlarm() from ErrorHook, PreTaskHook and PostTaskHook	Return E_OK if AlarmID is valid and used
Interrupt processing in Hook routines :		
15	Interrupt activation in PostTaskHook of a task preempted by an alarm which activate a task.	
16	Interrupt activation in PreTaskHook of a task preempted by an alarm which activate a task.	
17	Interrupt activation in PostTaskHook of a task preempted by an ISR2.	
18	Interrupt activation in PreTaskHook of a task preempted by an ISR2.	
19	Interrupt activation in PostTaskHook of a task activated by an task (preempted or not).	
20	Interrupt activation in PreTaskHook of a task activated by an task (preempted or not).	
21	Interrupt activation in PostTaskHook of a task activated by an alarm which will give back the hand to the previous running task.	
22	Interrupt activation in PreTaskHook of a task activated by an alarm which will give back the hand to the previous running task.	
23	Interrupt activation in PostTaskHook of an ISR2 which will give back the hand to the previous running task.	
24	Interrupt activation in PreTaskHook of an ISR2 which will give back the hand to the previous running task.	
25	Interrupt triggering with an activation in PostTaskHook of a task preempted by an alarm which activate a task.	
26	Interrupt triggering with an activation in PreTaskHook of a task preempted by an alarm which activate a task.	
27	Interrupt triggering with an activation in PostTaskHook of a task preempted by an ISR2.	
28	Interrupt triggering with an activation in PreTaskHook of a task preempted by an ISR2.	
29	Interrupt triggering with an activation in PostTaskHook of a task followed by an task (preempted or not).	
30	Interrupt triggering with an activation in PreTaskHook of a task followed by an task (preempted or not).	
31	Interrupt triggering with an activation in PostTaskHook of a task activated by an alarm which will give back the hand to the previous running task.	

Test Case No.	Action	Expected Result
32	Interrupt triggering with an activation in PreTaskHook of a task activated by an alarm which will give back the hand to the previous running task.	
33	Interrupt triggering with an activation in PostTaskHook of an ISR2 which will give back the hand to the previous running task.	
34	Interrupt triggering with an activation in PreTaskHook of an ISR2 which will give back the hand to the previous running task.	
35	Interrupt activation in ErrorHook.	
36	Interrupt triggering with an activation in ErrorHook.	

2.7 Internal COM

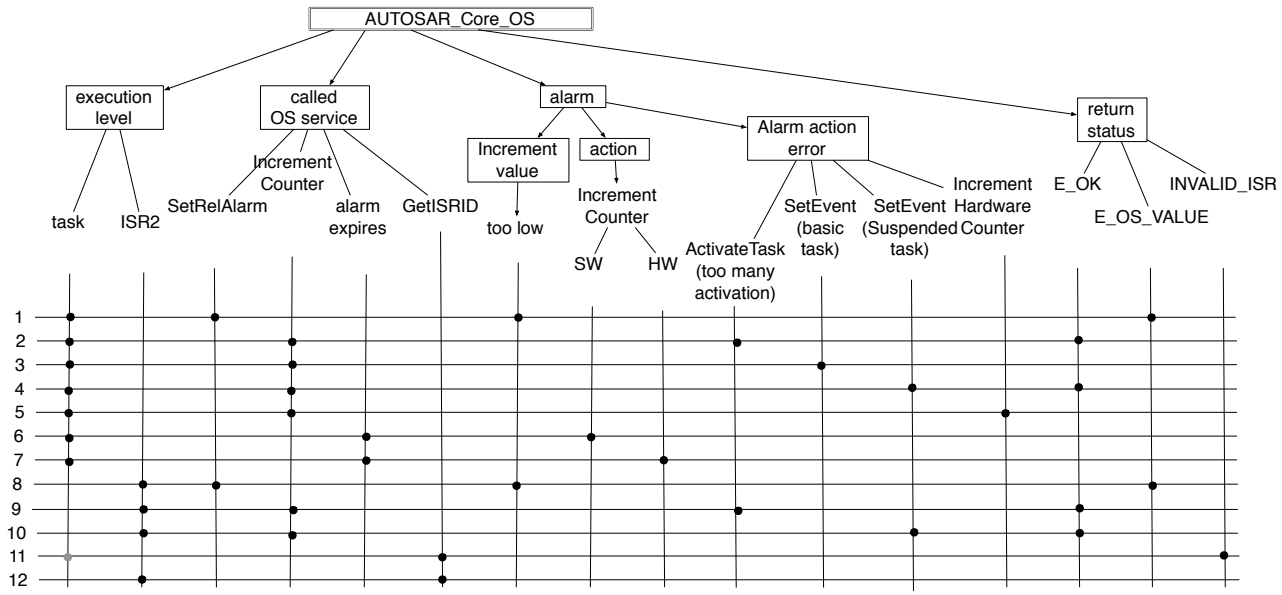
Test Case No.	Action	Expected Result
1	Call SendMessage() to an unqueued message	Service returns E_OK
2	Call SendMessage() to an unqueued message with <Message> out of range	Service returns E_COM_ID
3	Call SendMessage() to a queued message	Service returns E_OK
4	Call SendMessage() to a queued message with <Message> out of range	Service returns E_COM_ID
5	Call ReceiveMessage() to an unqueued message with <Message> out of range	Service returns E_COM_ID
6	Call ReceiveMessage() to an unqueued message	Service returns E_OK
7	Call ReceiveMessage() to an unqueued message with a notification which activate a task	Service returns E_OK
8	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "always" filter	Service returns E_OK
9	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "never" filter	Service returns E_OK
10	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "MaskedNewEqualX" filter	Service returns E_OK
11	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "MaskedNewDiffersX" filter	Service returns E_OK
12	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "NewIsEqual" filter	Service returns E_OK
13	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "NewIsDifferent" filter	Service returns E_OK
14	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "MaskedNewEqualsMaskedOld" filter	Service returns E_OK
15	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "MaskedNewEqualsMaskedOld" filter	Service returns E_OK
16	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "NewIsWithin" filter	Service returns E_OK
17	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "NewIsOutside" filter	Service returns E_OK
18	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "NewIsGreater" filter	Service returns E_OK
19	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "NewIsLessOrEqual" filter	Service returns E_OK


OSEK COM
execution level
task/ ISR2
COMError Hook
called OS service
GetComApplicationMode
ReceiveMessage
SendMessage
ComError_[Name1]_Message
ComErrorGetServiceId
GetMessageStatus
ComError_[Name1]_DataRef
Notification Class1
activateTask
setEvent
comCallBack
flag
affected message
filter
Always
Never
MaskedNewEqualX
MaskedNewDiffersX
NewIsDifferent
NewIsEqual
MaskedNewEqualsMaskedOld
MaskedNewDiffersMaskedOld
NewIsWithin
NewIsOutside
NewIsLessOrEqual
NewIsGreater
NewIsLess
NewIsGreaterOrEqual
OneEveryN
type
Unqueued
Queued
Conditions
out of range
overflow
overflow cleared
message empty
return status
E_OK
E_COM_LIMIT
E_COM_NOMSG
E_COM_ID
E_COM_CALLEVEL

Test Case No.	Action	Expected Result
20	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "NewIsLess" filter	Service returns E_OK
21	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "NewIsGreaterOrEqual" filter	Service returns E_OK
22	Call ReceiveMessage() to an unqueued message with a notification which activate a task and a "OneEveryN" filter	Service returns E_OK
23	Call ReceiveMessage() to an unqueued message with a notification which set an event	Service returns E_OK
24	Call ReceiveMessage() to an unqueued message with a notification which callback a routine	Service returns E_COM_CALLEVEL
25	Call ReceiveMessage() to an unqueued message with a notification which set a flag	Service returns E_OK
26	Call ReceiveMessage() to a queued message with <Message> out of range	Service returns E_COM_ID
27	Call ReceiveMessage() to a queued message which had an overflow on last SendMessage	Service returns E_COM_LIMIT and reset the overflow flag
28	Call ReceiveMessage() to a queued message which had an overflow cleared on last call to ReceiveMessage	Service returns E_OK
29	Call ReceiveMessage() to a queued message which is empty	Service returns E_COM_NOMSG
30	Call ReceiveMessage() to a queued message	Service returns E_OK
31	Call GetMessageStatus() to an unqueued message	Service returns E_COM_ID
32	Call GetMessageStatus() to a queued message	Service returns E_OK
33	Call GetMessageStatus() to a queued message with <Message> out of range	Service returns E_COM_ID
34	Call GetMessageStatus() to a queued message which had an overflow on last SendMessage	Service returns E_COM_LIMIT
35	Call GetMessageStatus() to a queued message which is empty	Service returns E_COM_NOMSG
36	Call ComErrorGetServiceId() from ComErrorHook with SendMessage error	Service returns COMServiceId_SendMessage
37	Call ComError_SendMessage_Message from ComErrorHook	Service returns <Message> used in last SendMessage
38	Call ComError_SendMessage_DataRef from ComErrorHook	Service returns <DataRef> used in last SendMessage
39	Call ComErrorGetServiceId() from ComErrorHook with ReceiveMessage error	Service returns COMServiceId_ReceiveMessage
40	Call ComError_ReceiveMessage_Message from ComErrorHook	Service returns <Message> used in last ReceiveMessage
41	Call ComError_ReceiveMessage_DataRef from ComErrorHook	Service returns <DataRef> used in last ReceiveMessage
42	Call ComErrorGetServiceId() from ComErrorHook with GetMessageStatus error	Service returns COMServiceId_GetMessageStatus
43	Call ComError_GetMessageStatus_Message from ComErrorHook	Service returns <Message> used in last GetMessageStatus

2.8 AUTOSAR - Core OS

OS Requirements : 263*, 264*, 285, 301, 304, 321
Test cases 3 and 5 are GOIL test cases. Test case 7 is impossible to test.

