OSEK/VDX

OS Test Procedure

Version 2.0

21st April 1999

What is OSEK/VDX?

OSEK/VDX is a joint project of the automotive industry. It aims at an industry standard for an open-ended architecture for distributed control units in vehicles.
A real-time operating system, software interfaces and functions for communication and network management tasks are thus jointly specified.
The term OSEK means ”Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug” (Open systems and the corresponding interfaces for automotive electronics).
The term VDX means ”Vehicle Distributed eXecutive”. The functionality of OSEK operating system was harmonized with VDX. For simplicity OSEK will be used instead of OSEK/VDX in this document.

OSEK partners:

Adam Opel AG, BMW AG, Daimler-Benz AG, IIIT University of Karlsruhe, Mercedes-Benz AG, Robert Bosch GmbH, Siemens AG, Volkswagen AG., GIE.RE. PSA-Renault.

Motivation:

- High, recurring expenses in the development and variant management of non-application related aspects of control unit software.
- Incompatibility of control units made by different manufacturers due to different interfaces and protocols.

Goal:

Support of the portability and reusability of the application software by:

- Specification of interfaces which are abstract and as application-independent as possible, in the following areas: real-time operating system, communication and network management.
- Specification of a user interface independent of hardware and network.
- Efficient design of architecture: The functionality shall be configurable and scaleable, to enable optimal adjustment of the architecture to the application in question.
- Verification of functionality and implementation of prototypes in selected pilot projects.

Advantages:

- Clear savings in costs and development time.
- Enhanced quality of the control units software of various companies.
- Standardized interfacing features for control units with different architectural designs.
- Sequenced utilization of the intelligence (existing resources) distributed in the vehicle, to enhance the performance of the overall system without requiring additional hardware.
- Provides absolute independence with regards to individual implementation, as the specification does not prescribe implementational aspects.

OSEK conformance testing

OSEK conformance testing aims at checking conformance of products to OSEK specifications. Test suites are thus specified for implementations of OSEK operating system, communication and network management.

Work around OSEK conformance testing is supported by the MODISTARC project sponsored by the Commission of European Communities. The term MODISTARC means ”Methods and tools for the validation of OSEK/VDX based DISTributed ARChitectures”.

This document has been drafted by the MODISTARC members of the OS-Workgroup:

Bernd Büchs	Adam Opel AG
Wolfgang Kremer	BMW AG
Stefan Schmerler	FZI
Franz Adis	FZI
Yves Sorel	INRIA
Robert France	Motorola
Barbara Ziker	Motorola
Jean-Emmanuel Hanne	Peugeot Citroën S.A.
Eric Brodin	Sagem SA
Gerhard Goeser	Siemens Automotive SA
Patrick Palmieri	Siemens Automotive SA

Table of Contents

1. Introduction

This document describes the test procedure for the conformance test of the operating system. The test procedure contains the definition of test sequences.

Chapter 2 shows the needed test environment and the used methods to obtain the test results.

Chapter 3 contains the definition of the used test sequences. The test sequences determine how the test cases will be tested. This contains the sequence of actions that must be taken by the test program, and their expected reactions. The definition of the test sequences is based on the test cases defined in the OSEK OS Test Plan [2].

2. Test environment

As agreed in the Conformance Testing Methodology [1] the implementation under test is seen as a black box whose external interfaces – the OS API – are accessible only. Implementation-specific details will not be taken into account and only that parts of the specification which are accessible and observable by the operating system's service routines, can be tested for conformance. Therefore, executing the conformance test means that a test application is generated and executed together with the implementation to be tested. The actions and verifications this application has to perform are defined by the test sequences as defined in chapter 3.

A test is passed if

1. all function calls returned with the specified value and
2. all statements of the application are executed in the specified sequence.

To verify that these requirements are fulfilled it is necessary to trace the execution of the test application. One approach can be the usage of a debugger which has the drawback that it is very dependant on the hardware and the development environment (there may be no debugger available for some systems) and that it doesn't allow an automation of the testing procedure.