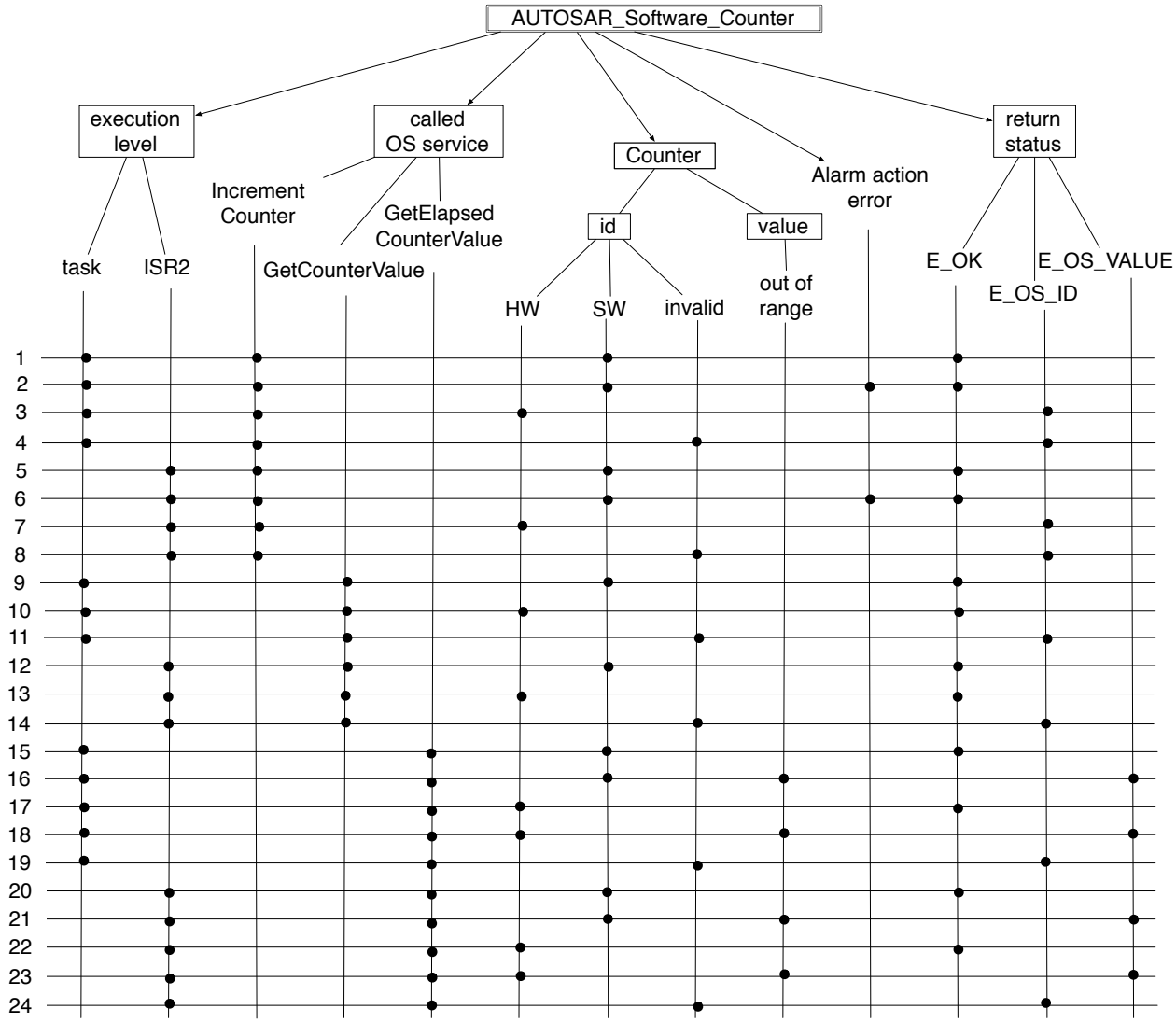


Test Case No.	Action	Expected Result	OS Requirements
1	Call SetRelAlarm() from task with <increment> value equal to zero	Service returns E_OS_VALUE	OS304
2	Call IncrementCounter() of a software counter from task (alarm action results in an error : ActivateTask() on a task which has already its max number of activation)	Errorhook is called. Service returns E_OK	OS321
3	It is impossible to call IncrementCounter() setting an event from an alarm expiration to a basic task.	error : An alarm can't set an Event to a basic task (Task t1 is a basic task).	OS321
4	Call IncrementCounter() of a software counter from task (alarm action results in an error : SetEvent() on a task is suspended)	Errorhook is called. Service returns E_OK	OS321
5	It is impossible to call IncrementCounter() incrementing a hardware counter from an alarm expiration.	error : It is impossible to increment a hardware counter (Z is not a software counter).	OS285
6	Expiration of alarm which increment a software counter	Software counter is incremented and alarm(s) is(are) launched if needed	OS301
7	Increment a hardware counter from an alarm expiration is impossible. GOIL generation should forbid to create an alarm which increment a hardware counter		
8	Call SetRelAlarm() from ISR2 with <increment> value equal to zero	Service returns E_OS_VALUE	OS304
9	Call IncrementCounter() of a software counter from ISR2 (alarm action results in an error : ActivateTask() on a task which has already its max number of activation)	Errorhook is called. Service returns E_OK	OS321
10	Call IncrementCounter() of a software counter from ISR2 (alarm action results in an error : SetEvent() on a task is suspended)	Errorhook is called. Service returns E_OK	OS321
11	Call GetISRID() from an other object than ISR2 or Hook routine called inside an ISR2	Service returns INVALID_ISR	OS264

Test Case No.	Action	Expected Result	OS Requirements
12	Call GetISRID() from an ISR2	Service returns the identifier of the currently running ISR2	OS263

2.9 AUTOSAR - Software Counter

OS Requirements : 285, 286, 321,376, 377, 381, 382, 383, 391, 392, 399, 460
OS374 and OS384 are indirectly tested thanks to the good fonctionning of the counter.



Test Case No.	Action	Expected Result	OS Requirements
1	Call IncrementCounter() of a software counter from task	Service returns E_OK	OS286, OS399

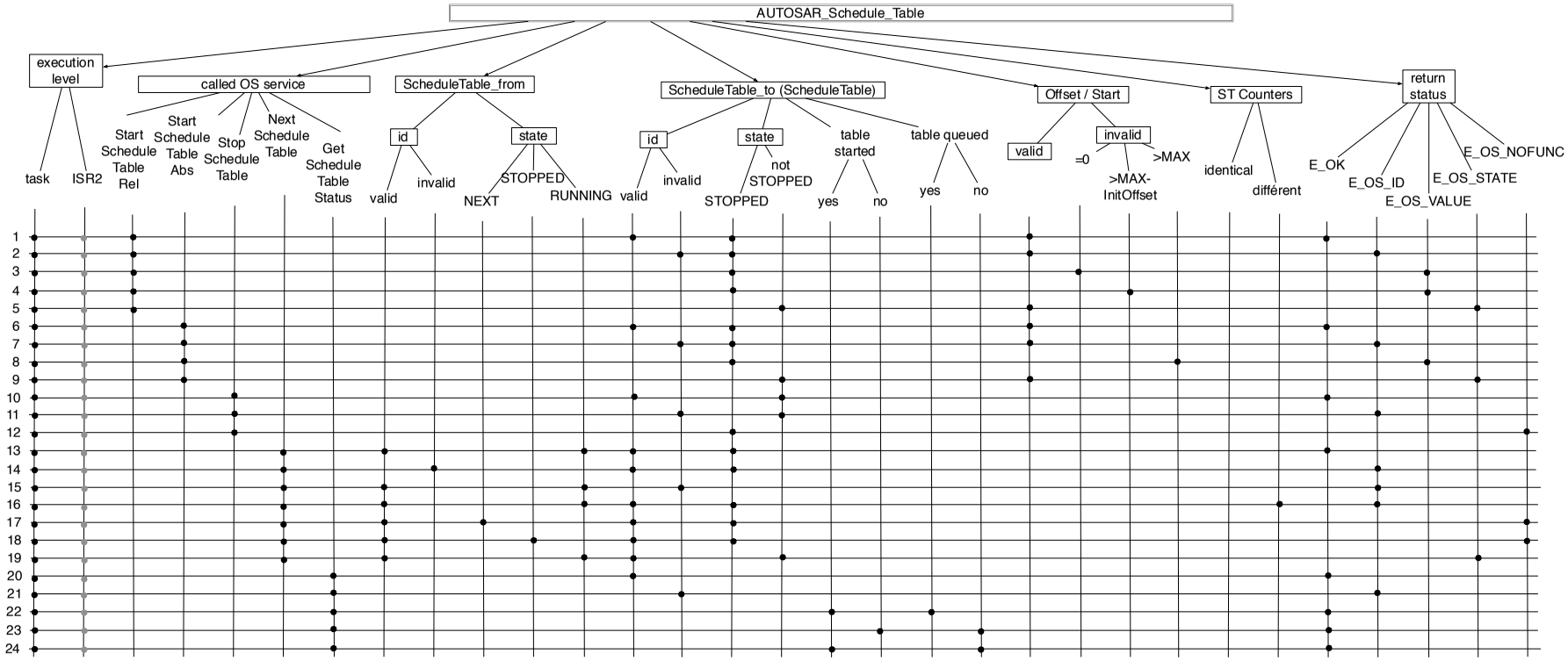
Test Case No.	Action	Expected Result	OS Requirements
2	Call IncrementCounter() of a software counter from task (alarm action results in an error)	Errorhook is called. Service returns E_OK	OS321
3	Call IncrementCounter() of a hardware counter from task	Service returns E_OS_ID	OS285
4	Call IncrementCounter() from task with invalid ID	Service returns E_OS_ID	OS285
5	Call IncrementCounter() of a software counter from ISR2	Service returns E_OK	
6	Call IncrementCounter() of a software counter from ISR2 (alarm action results in an error)	Errorhook is called. Service returns E_OK	
7	Call IncrementCounter() of a hardware counter from ISR2	Service returns E_OS_ID	
8	Call IncrementCounter() from ISR2 with invalid ID	Service returns E_OS_ID	
9	Call GetCounterValue() of a sofwtare counter from task	Service returns E_OK and <Value> of the counter	OS377, OS383
10	Call GetCounterValue() of a hardware counter from task	Service returns E_OK and <Value> of the counter	OS377, OS383
11	Call GetCounterValue() from task with invalid ID	Service returns E_OS_ID	OS376
12	Call GetCounterValue() of a sofwtare counter from ISR2	Service returns E_OK and <Value> of the counter	
13	Call GetCounterValue() of a hardware counter from ISR2	Service returns E_OK and <Value> of the counter	
14	Call GetCounterValue() from ISR2 with invalid ID	Service returns E_OS_ID	
15	Call GetElapsedCounterValue() of a software counter from task	Service returns E_OK, the <Value> of the counter and the number of elapsed ticks since the given <Value> value via <ElapsedValue>	OS382, OS392, OS460
16	Call GetElapsedCounterValue() of a software counter from task with <Value> out of range	Service returns E_OS_VALUE	OS391
17	Call GetElapsedCounterValue() of a hardware counter from task	Service returns E_OK, the <Value> of the counter and the number of elapsed ticks since the given <Value> value via <ElapsedValue>	OS382, OS392, OS460
18	Call GetElapsedCounterValue() of a hardware counter from task with <Value> out of range	Service returns E_OS_VALUE	OS391
19	Call GetElapsedCounterValue() from task with invalid ID	Service returns E_OS_ID	OS381
20	Call GetElapsedCounterValue() of a software counter from ISR2	Service returns E_OK, the <Value> of the counter and the number of elapsed ticks since the given <Value> value via <ElapsedValue>	
21	Call GetElapsedCounterValue() of a software counter from ISR2 with <Value> out of range	Service returns E_OS_VALUE	
22	Call GetElapsedCounterValue() of a hardware counter from ISR2	Service returns E_OK, the <Value> of the counter and the number of elapsed ticks since the given <Value> value via <ElapsedValue>	
23	Call GetElapsedCounterValue() of a hardware counter from ISR2 with <Value> out of range	Service returns E_OS_VALUE	
24	Call GetElapsedCounterValue() from ISR2 with invalid ID	Service returns E_OS_ID	

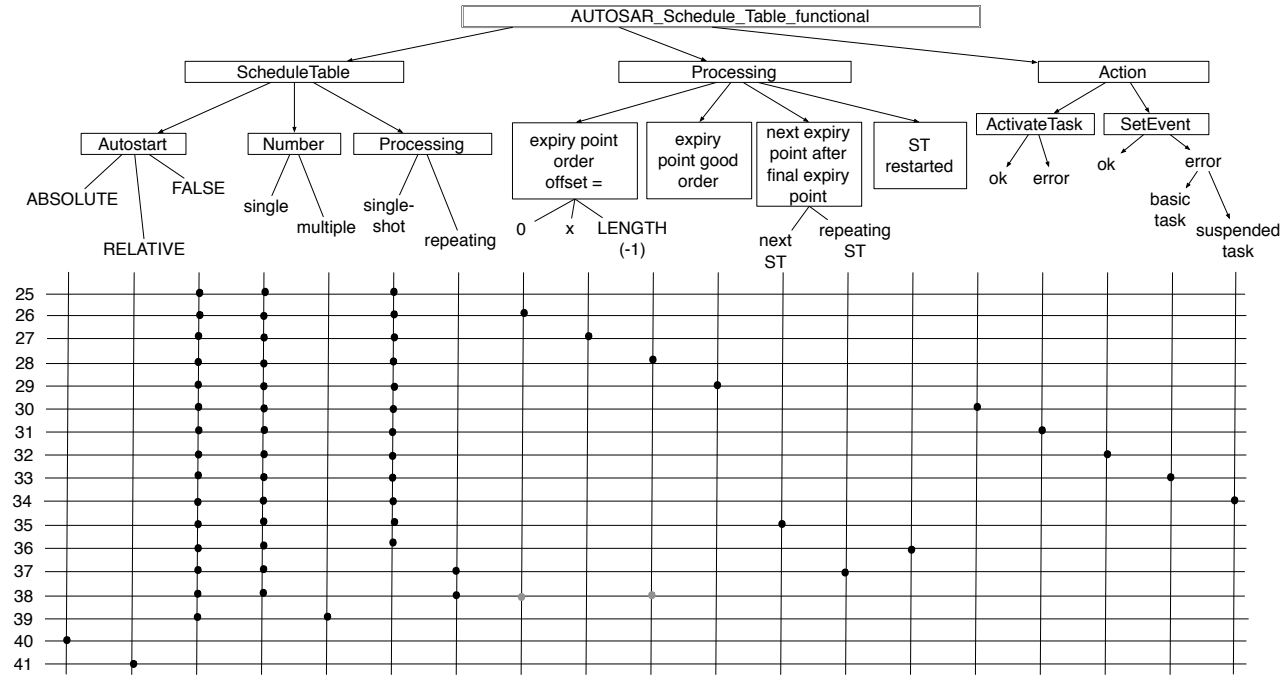
Test Case No.	Action	Expected Result	OS Requirements

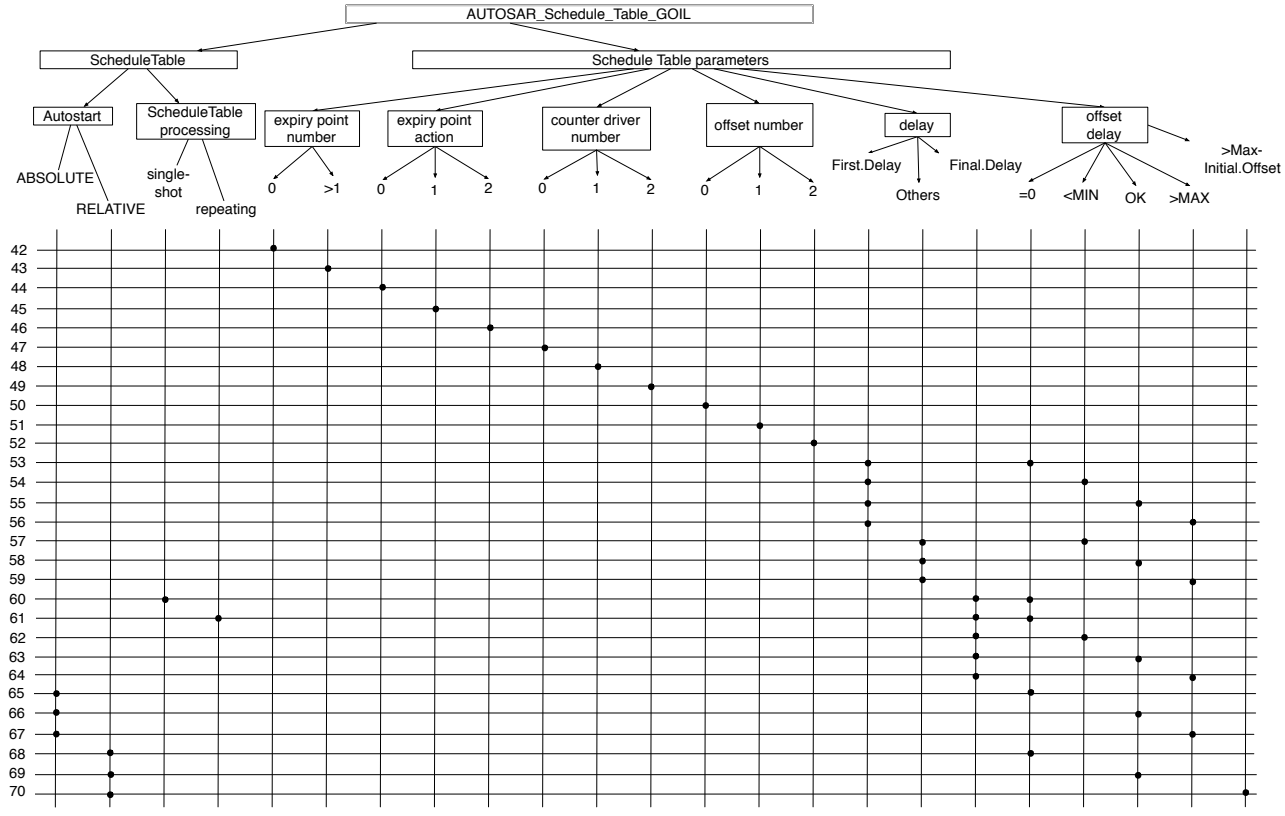
2.10 AUTOSAR - Schedule Table

OS Requirements : 002, 006, 007, 009, 191, 194, 275, (276), 277, 278, 279, 280, 281, 282, 283, 284, 289, 291, 293, 309, 324, 330, 332, 347, 348, 349, 350, 351, 353, 358, 359, 410, 412, 414, 428, 453.
OS Requirements 401, 402, 403, 404, 407, 408, 409, 427, 442, 443, 444 are GOIL test cases (Test cases 33 to 42 and 70).
OS411 can't be tested. As a schedule table is automatically set to single-shot if not specified, OS413 can't be tested.

Test Case No.	Action	Expected Result	OS Requirements
1	Call StartScheduleTableRel() from task	Service returns E_OK	OS278, OS358
2	Call StartScheduleTableRel() from task with invalid id	Service returns E_OS_ID	OS275
3	Call StartScheduleTableRel() from task with <offset> value equal to zero	Service returns E_OS_VALUE	OS332
4	Call StartScheduleTableRel() from task with <offset> > (MAXALLOWEDVALUE - InitialOffset)	Service returns E_OS_VALUE	OS276
5	Call StartScheduleTableRel() from task when schedule table is not in state SCHEDULETABLE_STOPPED	Service returns E_OS_STATE (in STANDARD and EXTENDED)	OS277
6	Call StartScheduleTableAbs() from task	Service returns E_OK	OS347, OS351
7	Call StartScheduleTableAbs() from task with invalid id	Service returns E_OS_ID	OS348
8	Call StartScheduleTableAbs() from task with <offset> > (MAXALLOWEDVALUE)	Service returns E_OS_VALUE	OS349
9	Call StartScheduleTableAbs() from task when schedule table is in state SCHEDULETABLE_STOPPED	Service returns E_OS_STATE (in STANDARD and EXTENDED)	OS350
10	Call StopScheduleTable() from task	Service returns E_OK	OS006 OS281, OS453
11	Call StopScheduleTable() from task with invalid id	Service returns E_OS_ID	OS279
12	Call StopScheduleTable() from task when schedule table is in state SCHEDULETABLE_STOPPED	Service returns E_OS_NOFUNC (in STANDARD and EXTENDED)	OS280
13	Call NextScheduleTable() from task	Service returns E_OK	OS191, OS284, OS324, OS414
14	Call NextScheduleTable() from task with invalid ScheduleTableID_From	Service returns E_OS_ID	OS282
15	Call NextScheduleTable() from task with invalid ScheduleTableID_To	Service returns E_OS_ID	OS282
16	Call NextScheduleTable() from task with different schedule table counters	Service returns E_OS_ID	OS330
17	Call NextScheduleTable() from task when schedule table "from" is in state SCHEDULETABLE_NEXT	Service returns E_OS_NOFUNC (in STANDARD and EXTENDED)	OS283
18	Call NextScheduleTable() from task when schedule table "from" is in state SCHEDULETABLE_STOPPED	Service returns E_OS_NOFUNC (in STANDARD and EXTENDED)	OS283
19	Call NextScheduleTable() from task when schedule table "to" is not in state SCHEDULETABLE_STOPPED	Service returns E_OS_STATE	OS309
20	Call GetMessageStatus() from task	Service returns E_OK	OS359
21	Call GetMessageStatus() from task with invalid id	Service returns E_OS_ID	OS293
22	Call GetMessageStatus() from task for a schedule table which waits for the end of the current schedule table	Service returns E_OK and SCHEDULETABLE_NEXT via <ScheduleStatus>	OS353


AUTOSAR_Schedule_Table
execution level
task
ISR2
called OS service
Start Schedule Table Rel
Start Schedule Table Abs
Stop Schedule Table
Next Schedule Table
Get Schedule Table Status
ScheduleTable_from
id
valid
invalid
state
NEXT
STOPPED
RUNNING
ScheduleTable_to (ScheduleTable)
id
valid
invalid
state
STOPPED
not STOPPED
table started
yes
no
table queued
yes
no
Offset / Start
valid
invalid
=0
>MAX-InitOffset
>MAX
ST Counters
identical
different
return status
E_OK
E_OS_ID
E_OS_VALUE
E_OS_STATE
E_OS_NOFUNC
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24


AUTOSAR_Schedule_Table_functional
ScheduleTable
Processing
Action
Autostart
Number
Processing
expiry point order offset =
expiry point good order
next expiry point after final expiry point
ST restarted
ActivateTask
SetEvent
ABSOLUTE
RELATIVE
FALSE
single
multiple
single-shot
repeating
0
x
LENGTH (-1)
next ST
repeating ST
ok
error
ok
error
basic task
suspended task
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41


AUTOSAR_Schedule_Table_GOIL
ScheduleTable
Schedule Table parameters
Autostart
ScheduleTable processing
expiry point number
expiry point action
counter driver number
offset number
delay
offset delay
ABSOLUTE
RELATIVE
single-shot
repeating
0
>1
0
1
2
0
1
2
0
1
2
First.Delay
Final.Delay
Others
=0
<MIN
OK
>MAX
>Max-Initial.Offset
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70