Within MODISTARC the following approach to trace the execution of the test application will be used:

The test execution will create after each call to an OS service a bit pattern which consists of the following information:

- execution level, i. e. ID of running task, ISR, hook routine, etc.
- error flag which is set if an OS service returned an unexpected status or value
- sequence number to identify the executed statement

This pattern will be outputted by a method suitable for the target system. The following solutions may be possible:

- Write the patterns directly into a file if the target system supports a file system (e.g. PC).
- Write the patterns into a special memory area of the target system where it can be read out by special hardware (e.g. emulator for target ECU).
- Write the patterns to I/O ports of the target ECU where it can be observed by a logic analyzer.
- Transfer the patterns via serial link (RS232) or network (CAN) during test execution to a remote host.
- etc.

Each test application has its own dedicated pattern sequence. Any discrepancy between this pattern sequence and the sequence received from the test execution means that the test failed. The patterns may be transferred automatically to an evaluation system which analyzes the patterns and checks them for correctness.

The format of the patterns (size, meaning of each bit) has to be defined by the implementor of the test suite.

Figure 1 shows an example of a test environment.

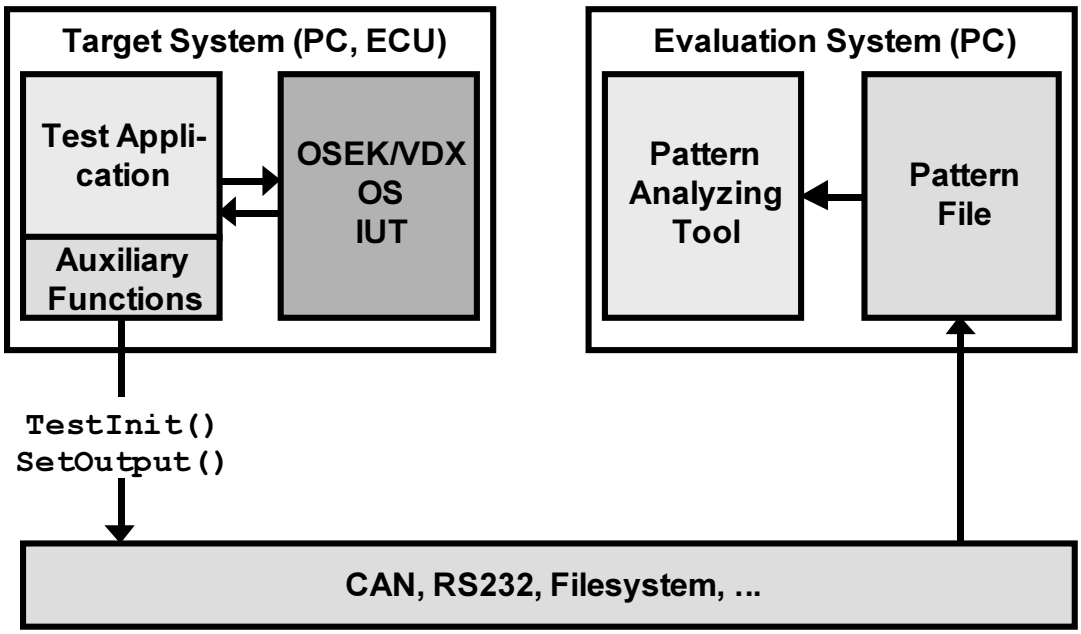

Figure 1: Test environment

3. Test sequences

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

3.1. Task management

Test Sequence 1:

Test cases: 1, 10, 15, 20, 21, 22, 24, 25, 26, 27, 30, 35, 36, 37, 38, 40
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1
type: basic
autostart: yes
priority: 1
max. activationss: 1
preemptive: non, full
Task2
type: basic
autostart: no
priority: 2
max. activationss: 1
preemptive: non, full
ISRs: ISR2
category: 2
ISR3
category: 3

Running task	Called OS service	Return status	Test case
Task1	`EnableInterrupt(ISR2|ISR3)`	`E_OK|E_OS_NOFUNC`	
Task1	`ActivateTask(INVALID_TASK)`	`E_OS_ID`	1
Task1	`GetTaskState(INVALID_TASK)`	`E_OS_ID`	40
Task1	`ChainTask(INVALID_TASK)`	`E_OS_ID`	24
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	*force scheduling*		
Task2	`ActivateTask(Task1)`	`E_OS_LIMIT`	10
Task2	`ActivateTask(Task2)`	`E_OS_LIMIT`	15
Task2	`ChainTask(Task1)`	`E_OS_LIMIT`	30
Task2	`TerminateTask()`		
Task1	`GetResource(RES_SCHEDULER)`	`E_OK`	
Task1	`TerminateTask()`	`E_OS_RESOURCE`	22
Task1	`ChainTask(Task2)`	`E_OS_RESOURCE`	27