Test Case No.	Action	Expected Result	OS Requirements
23	Call GetMessageStatus() from task for a schedule table which is not started	Service returns E_OK and SCHEDULETABLE_STOPPED via <ScheduleStatus>	OS289
24	Call GetMessageStatus() from task for a schedule table which is started	Service returns E_OK and SCHEDULETABLE_RUNNING via <ScheduleStatus>	OS291
25	If single-shot ST, stop the schedule table Final Delay ticks after the Final Expiry Point is processed		OS009
26	If single-shot ST, an expiry point can be set to offset=0		OS002
27	The schedule table has to be processed from the InitialExpiryPoint to the FinalExpiryPoint in order of increasing offset		OS002, OS410
28	If single-shot ST, an expiry point can be set to offset=LENGTH		OS002
29	If single-shot ST, The OS shall process all task activations on an expiry point first and then set events		OS412
30	Action of a ST results in a ActivateTask		
31	Action of a ST results in a ActivateTask and and overflow of Activation occurs.	ErrorHook is launched	
32	Action of a ST results in a SetEvent		
33	Action of a ST results in a SetEvent on a basic task.	error : An action can't set an Event to a basic task (Task t1 is a basic task).	
34	Action of a ST results in a SetEvent on a suspended task.	ErrorHook is launched	
35	If single-shot ST, Intial expiry point of a 'nexted' ST shall be launched at Final Expiry point + Final Delay + Initial Expiry point		OS414
36	A ST restarts from the begging (offset=0)		OS428
37	If repeating ST, Initial Expiry Point shall be launched at Final Expiry Point + Final Delay + Initial Offset		OS194
38	If repeating ST, an expiry point can be set to offset=0 and at offset=LENGTH-1		OS002
39	Multiple ST are allowed		OS007
40	A ST can be autostarted with ABSOLUTE mode. <OFFSET> should be in the range MINCYCLE..MAXALLOWEDVALUE OR equal to 0		OsScheduleTableAutostart
41	A ST can be autostarted with RELATIVE mode. <START> should be in the range MINCYCLE..MAXALLOWEDVALUE		OsScheduleTableAutostart
42	No Expiry point in a schedule table	error : no EXPIRY_POINT found for SCHEDULETABLE X	OS401
43	One or several expiry points in a schedule table		OS401
44	No Action in an expiry point	error : no ACTION found for EXPIRY_POINT Y	OS407
45	One action in an expiry point		OS402, OS403
46	Several actions in an expiry point		OS407
47	No counter in a schedule table	error : Counter is not defined in X	OS409
48	One counter in a schedule table		OS409
49	Several counters in a schedule table	error : COUNTER attribute already defined for Schedule Table X	OS409
50	No offset in an expiry point	error : OFFSET is missing for expiry point Y	OS404

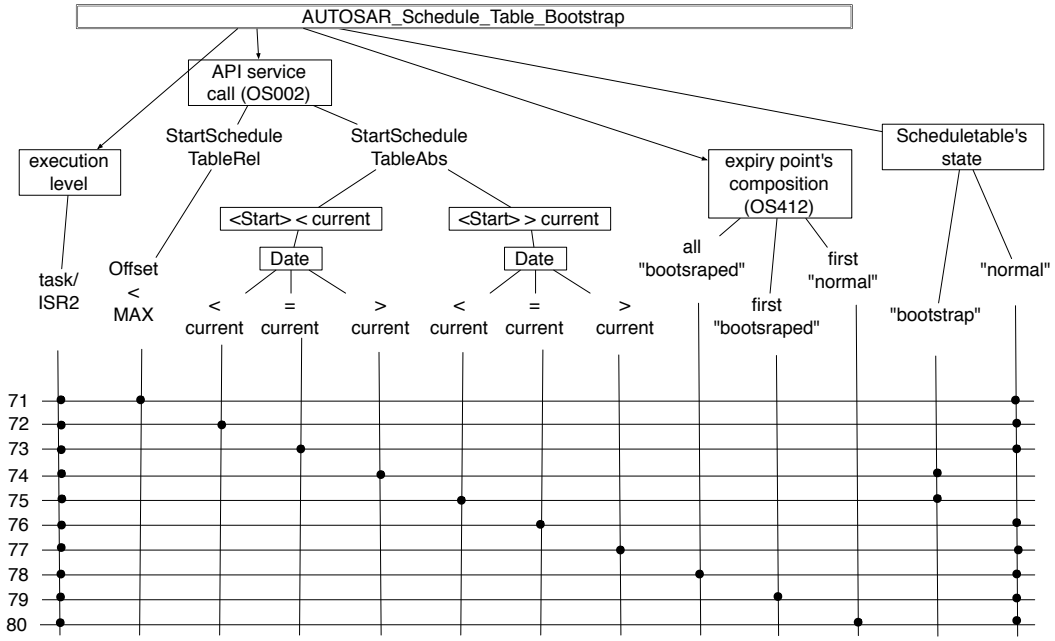
Test Case No.	Action	Expected Result	OS Requirements
51	One offset in an expiry point		OS442
52	Several offsets in an expiry point	error : OFFSET Redefinition	OS442
53	First.Delay is equal to 0		OS443
54	First.Delay is lower than MINCYCLE	error : OFFSET of first expiry point is lower than MINCYCLE of the driving counter and not equal to 0.	OS443
55	First.Delay is in the range		OS443
56	First.Delay is greater than MAXALLOWEDVALUE	error : OFFSET of first expiry point is greater than MAXALLOWEDVALUE of the driving counter	OS443
57	Delay between adjacent expiry point is lower than MINCYCLE	error : Delay between expiry point number A and B is lower than MINCYCLE of the driving counter	OS408
58	Delay between adjacent expiry point is in the range		OS408
59	Delay between adjacent expiry point is greater than MAXALLOWEDVALUE	error : Delay between expiry point number A and B is greater than MAXALLOWEDVALUE of the driving counter	OS408
60	In single-shot, Final.Delay is equal to 0		OS427
61	In repeating, Final.Delay is equal to 0	error : Final delay can be equal to 0 only for single-shot schedule table and X is a repeating one	OS444
62	Final.Delay is lower than MINCYCLE	error : Final delay should be within MINCYCLE and MAXALLOWEDVALUE of the driving counter	OS444
63	Final.Delay is in the range		OS444
64	Final.Delay is greater than MAXALLOWEDVALUE	error : Final delay should be within MINCYCLE and MAXALLOWEDVALUE of the driving counter	OS444
65	In an ABSOLUTE autostarted schedule table, <OFFSET> is equal to 0		
66	In an ABSOLUTE autostarted schedule table, <OFFSET> is lower than MAXALLOWEDVALUE		
67	In an ABSOLUTE autostarted schedule table, <OFFSET> is greater than MAXALLOWEDVALUE	error : X autostart's offset is greater than MAXALLOWEDVALUE	OS349
68	In an RELATIVE autostarted schedule table, <START> is equal to 0	error : X autostart's offset is equal to 0	OS332
69	In an RELATIVE autostarted schedule table, <START> is lower than (MAXALLOWEDVALUE - Initial.Offset)		
70	In an RELATIVE autostarted schedule table, <START> is greater than (MAXALLOWEDVALUE - Initial.Offset)	error : X autostart's offset is greater than (MAXALLOWEDVALUE - Initial.Offset)	OS276

When a schedule table is started, the first expiry point can be set to the "second" value of a counter tick (only with StartScheduleTableAbs) if :

- (<start> > current date) AND (<start> + FirstDelay - MAX_ALLOWED_VALUE) > current date
- (<start> < current date) AND ((<start> + FirstDelay) > current date)

Because of that, more tests has to be done to check that the expiry point is not launched at the first value of the

counter but at the "second". In Trampoline, we use a "Bootstrap" to implement the solution. A bit of the schedule table's state is set to '1' when the first expiry point has reached the conditions above. When the time object is launched, we take a look at the state and if the bit is '1', we take out the time object and place it before the current date, setting the bit to '0'. In this way, the expiry point is shifted to the "second" value of the counter.
Moreover, other tests have to check the correct functionning of the sequences when there are only "bootstraped" schedule table on an expiry point, or when there are "bootstraped" and "normal" schedule tabe, whatever the first inserted in the counter's date.
The plan below conclues on the schedule table tests. "Date" is the date of the first expiry point.



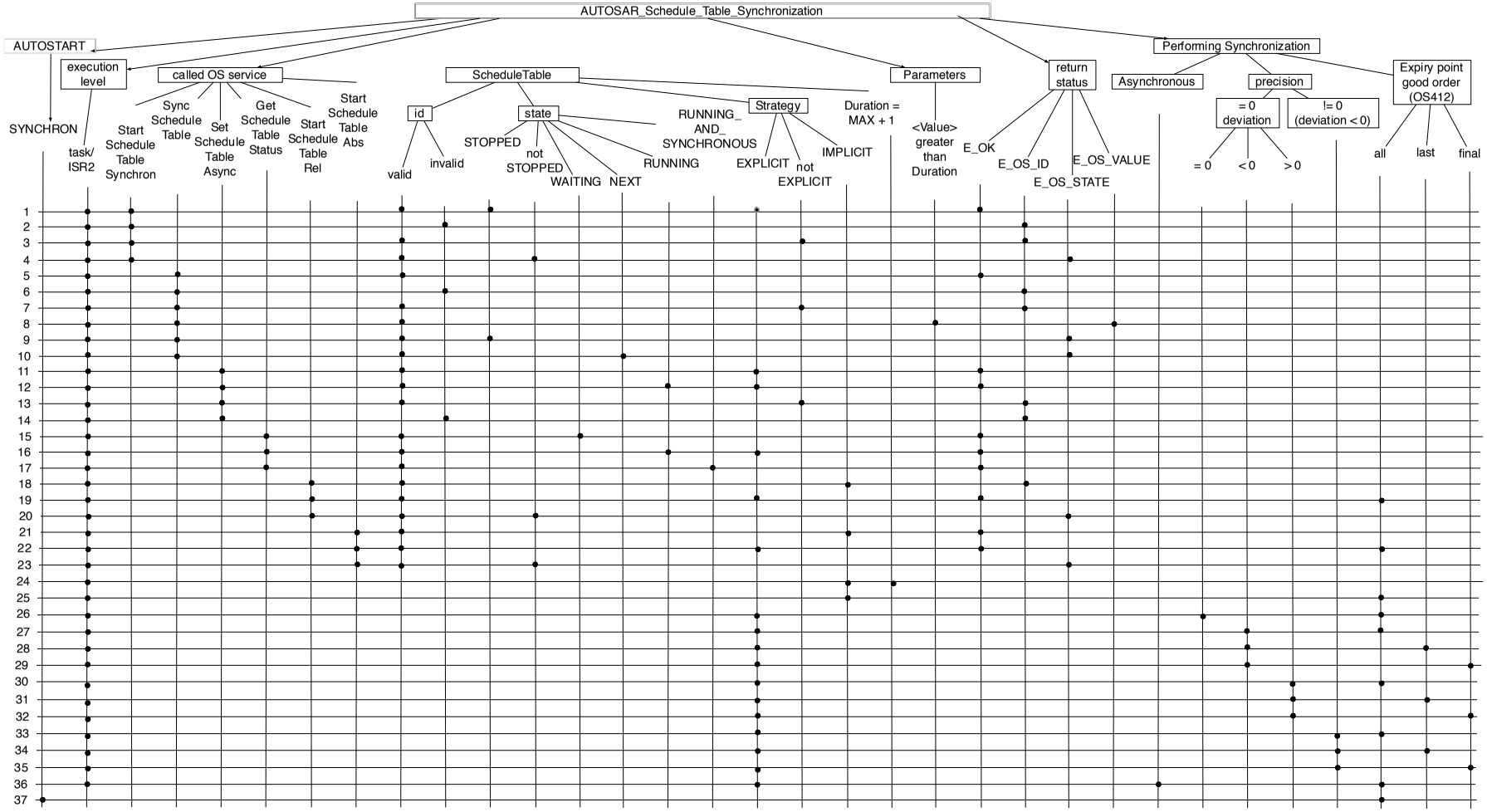
Test Case No.	Action	Expected Result
71	Call StartScheduleTableRel() from task. Offset is lower than max allowed value of the counter.	Service returns E_OK
72	Call StartScheduleTableAbs() from task. <Start> and Date are lower than current date.	Service returns E_OK
73	Call StartScheduleTableAbs() from task. <Start> is lower than current date and Date is equal to current date.	Service returns E_OK
74	Call StartScheduleTableAbs() from task. <Start> is lower than current date and Date is greater than current date.	Service returns E_OK. The schedule table is set to a "bootstrap" one.
75	Call StartScheduleTableAbs() from task. <Start> is greater than current date and Date is lower than current date.	Service returns E_OK
76	Call StartScheduleTableAbs() from task. <Start> is greater than current date and Date is equal to current date.	Service returns E_OK

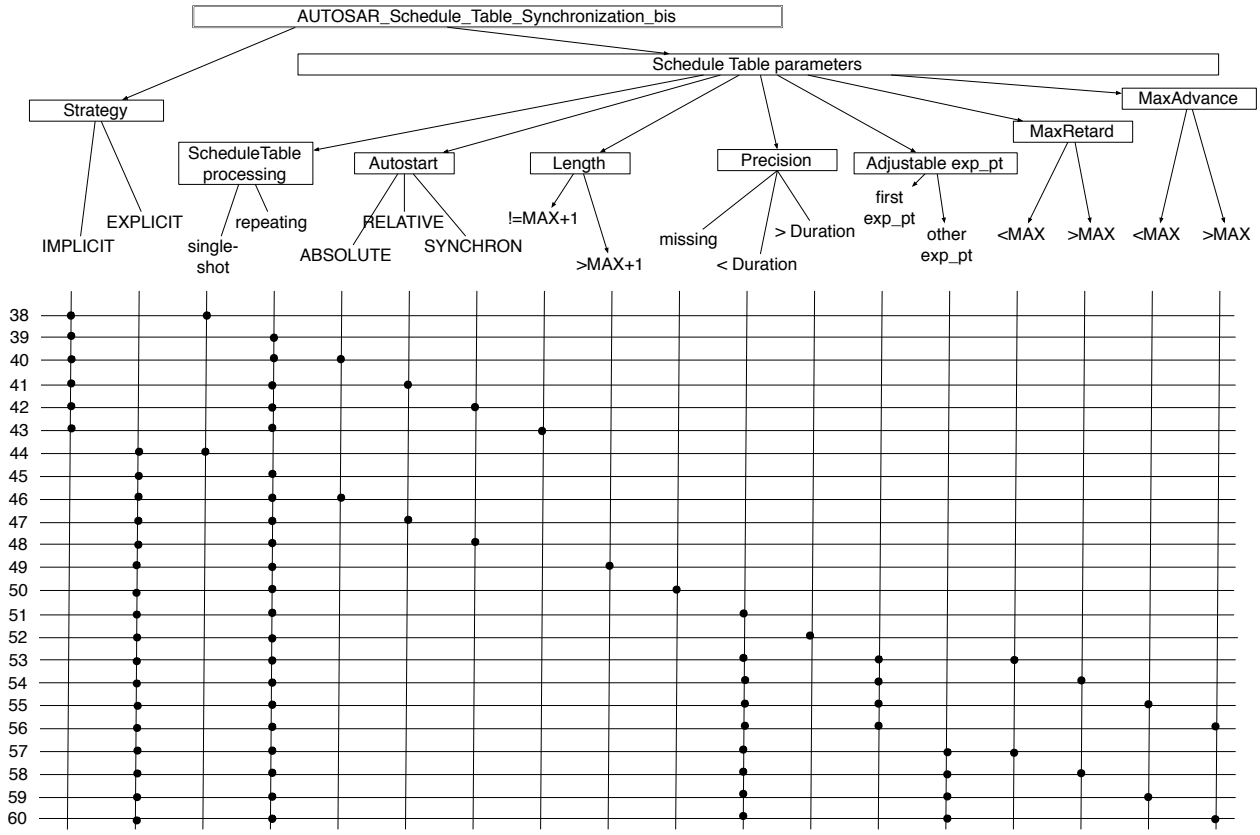
Test Case No.	Action	Expected Result
77	Call StartScheduleTableAbs() from task. <Start> and Date are greater than current date.	Service returns E_OK. The schedule table is set to a "bootstrap" one.
78	Set several "bootstraped" schedule table to a same date	Expiry points stay in the list and schedule table state becomes "normal"
79	Set several "bootstraped" and "normal" schedule table to a same date. A "bootstrap" schedule table is inserted first in the list.	Expiry points which was "bootstraped" stay in the list and there schedule table state becomes "normal". Expiry point which was "normal" are taken out of the list.
80	Set several "bootstraped" and "normal" schedule table to a same date. A "normal" schedule table is inserted first in the list.	Expiry points which was "bootstraped" stay in the list and there schedule table state becomes "normal". Expiry point which was "normal" are taken out of the list.

2.11 AUTOSAR - Schedule Table Synchronisation

OS Requirements : 013, 199, 201, 206, 227, 278, 290, 291, 300, 323, 351, 354, 362, (363), 387, 388, 389, 417, 418, 419, 420, 421, 422, 429, 430, 434, 435, 452, 454, 455, 456, 457, 458
OS462 and OS463 can't be tested.
OS Requirements 415, 416, 429, 430, 431, 436, 437, 438 are GOIL test cases (Test cases 38 to 60).

Test Case No.	Action	Expected Result	OS Requirements
1	Call StartScheduleTableSynchron() from task/ISR2. The state of the schedule table is equal to SCHEDULETABLE_STOPPED	Service returns E_OK, the state is set to SCHEDULETABLE_WAITING	OS389, OS435
2	Call StartScheduleTableSynchron() from task/ISR2 with invalid id	Service returns E_OS_ID	OS387
3	Call StartScheduleTableSynchron() from task/ISR2. The schedule table is not explicitly synchronized	Service returns E_OS_ID	OS387
4	Call StartScheduleTableSynchron() from task/ISR2. The state of the schedule table is not equal to SCHEDULETABLE_STOPPED	Service returns E_OS_STATE (in STANDARD and EXTENDED)	OS388
5	Call SyncScheduleTable() from task/ISR2.	Service returns E_OK, the processing of the schedule table is started	OS013, OS457, OS199, OS201
6	Call SyncScheduleTable() from task/ISR2 with invalid id	Service returns E_OS_ID	OS454
7	Call SyncScheduleTable() from task/ISR2. The schedule table is not explicitly synchronized	Service returns E_OS_ID	OS454
8	Call SyncScheduleTable() from task/ISR2. The <value> is greater than OSScheduleTableDuration	Service returns E_OS_VALUE	OS455
9	Call SyncScheduleTable() from task/ISR2. The state of the schedule table is equal to SCHEDULETABLE_STOPPED	Service returns E_OS_STATE	OS456
10	Call SyncScheduleTable() from task/ISR2. The state of the schedule table is equal to SCHEDULETABLE_NEXT	Service returns E_OS_STATE	OS456


AUTOSAR_Schedule_Table_Synchronization
AUTOSTART
SYNCHRON
execution level
task/ ISR2
called OS service
Start Schedule Table Synchron
Sync Schedule Table
Set Schedule Table Async
Get Schedule Table Status
Start Schedule Table Rel
Start Schedule Table Abs
ScheduleTable
id
valid
invalid
state
STOPPED
not STOPPED
WAITING
NEXT
RUNNING
RUNNING_ AND_ SYNCHRONOUS
Strategy
EXPLICIT
not EXPLICIT
IMPLICIT
Duration = MAX + 1
Parameters
<Value> greater than Duration
return status
E_OK
E_OS_ID
E_OS_VALUE
E_OS_STATE
Performing Synchronization
Asynchronous
precision
= 0 deviation
!= 0 (deviation < 0)
= 0
< 0
> 0
Expiry point good order (OS412)
all
last
final
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37



Test Case No.	Action	Expected Result	OS Requirements
11	Call SetScheduleTableAsync() from task/ISR2. The schedule table is explicitly synchronized	Service returns E_OK, the state is set to SCHEDULETABLE_RUNNING	OS300
12	Call SetScheduleTableAsync() from task/ISR2. The schedule table is explicitly synchronized and the state of the schedule table is equal to SCHEDULETABLE_RUNNING	Service returns E_OK, the synchronisation is stopped but expiry point are still processed	OS362, OS323, OS422
13	Call SetScheduleTableAsync() from task/ISR2. The schedule table's strategy is not equal to EXPLICIT	Service returns E_OS_ID	OS458
14	Call SetScheduleTableAsync() from task/ISR2 with invalid id	Service returns E_OS_ID	OS458
15	Call GetScheduleTableStatus() from task/ISR2. The schedule table is EXCPLICIT and no synchronisation count was provided	Service returns E_OK and SCHEDULETABLE_WAITING via <ScheduleStatus>	OS354, OS227
16	Call GetScheduleTableStatus() from task/ISR2. The schedule table is started AND NOT synchronous	Service returns E_OK and SCHEDULETABLE_RUNNING via <ScheduleStatus>	OS291
17	Call GetScheduleTableStatus() from task/ISR2. The schedule table is started AND synchronous (deviation in the precision interval)	Service returns E_OK and SCHEDULETABLE_RUNNING_AND_SYNCHRONOUS via <ScheduleStatus>	OS290

Test Case No.	Action	Expected Result	OS Requirements
18	Call StartScheduleTableRel() from task/ISR2. The schedule table's strategy is IMPLICIT	Service returns E_OS_ID	OS452, OS430
19	Call StartScheduleTableRel() from task/ISR2. The schedule table's strategy is EXPLICIT	Service returns E_OK, the processing of the schedule table is started and the state is SCHEDULETABLE_RUNNING	OS278, OS434
20	Call StartScheduleTableRel() from task/ISR2. The schedule table's strategy is EXPLICIT and its state is not stopped	Service returns E_OS_STATE	OS277
21	Call StartScheduleTableAbs() from task/ISR2. The schedule table's strategy is IMPLICIT	Service returns E_OK, the processing of the schedule table is started and the state is SCHEDULETABLE_RUNNING	OS351
22	Call StartScheduleTableAbs() from task/ISR2. The schedule table's strategy is EXPLICIT	Service returns E_OK, the processing of the schedule table is started and the state is SCHEDULETABLE_RUNNING	OS351, OS434
23	Call StartScheduleTableAbs() from task/ISR2. The schedule table's strategy is EXPLICIT and its state is not stopped	Service returns E_OS_STATE	OS350
24	An IMPLICIT schedule table shall have a period equal to (MAX_ALLOWED_VALUE + 1) of its counter		OS429
25	An IMPLICIT schedule table is always synchronized.	Next expiry point is inserted in the list	
26	No synchronisation with deviation equal to 0	Next expiry point is inserted in the list	OS389, OS201
27	Performing synchronisation with precision equal to 0 and deviation less than 0. Check expiry point good order	According to deviation and MaxRetard, Next expiry point is inserted in the list	OS206, OS417, OS420
28	Performing synchronisation with precision equal to 0 and deviation less than 0. Check expiry point good order on last expiry point	According to deviation and MaxRetard, First expiry point is adjusted and if comes before Final expiry point, Final expiry point is adjuted to the same offset of First expiry point and inserted in the list and First expiry point offset becomes 0	OS420
29	Performing synchronisation with precision equal to 0 and deviation less than 0. Check expiry point good order on final expiry point	According to deviation and MaxRetard, First expiry point is launched now if First.Delay equal to 0, otherwise if only one expiry point in the ST (the final one), adjust the Final expiry point, insert it in the list and First expiry point offset becomes 0 otherwise is adjusted and inserted in the list	OS420
30	Performing synchronisation with precision equal to 0 and deviation greater than 0. Check expiry point good order	According to deviation and MaxAdvance, Next expiry point is inserted in the list	OS421
31	Performing synchronisation with precision equal to 0 and deviation greater than 0. Check expiry point good order on last expiry point	According to deviation and MaxAdvance, First expiry point is adjusted and Final expiry point is inserted in the list	OS421
32	Performing synchronisation with precision equal to 0 and deviation greater than 0. Check expiry point good order on final expiry point	According to deviation and MaxAdvance, First expiry point is launched now if First.Delay equal to 0, otherwise is adjusted and inserted in the list	OS421
33	Performing synchronisation with precision different than 0 and deviation less than 0. Check expiry point good order	According to deviation, precision and MaxRetard, Next expiry point is inserted in the list	OS418, OS419