Running task	Called OS service	Return status	Test case
Task1	`ReleaseResource(RES_SCHEDULER)`	`E_OK`	
Task1	*trigger interrupt ISR2*		
ISR2	`TerminateTask()`	`E_OS_CALLEVEL`	20
ISR2	`ChainTaskTask(Task2)`	`E_OS_CALLEVEL`	25
ISR2	`Schedule()`	`E_OS_CALLEVEL`	35
ISR2	`GetTaskID()`	`E_OS_CALLEVEL`	37
Task1	*trigger interrupt ISR3*		
ISR3	`EnterISR()`		
ISR3	`TerminateTask()`	`E_OS_CALLEVEL`	21
ISR3	`ChainTaskTask(Task3)`	`E_OS_CALLEVEL`	26
ISR3	`Schedule()`	`E_OS_CALLEVEL`	36
ISR3	`GetTaskID()`	`E_OS_CALLEVEL`	38
ISR3	`LeaveISR()`		
Task1	`TerminateTask()`		

Test Sequence 2:

Test cases: 2, 34
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: yes
- priority: 1
- max. activationss: 1
- preemptive: non

Task2
- type: basic
- autostart: no
- priority: 2
- max. activationss: 1
- preemptive: non

Task3
- type: basic
- autostart: no
- priority: 3
- max. activationss: 1
- preemptive: non

Running task	Called OS service	Return status	Test case
Task1	`ActivateTask(Task2)`	`E_OK`	2
Task1	`ActivateTask(Task3)`	`E_OK`	2
Task1	`Schedule()`	`E_OK`	34
Task3	`TerminateTask()`		
Task2	`TerminateTask()`		

Running task	Called OS service	Return status	Test case
Task1	`TerminateTask()`		

Test Sequence 3:

Test cases: 3, 4
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: mixed-, full-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: yes
- priority: 1
- max. activations: 1
- preemptive: full

Task2
- type: basic
- autostart: no
- priority: 2
- max. activations: 1
- preemptive: full

Task3
- type: basic
- autostart: no
- priority: 3
- max. activations: 1
- preemptive: full

Running task	Called OS service	Return status	Test case
Task1	`ActivateTask(Task3)`	`E_OK`	3
Task3	`ActivateTask(Task2)`	`E_OK`	4
Task3	`TerminateTask()`		
Task2	`TerminateTask()`		
Task1	`TerminateTask()`		

Test Sequence 4:

Test cases: 6
Conformance Class: ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-preemptive
Hooks: -
Tasks: Task1
- type: extended
- autostart: yes
- priority: 1
- max. activations: 1
- preemptive: non
- events: Event1

Task2
type: extended
autostart: no
priority: 2
max. activations: 1
preemptive: non
events: Event2

Events: Event1, Event2

Running task	Called OS service	Return status	Test case
Task1	`ActivateTask(Task2)`	`E_OK`	6
Task1	`GetEvent(Task1, &EventMask)`	`E_OK, EventMask=0x0`	
Task1	`GetEvent(Task2, &EventMask)`	`E_OK, EventMask=0x0`	
Task1	`Schedule()`	`E_OK`	
Task2	`TerminateTask()`		
Task1	`TerminateTask()`		

Test Sequence 5:

Test cases: 7, 8
Conformance Class: ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: mixed-, full-preemptive
Hooks: -
Tasks: Task1
type: basic
autostart: yes
priority: 1
max. activations: 1
preemptive: full
Task2
type: extended
autostart: no
priority: 2
max. activations: 1
preemptive: full
events: Event2
Task3
type: extended
autostart: no
priority: 3
max. activations: 1
preemptive: full
events: Event3

Events: Event2, Event3

Running task	Called OS service	Return status	Test Case
Task1	`ActivateTask(Task3)`	`E_OK`	7
Task3	`GetEvent(Task3, &EventMask)`	`E_OK, EventMask=0x0`	
Task3	`ActivateTask(Task2)`	`E_OK`	8
Task3	`TerminateTask()`		
Task2	`GetEvent(Task2, &EventMask)`	`E_OK, EventMask=0x0`	
Task2	`TerminateTask()`		
Task1	`TerminateTask()`		