Test Case No.	Action	Expected Result	OS Requirements
34	Performing synchronisation with precision different than 0 and deviation less than 0. Check expiry point good order on last expiry point	According to deviation, precision and MaxRetard, First expiry point is adjusted and if comes before Final expiry point, Final expiry point is adjuted to the same offset of First expiry point and inserted in the list and First expiry point offset becomes 0	OS418, OS419
35	Performing synchronisation with precision different than 0 and deviation less than 0. Check expiry point good order on final expiry point	According to deviation, precision and MaxRetard, First expiry point is launched now if First.Delay equal to 0, otherwise if only one expiry point in the ST (the final one), adjust the Final expiry point, insert it in the list and First expiry point offset becomes 0 otherwise is adjusted and inserted in the list	OS418, OS419
36	No synchronisation if schedule table asynchronous	Next expiry point is inserted in the list	OS362, OS323
37	A schedule table can be autostarted with SYNCHRON mode	The state is SCHEDULETABLE_WAITING	OsScheduleTableAutostart
38	IMPLICIT schedule table is single-shot	A synchronized schedule table shall be repeating otherwise, synchronisation can't be done.	
39	IMPLICIT schedule table is repeating		
40	IMPLICIT schedule table autostarts in ABSOLUTE mode		
41	IMPLICIT schedule table autostarts in RELATIVE mode	An IMPLICIT schedule table should be started in Absolute mode only	OS430
42	IMPLICIT schedule table autostarts in SYNCHRON mode	An IMPLICIT schedule table should be started in Absolute mode only	OS430
43	IMPLICIT schedule table duration is different to MAXALLOWEDVALUE + 1	An IMPLICIT schedule table should have a duration equal to OSMAXALLOWEDVALUE + 1 of its counter.	OS429
44	EXPLICIT schedule table is single-shot	A synchronized schedule table shall be repeating otherwise, synchronisation can't be done.	
45	EXPLICIT schedule table is repeating		
46	EXPLICIT schedule table autostarts in ABSOLUTE mode		
47	EXPLICIT schedule table autostarts in RELATIVE mode		
48	EXPLICIT schedule table autostarts in SYNCHRON mode		
49	EXPLICIT schedule table duration is greater than MAXALLOWEDVALUE + 1	An EXPLICIT schedule table shouldn't have a duration greater than OSMAXALLOWEVALUE + 1 of its counter.	OS431
50	EXPLICIT schedule table precision missing	PRECISION attribute is missing	
51	EXPLICIT schedule table precision lower than duration		
52	EXPLICIT schedule table precision greater than duration	An explicit schedule table shall have a precision in the range 0 to duration.	OS438
53	In the first expiry point of an EXPLICIT schedule table, MaxRetard is lower than the maximum value allowed		
54	In the first expiry point of an EXPLICIT schedule table, MaxRetard is greater than the maximum value allowed	In first expiry point, MaxRetard should be inferior to the previous delay minus MINCYCLE of the counter.	OS415, OS436

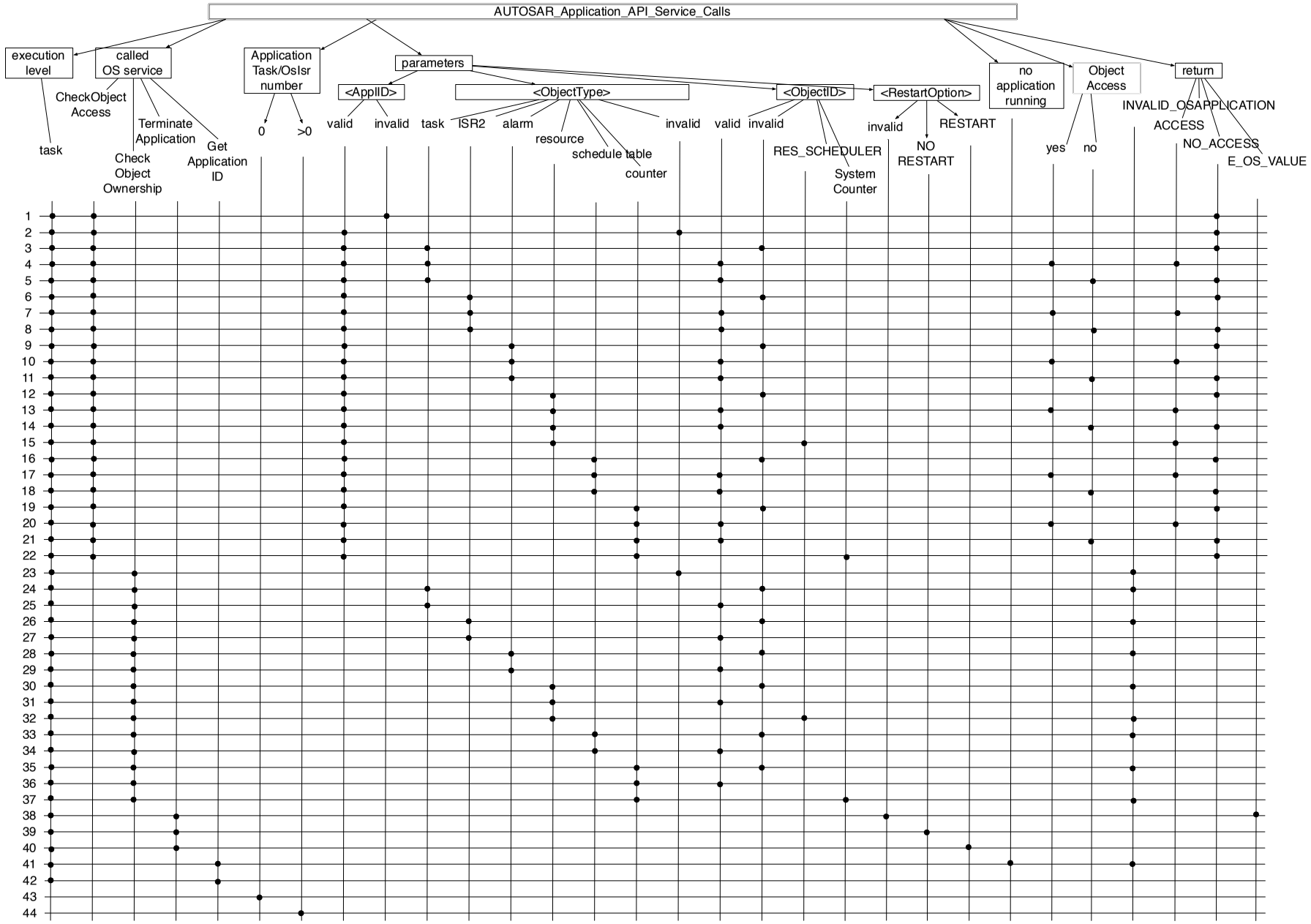
Test Case No.	Action	Expected Result	OS Requirements
55	In the first expiry point of an EXPLICIT schedule table, MaxAdvance is lower than the maximum value allowed		
56	In the first expiry point of an EXPLICIT schedule table, MaxAdvance is greater than the maximum value allowed	In first expiry point, MaxAdvance should be inferior to duration minus the first delay.	OS416, OS437
57	In an expiry point of an EXPLICIT schedule table, MaxRetard is lower than the maximum value allowed		
58	In an expiry point of an EXPLICIT schedule table, MaxRetard is greater than the maximum value allowed	In expiry point at offset X, MaxRetard should be inferior to the previous delay minus MINCYCLE of the counter.	OS415, OS436
59	In an expiry point of an EXPLICIT schedule table, MaxAdvance is lower than the maximum value allowed		
60	In an expiry point of an EXPLICIT schedule table, MaxAdvance is greater than the maximum value allowed	In expiry point at offset X, MaxAdvance should be inferior to duration minus the previous delay.	OS416, OS4337

2.12 AUTOSAR - OS-Application

2.12.1 API Service Calls for OS objects

OS Requirements : 016, 017, 256, 258, 261, 262, 271, 272, 273, 274, 287, 318, 319, 346, 423, 445, 447, 450, 459
OS288* is in the sequence which test all the API service calls from wrong context.

Test Case No.	Action	Expected Result	OS Requirements
1	Call CheckObjectAccess() with <AppID> invalid	Service returns NO_ACCESS	OS423
2	Call CheckObjectAccess() with <ObjectType> invalid	Service returns NO_ACCESS	OS423
3	Call CheckObjectAccess() for a task object type with <ObjectID> invalid	Service returns NO_ACCESS	OS423
4	Call CheckObjectAccess() for a task object type, *running* task/ISR2 has access to the object	Service returns ACCESS	OS256, OS271, OS450
5	Call CheckObjectAccess() for a task object type, *running* task/ISR2 has NO access to the object	Service returns NO_ACCESS	OS272
6	Call CheckObjectAccess() for an ISR2 object type with <ObjectID> invalid	Service returns NO_ACCESS	
7	Call CheckObjectAccess() for an ISR2 object type, *running* task/ISR2 has access to the object	Service returns ACCESS	
8	Call CheckObjectAccess() for an ISR2 object type, *running* task/ISR2 has NO access to the object	Service returns NO_ACCESS	


AUTOSAR_Application_API_Service_Calls
execution level
called OS service
Application Task/OsIsr number
parameters
no application running
Object Access
return
task
CheckObject Access
Check Object Ownership
Terminate Application
Get Application ID
0
>0
<AppID>
valid
invalid
<ObjectType>
task
ISR2
alarm
resource
schedule table
counter
<ObjectID>
invalid
valid
invalid
RES_SCHEDULER
System Counter
<RestartOption>
invalid
NO RESTART
RESTART
yes
no
INVALID_OSAPPLICATION
ACCESS
NO_ACCESS
E_OS_VALUE
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44

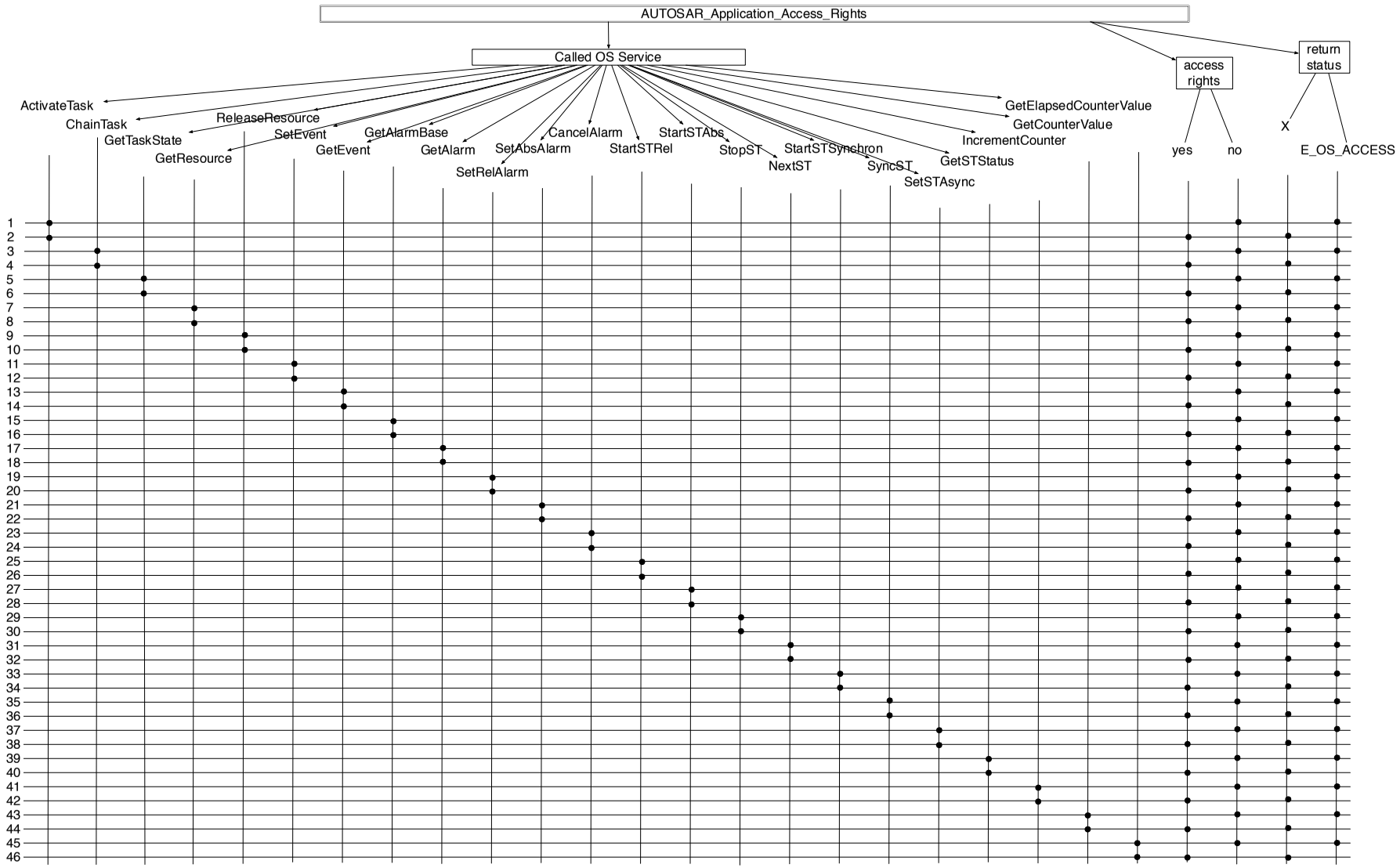
Test Case No.	Action	Expected Result	OS Requirements
9	Call CheckObjectAccess() for an alarm object type with <ObjectID> invalid	Service returns NO_ACCESS	
10	Call CheckObjectAccess() for an alarm object type, *running* task/ISR2 has access to the object	Service returns ACCESS	
11	Call CheckObjectAccess() for an alarm object type, *running* task/ISR2 has NO access to the object	Service returns NO_ACCESS	
12	Call CheckObjectAccess() for a resource object type with <ObjectID> invalid	Service returns NO_ACCESS	
13	Call CheckObjectAccess() for a resource object type, *running* task/ISR2 has access to the object	Service returns ACCESS	
14	Call CheckObjectAccess() for a resource object type, *running* task/ISR2 has NO access to the object	Service returns NO_ACCESS	
15	Call CheckObjectAccess() for a resource object type (RES_SCHEDULER)	Service returns ACCESS	OS318
16	Call CheckObjectAccess() for a schedule table object type with <ObjectID> invalid	Service returns NO_ACCESS	
17	Call CheckObjectAccess() for a schedule table object type, *running* task/ISR2 has access to the object	Service returns ACCESS	
18	Call CheckObjectAccess() for a schedule table object type, *running* task/ISR2 has NO access to the object	Service returns NO_ACCESS	
19	Call CheckObjectAccess() for a counter object type with <ObjectID> invalid	Service returns NO_ACCESS	
20	Call CheckObjectAccess() for a counter object type, *running* task/ISR2 has access to the object	Service returns ACCESS	
21	Call CheckObjectAccess() for a counter object type, *running* task/ISR2 has NO access to the object	Service returns NO_ACCESS	
22	Call CheckObjectAccess() for a counter object type (SystemCounter)	Service returns NO_ACCESS	
23	Call CheckObjectOwnerShip() with <ObjectType> invalid	Service returns INVALID_OSAPPLICATION	OS274, OS017
24	Call CheckObjectOwnerShip() for a task object type with <ObjectID> invalid	Service returns INVALID_OSAPPLICATION	OS274
25	Call CheckObjectOwnerShip() for a task object type	Service returns the identifier of the OS-Application to which the object belongs	OS273
26	Call CheckObjectOwnerShip() for an ISR2 object type with <ObjectID> invalid	Service returns INVALID_OSAPPLICATION	
27	Call CheckObjectOwnerShip() for an ISR2 object type	Service returns the identifier of the OS-Application to which the object belongs	
28	Call CheckObjectOwnerShip() for an alarm object type with <ObjectID> invalid	Service returns INVALID_OSAPPLICATION	
29	Call CheckObjectOwnerShip() for an alarm object type	Service returns the identifier of the OS-Application to which the object belongs	

Test Case No.	Action	Expected Result	OS Requirements
30	Call CheckObjectOwnerShip() for a resource object type with <ObjectID> invalid	Service returns INVALID_OSAPPLICATION	
31	Call CheckObjectOwnerShip() for a resource object type	Service returns the identifier of the OS-Application to which the object belongs	
32	Call CheckObjectOwnerShip() for a resource object type (RES_SCHEDULER)	Service returns INVALID_OSAPPLICATION	OS319
33	Call CheckObjectOwnerShip() for a schedule table object type with <ObjectID> invalid	Service returns INVALID_OSAPPLICATION	
34	Call CheckObjectOwnerShip() for a schedule table object type	Service returns the identifier of the OS-Application to which the object belongs	
35	Call CheckObjectOwnerShip() for a counter object type with <ObjectID> invalid	Service returns INVALID_OSAPPLICATION	
36	Call CheckObjectOwnerShip() for a counter object type	Service returns the identifier of the OS-Application to which the object belongs	
37	Call CheckObjectOwnerShip() for a counter object type (SystemCounter)	Service returns INVALID_OSAPPLICATION	
38	Call TerminateApplication() with <RestartOption> invalid	Service returns E_OS_VALUE	OS459
39	Call TerminateApplication() with <RestartOption> equals NO RESTART	The OS shall terminate the calling OS-Application (i.e. to kill all tasks, disable the interrupt sources of those OsIsrs which belong to the OS-Application and free all other OS resources associated with the application)	OS258, OS287, OS447
40	Call TerminateApplication() with <RestartOption> equals RESTART	The OS shall terminate the calling OS-Application (i.e. to kill all tasks, disable the interrupt sources of those OsIsrs which belong to the OS-Application and free all other OS resources associated with the application) and shall activate the configured *OsRestartTask* of the terminated OS-Application	OS258, OS346, OS447
41	Call GetApplicationID() and no OS-Application is running	Service returns INVALID_OSAPPLICATION	OS262
42	Call GetApplicationID() and one OS-Application is running	Service returns the application identifier to which the executing Task/OsIsr/hook belongs	OS016, OS261
43	No Task nor ISR2 in an application	error : An application should have at least one Task OR ISR2.	OS445
44	At least one Task or OsIsr in an application		OS445

2.12.2 Access Rights for objects in API services

OS Requirements : 56, 448

Test Case No.	Action	Expected Result	OS Requirements
1	Call ActivateTask() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	OS448
2	Call ActivateTask() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	OS056, OS448
3	Call ChainTask() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	


AUTOSAR_Application_Access_Rights
Called OS Service
access rights
return status
ActivateTask
ChainTask
GetTaskState
GetResource
ReleaseResource
SetEvent
GetEvent
GetAlarmBase
GetAlarm
SetRelAlarm
SetAbsAlarm
CancelAlarm
StartSTRel
StartSTAbs
StopST
NextST
StartSTSynchron
SyncST
SetSTAsync
GetSTStatus
IncrementCounter
GetCounterValue
GetElapsedCounterValue
yes
no
X
E_OS_ACCESS
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46

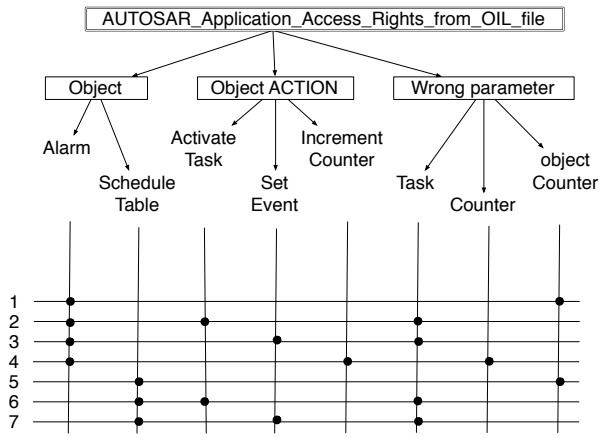
Test Case No.	Action	Expected Result	OS Requirements
4	Call ChainTask() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
5	Call GetTaskState() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
6	Call GetTaskState() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
7	Call GetResource() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
8	Call GetResource() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
9	Call ReleaseResource() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
10	Call ReleaseResource() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
11	Call SetEvent() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
12	Call SetEvent() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
13	Call GetEvent() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
14	Call GetEvent() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
15	Call GetAlarmBase() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
16	Call GetAlarmBase() for a task which can't be accessed by the *running* task/ISR 2	Service returns E_OS_ACCESS	
17	Call GetAlarm() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
18	Call GetAlarm() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
19	Call SetRelAlarm() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
20	Call SetRelAlarm() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
21	Call SetAbsAlarm() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
22	Call SetAbsAlarm() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
23	Call CancelAlarm() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
24	Call CancelAlarm() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
25	Call StartScheduleTableRel() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
26	Call StartScheduleTableRel() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
27	Call StartScheduleTableAbs() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
28	Call StartScheduleTableAbs() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	