Test Sequence 6:

Test cases: 11, 16, 19, 31, 33
Conformance Class: ECC1, ECC2
Return Status: extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1
- type: extended
- autostart: yes
- priority: 1
- max. activations: 1
- preemptive: non, full

Task2
- type: extended
- autostart: no
- priority: 2
- max. activations: 1
- preemptive: non, full
- events: Event2

Events: Event2

Running task	Called OS service	Return status	Test Case
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	*force scheduling*		
Task2	`ActivateTask(Task1)`	`E_OS_LIMIT`	11
Task2	`ActivateTask(Task2)`	`E_OS_LIMIT`	16
Task2	`WaitEvent(Event2)`	`E_OK`	
Task1	`GetTaskState(Task2)`	`E_OK, TaskState=WAITING`	
Task1	`ActivateTask(Task2)`	`E_OS_LIMIT`	19
Task1	`ChainTask(Task2)`	`E_OS_LIMIT`	33
Task1	`SetEvent(Task2, Event2)`	`E_OK`	
Task1	*force scheduling*		
Task2	`ChainTask(Task1)`	`E_OS_LIMIT`	31
Task2	`TerminateTask()`		
Task1	`TerminateTask()`		

Test Sequence 7:

Test cases: 12, 17, 32
Conformance Class: BCC2, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-preemptive
Hooks: -
Tasks: Task1
type: basic
autostart: yes
priority: 1
max. activations: 2
preemptive: non
Task2
type: basic
autostart: no
priority: 2
max. activations: 2
preemptive: non
Task3
type: basic
autostart: no
priority: 3
max. activations: 2
preemptive: non

Running task	Called OS service	Return status	Test case
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	`ActivateTask(Task2)`	`E_OK`	12
Task1	`Schedule()`	`E_OK`	
Task2	`TerminateTask()`		
Task2	`TerminateTask()`		
Task1	`ActivateTask(Task1)`	`E_OK`	17
Task1	`ActivateTask(Task3)`	`E_OK`	
Task1	`ChainTask(Task3)`		32
Task3	`TerminateTask()`		
Task3	`TerminateTask()`		
Task1	`ActivateTask(Task1)`	`E_OK`	
Task1	`TerminateTask()`		
Task1	`TerminateTask()`		

Test Sequence 8:

Test cases: 5, 13, 14, 18
Scheduling policy: mixed-, full-preemptive
Conformance Class: BCC2, ECC2
Return Status: extended
Scheduling Policy: mixed-, full-preemptive
Hooks: -

Tasks: Task1
- type: basic
- autostart: yes
- priority: 1
- max. activations: 2
- preemptive: full

Task2
- type: basic
- autostart: no
- priority: 2
- max. activations: 1
- preemptive: full

Task3
- type: basic
- autostart: no
- priority: 2
- max. activations: 1
- preemptive: full

Running task	Called OS service	Return status	Test case
Task1	`ActivateTask(Task2)`	`E_OK`	5
Task2	`ActivateTask(Task1)`	`E_OK`	13
Task2	`ActivateTask(Task3)`	`E_OK`	14
Task2	`TerminateTask()`		
Task3	`TerminateTask()`		
Task1	`TerminateTask()`		
Task1	`ActivateTask(Task1)`	`E_OK`	18
Task1	`TerminateTask()`		
Task1	`TerminateTask()`		

Test Sequence 9:

Test cases: 23, 28, 29, 39, 41
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: yes
- priority: 1
- max. activations: 1
- preemptive: non, full

Task2
- type: basic
- autostart: no
- priority: 2
- max. activations: 1
- preemptive: non, full

Task3

type:	basic
autostart:	no
priority:	3
max. activations:	1
preemptive:	non, full

Running task	Called OS service	Return status	Test case
Task1	`GetTaskID(&TaskID)`	`E_OK, TaskID=Task1`	39
Task1	`GetTaskState(Task1, &TaskState)`	`E_OK, TaskState=RUNNING`	41
Task1	`GetTaskState(Task2, &TaskState)`	`E_OK, TaskState=SUSPENDED`	41
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	*force scheduling*	`E_OK`	
Task2	`GetTaskState(Task1, &TaskState)`	`E_OK, TaskState=READY`	41
Task2	`TerminateTask()`		
Task1	`ChainTask(Task3)`		28
Task3	`ChainTask(Task3)`		29
Task3	`TerminateTask()`		23