Test Case No.	Action	Expected Result	OS Requirements
29	Call StopScheduleTable() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
30	Call StopScheduleTable() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
31	Call NextScheduleTable() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
32	Call NextScheduleTable() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
33	Call StartScheduleTableSynchron() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
34	Call StartScheduleTableSynchron() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
35	Call SyncScheduleTable() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
36	Call SyncScheduleTable() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
37	Call SetScheduleTableAsync() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
38	Call SetScheduleTableAsync() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
39	Call GetScheduleTableStatus() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
40	Call GetScheduleTableStatus() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
41	Call IncrementCounter() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
42	Call IncrementCounter() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
43	Call GetCounterValue() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
44	Call GetCounterValue() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	
45	Call GetElapsedCounterValue() for a task which can be accessed by the *running* task/ISR2	Service returns E_OK if no error	
46	Call GetElapsedCounterValue() for a task which can't be accessed by the *running* task/ISR2	Service returns E_OS_ACCESS	

2.12.3 Access Rights for objects from OIL file

OS Requirements: 056

Test Case No.	Action	Expected Result	OS Requirements

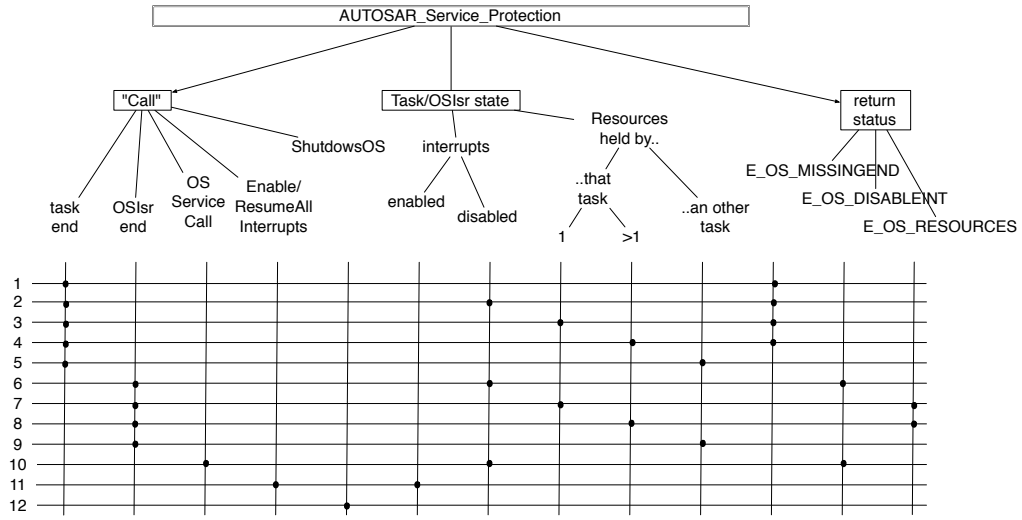


Test Case No.	Action	Expected Result	OS Requirements
1	Alarm's Counter doesn't belong to the same application of the alarm and the alarm has no access rights to the counter's application	error : Counter C doesn't belong to the same application of alarm A	
2	Action of an alarm results in a ActivateTask. Action's Task doesn't belong to the same application of the alarm and the alarm has no access rights to the task's application	error : Task T doesn't belong to the same application of alarm A	
3	Action of an alarm results in a SetEvent. Action's Task doesn't belong to the same application of the alarm and the alarm has no access rights to the task's application	error : Task T doesn't belong to the same application of alarm A	
4	Action of an alarm results in a IncrementCounter. Action's Counter doesn't belong to the same application of the alarm and the alarm has no access rights to the counter's application	error : Counter C doesn't belong to the same application of alarm A	
5	Schedule table's Counter doesn't belong to the same application of the schedule table and the schedule table has no access rights to the counter's application	error : Counter C doesn't belong to the same application of schedule table S	
6	Action of an expiry point of a schedule table results in a ActivateTask. Action's Task doesn't belong to the same application of the schedule table and the schedule table has no access rights to the task's application	error : Task T doesn't belong to the same application of schedule table S	
7	Action of an expiry point of a schedule table results in a SetEvent. Action's Task doesn't belong to the same application of the schedule table and the schedule table has no access rights to the task's application	error : Task T doesn't belong to the same application of schedule table S	

2.13 AUTOSAR - Service Protection

OS Requirements : 52, 69, 70, 71, 92, 93, 239, 368, 369
Test case 11 can't be tested because enabling/resuming API service call doesn't return.
As specified in AUTOSAR OS Specifications, when an API service call happens when interrupts are disabled, the service should be ignored and should return E_OS_DISABLEDINT when the service return a StatusType (OS093, Test Case 10). The ErrorHook(s) is(are) called.

As nothing is described for API services which doesn't return a StatusType, we decide executing the service correctly, calling the Errorhook(s) with E_OS_DISABLEDINT as sequence 5 in the procedure (See GetActiveApplicationMode(), GetApplicationID(), GetISRID(), CheckObjectAccess(), CheckObjectOwnership()).



Test Case No.	Action	Expected Result	OS Requirements
1	Ending a task without making a TerminateTask() or ChainTask() call	The OS shall terminate the task, call the errorhook (if configured) with status E_OS_MISSINGEND before leaving RUNNING state and call the posttaskhook (is configured)	OS052, OS069
2	Ending a task without making a TerminateTask() with interrupts disabled	The OS shall terminate the task, call the errorhook (if configured) with status E_OS_MISSINGEND and enabling interrupts	OS239
3	Ending a task without making a TerminateTask(), holding 1 resource	The OS shall terminate the task, call the errorhook (if configured) with status E_OS_MISSINGEND and release the resource	OS070
4	Ending a task without making a TerminateTask(), holding several resources	The OS shall terminate the task, call the errorhook (if configured) with status E_OS_MISSINGEND and release resources	OS070
5	Ending a task without making a TerminateTask(), an other task holding resource(s)	The OS shall terminate the task, call the errorhook (if configured) with status E_OS_MISSINGEND	OS070
6	Ending an ISR2 with interrupts disabled	The OS shall call the errorhook (if configured) with status E_OS_DISABLEDINT and enabling interrupts	OS368
7	Ending an ISR2, holding 1 resource	The OS shall call the errorhook (if configured) with status E_OS_RESOURCE and release the resource	OS369
8	Ending an ISR2, holding several resources	The OS shall call the errorhook (if configured) with status E_OS_RESOURCE and release resources	OS369

Test Case No.	Action	Expected Result	OS Requirements
9	Ending an ISR2, an other task holding resource(s)	The OS shall call the errorhook (if configured) with status E_OS_RESOURCE	OS369
10	Call an OS service when interrupts are disabled	Service (which can) returns E_OS_DISABLEDINT, ignoring the service	OS093
11	Enabling/Resuming ingterrupts when interrupts are already enabled	Service ignored	OS092
12	Call ShutdownOS()	PostTaskHook is not performed (even if PostTaskHook is configured)	OS071

A Interrupts Management

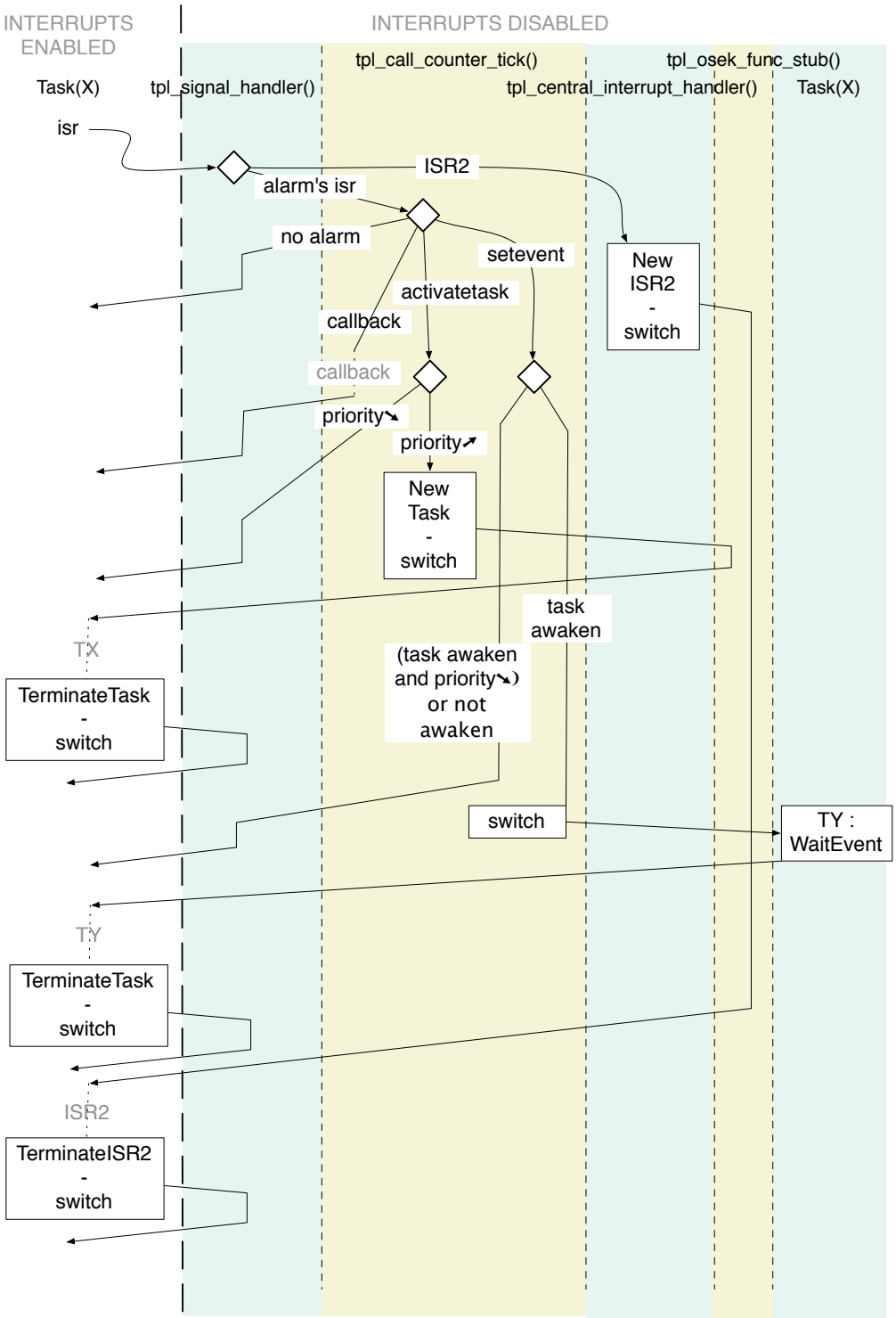
References

[1] Consortium OSEK/VDX. *OSEK/VDX OS Test Plan*, 2.0 edition, 16th April 1999.

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

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

[4] Consortium OSEK/VDX. *OSEK/VDX Communication*, 3.0.3 edition, 2004.


INTERRUPTS ENABLED
INTERRUPTS DISABLED
Task(X)
tpl_signal_handler()
tpl_call_counter_tick()
tpl_central_interrupt_handler()
tpl_osek_func_stub()
Task(X)
isr
ISR2
alarm's isr
no alarm
setevent
activatetask
callback
New ISR2 - switch
callback
priority↘
priority↗
New Task - switch
task awaken
TX
(task awaken and priority↘) or not awaken
TerminateTask - switch
switch
TY : WaitEvent
TY
TerminateTask - switch
ISR2
TerminateISR2 - switch