Test Sequence 10:

Test cases:	9
Conformance Class:	ECC2
Return Status:	standard, extended
Scheduling Policy:	mixed-, full-preemptive
Hooks:	-
Tasks:	Task1

type:	basic
autostart:	yes
priority:	1
max. activations:	1
preemptive:	full

Task2

type:	extended
autostart:	no
priority:	2
max. activations:	1
preemptive:	full
events:	Event2

Task3

type:	extended
autostart:	no
priority:	2
max. activations:	1
preemptive:	full
events:	Event3

Events: Event2, Event3

Running task	Called OS service	Return status	Test case
Task1	`ActivateTask(Task2)`	`E_OK`	
Task2	`GetEvent(Task2, &EventMask)`	`E_OK, EventMask=0x0`	
Task2	`ActivateTask(Task3)`	`E_OK`	9
Task2	`TerminateTask()`		
Task3	`GetEvent(Task3, &EventMask)`	`E_OK, EventMask=0x0`	
Task3	`TerminateTask()`		
Task1	`TerminateTask()`		

Test Sequence 11:

Test cases: A task beeing released from the *waiting* state is treated like the newest task in the *ready* queue of its priority.
Conformance Class: ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: no
- priority: 1
- max. activations: 1
- preemptive: non, full

Task2
- type: extended
- autostart: yes
- priority: 2
- max. activations: 1
- preemptive: non, full
- events: Event2

Task3
- type: basic
- autostart: no
- priority: 2
- max. activations: 1
- preemptive: non, full

Task4
- type: basic
- autostart: no
- priority: 3
- max. activations: 1
- preemptive: non, full

Events: Event2

Running task	Called OS service	Return status	Test case
Task2	`ActivateTask(Task1)`	`E_OK`	
Task2	`WaitEvent(Event2)`	`E_OK`	
Task1	`ActivateTask(Task3)`	`E_OK`	
Task1	*force scheduling*		
Task3	`ActivateTask(Task4)`	`E_OK`	
Task3	*force scheduling*		
Task4	`SetEvent(Task2, Event2)`	`E_OK`	
Task4	`TerminateTask()`		
Task3	`TerminateTask()`		
Task2	`TerminateTask()`		
Task1	`TerminateTask()`		

Test Sequence 12:

Test cases: A preempted task is considered to be the first task in the *ready* queue of its current priority.
Conformance Class: BCC2, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: yes
- priority: 2
- max. activations: 1
- preemptive: non, full

Task2
- type: basic
- autostart: no
- priority: 1;
- max. activations: 3
- preemptive: non, full

Task3
- type: basic
- autostart: no
- priority: 1
- max. activations: 3
- preemptive: non, full

Running task	Called OS service	Return status	Test case
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	`ActivateTask(Task3)`	`E_OK`	
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	`ActivateTask(Task3)`	`E_OK`	
Task1	`ActivateTask(Task3)`	`E_OK`	
Task1	`TerminateTask()`		

Running task	Called OS service	Return status	Test case
Task2	`TerminateTask()`		
Task3	`TerminateTask()`		
Task2	`TerminateTask()`		
Task2	`TerminateTask()`		
Task3	`TerminateTask()`		
Task3	`TerminateTask()`		

Test Sequence 13:

Test cases: Number of tasks which are not in the *suspended* state ≥ 8
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1

type:	basic
autostart:	yes
priority:	8
max. activations:	1
preemptive:	non, full

Task2

type:	basic
autostart:	no
priority:	7
max. activations:	1
preemptive:	non, full

Task3

type:	basic
autostart:	no
priority:	6
max. activations:	1
preemptive:	non, full

Task4

type:	basic
autostart:	no
priority:	5
max. activations:	1
preemptive:	non, full

Task5

type:	basic
autostart:	no
priority:	4
max. activations:	1
preemptive:	non, full

Task6

type:	basic
autostart:	no
priority:	3

max. activations: 1
preemptive: non, full

Task7

type: basic
autostart: no
priority: 2
max. activations: 1
preemptive: non, full

Task8

type: basic
autostart: no
priority: 1
max. activations: 1
preemptive: non, full

Running task	Called OS service	Return status	Test case
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	`ActivateTask(Task3)`	`E_OK`	
Task1	`ActivateTask(Task4)`	`E_OK`	
Task1	`ActivateTask(Task5)`	`E_OK`	
Task1	`ActivateTask(Task6)`	`E_OK`	
Task1	`ActivateTask(Task7)`	`E_OK`	
Task1	`ActivateTask(Task8)`	`E_OK`	
Task1	`TerminateTask()`		
Task2	`TerminateTask()`		
Task3	`TerminateTask()`		
Task4	`TerminateTask()`		
Task5	`TerminateTask()`		
Task6	`TerminateTask()`		
Task7	`TerminateTask()`		
Task8	`TerminateTask()`		

Test Sequence 14:

Test cases: Number of tasks which are not in the *suspended* state ≥ 16,
number of events per task >= 8.
Conformance Class: ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1

type: extended
autostart: yes
priority: 16
max. activations: 1
preemptive: non, full
events: Task1_Event1, Task1_Event2, Task1_Event3, Task1_Event4, Task1_Event5, Task1_Event6, Task1_Event7, Task1_Event8

Task2

type:	extended
autostart:	no
priority:	15
max. activations:	1
preemptive:	non, full
events:	Task2_Event1, Task2_Event2, Task2_Event3, Task2_Event4, Task2_Event5, Task2_Event6, Task2_Event7, Task2_Event8

Task3

type:	extended
autostart:	no
priority:	14
max. activations:	1
preemptive:	non, full
events:	Task3_Event1, Task3_Event2, Task3_Event3, Task3_Event4, Task3_Event5, Task3_Event6, Task3_Event7, Task3_Event8

Task4

type:	extended
autostart:	no
priority:	13
max. activations:	1
preemptive:	non, full
events:	Task4_Event1, Task4_Event2, Task4_Event3, Task4_Event4, Task4_Event5, Task4_Event6, Task4_Event7, Task4_Event8

Task5

type:	extended
autostart:	no
priority:	12
max. activations:	1
preemptive:	non, full
events:	Task5_Event1, Task5_Event2, Task5_Event3, Task5_Event4, Task5_Event5, Task5_Event6, Task5_Event7, Task5_Event8

Task6

type:	extended
autostart:	no
priority:	11
max. activations:	1
preemptive:	non, full
events:	Task6_Event1, Task6_Event2, Task6_Event3, Task6_Event4, Task6_Event5, Task6_Event6, Task6_Event7, Task6_Event8

Task7

type:	extended
autostart:	no
priority:	10
max. activations:	1

preemptive: non, full
events: Task7_Event1, Task7_Event2, Task7_Event3, Task7_Event4, Task7_Event5, Task7_Event6, Task8
type: extended
autostart: no
priority: 9
max. activations: 1
preemptive: non, full
events: Task8_Event1, Task8_Event2, Task8_Event3, Task8_Event4, Task8_Event5, Task8_Event6, Task8_Event7, Task8_Event8

Task9
type: extended
autostart: no
priority: 8
max. activations: 1
preemptive: non, full
events: Task9_Event1, Task9_Event2, Task9_Event3, Task9_Event4, Task9_Event5, Task9_Event6, Task9_Event7, Task9_Event8

Task10
type: extended
autostart: no
priority: 7
max. activations: 1
preemptive: non, full
events: Task10_Event1, Task10_Event2, Task10_Event3, Task10_Event4, Task10_Event5, Task10_Event6, Task10_Event7, Task10_Event8

Task11
type: extended
autostart: no
priority: 6
max. activations: 1
preemptive: non, full
events: Task11_Event1, Task11_Event2, Task11_Event3, Task11_Event4, Task11_Event5, Task11_Event6, Task11_Event7, Task11_Event8

Task12
type: extended
autostart: no
priority: 5
max. activations: 1
preemptive: non, full
events: Task12_Event1, Task12_Event2, Task12_Event3, Task12_Event4, Task12_Event5, Task12_Event6, Task12_Event7, Task12_Event8

Task13
type: extended

autostart: no
priority: 4
max. activations: 1
preemptive: non, full
events: Task13_Event1, Task13_Event2, Task13_Event3, Task13_Event4, Task13_Event5, Task13_Event6, Task13_Event7, Task13_Event8

Task14
type: extended
autostart: no
priority: 3
max. activations: 1
preemptive: non, full
events: Task14_Event1, Task14_Event2, Task14_Event3, Task14_Event4, Task14_Event5, Task14_Event6, Task14_Event7, Task14_Event8

Task15
type: extended
autostart: no
priority: 2
max. activations: 1
preemptive: non, full
events: Task15_Event1, Task15_Event2, Task15_Event3, Task15_Event4, Task15_Event5, Task15_Event6, Task15_Event7, Task15_Event8

Task16
type: extended
autostart: no
priority: 1
max. activations: 1
preemptive: non, full
events: Task16_Event1, Task16_Event2, Task16_Event3, Task16_Event4, Task16_Event5, Task16_Event6, Task16_Event7, Task16_Event8

Events: Task1_Event1, Task1_Event2, Task1_Event3, Task1_Event4, Task1_Event5, Task1_Event6, Task1_Event7, Task1_Event8,
Task2_Event1, Task2_Event2, Task2_Event3, Task2_Event4, Task2_Event5, Task2_Event6, Task2_Event7, Task2_Event8
Task3_Event1, Task3_Event2, Task3_Event3, Task3_Event4, Task3_Event5, Task3_Event6, Task3_Event7, Task3_Event8
Task4_Event1, Task4_Event2, Task4_Event3, Task4_Event4, Task4_Event5, Task4_Event6, Task4_Event7, Task4_Event8
Task5_Event1, Task5_Event2, Task5_Event3, Task5_Event4, Task5_Event5, Task5_Event6, Task5_Event7, Task5_Event8
Task6_Event1, Task6_Event2, Task6_Event3, Task6_Event4, Task6_Event5, Task6_Event6, Task6_Event7, Task6_Event8
Task7_Event1, Task7_Event2, Task7_Event3, Task7_Event4, Task7_Event5, Task7_Event6, Task7_Event7, Task7_Event8
Task8_Event1, Task8_Event2, Task8_Event3, Task8_Event4, Task8_Event5, Task8_Event6, Task8_Event7, Task8_Event8

Task9_Event1, Task9_Event2, Task9_Event3, Task9_Event4, Task9_Event5, Task9_Event6, Task9_Event7, Task9_Event8
Task10_Event1, Task10_Event2, Task10_Event3, Task10_Event4, Task10_Event5, Task10_Event6, Task10_Event7, Task10_Event8
Task11_Event1, Task11_Event2, Task11_Event3, Task11_Event4, Task11_Event5, Task11_Event6, Task11_Event7, Task11_Event8
Task12_Event1, Task12_Event2, Task12_Event3, Task12_Event4, Task12_Event5, Task12_Event6, Task12_Event7, Task12_Event8
Task13_Event1, Task13_Event2, Task13_Event3, Task13_Event4, Task13_Event5, Task13_Event6, Task13_Event7, Task13_Event8
Task14_Event1, Task14_Event2, Task14_Event3, Task14_Event4, Task14_Event5, Task14_Event6, Task14_Event7, Task14_Event8
Task15_Event1, Task15_Event2, Task15_Event3, Task15_Event4, Task15_Event5, Task15_Event6, Task15_Event7, Task15_Event8
Task16_Event1, Task16_Event2, Task16_Event3, Task16_Event4, Task16_Event5, Task16_Event6, Task16_Event7, Task16_Event8

Running task	Called OS service	Return status	Test case
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	`ActivateTask(Task3)`	`E_OK`	
Task1	`ActivateTask(Task4)`	`E_OK`	
Task1	`ActivateTask(Task5)`	`E_OK`	
Task1	`ActivateTask(Task6)`	`E_OK`	
Task1	`ActivateTask(Task7)`	`E_OK`	
Task1	`ActivateTask(Task8)`	`E_OK`	
Task1	`ActivateTask(Task10)`	`E_OK`	
Task1	`ActivateTask(Task11)`	`E_OK`	
Task1	`ActivateTask(Task12)`	`E_OK`	
Task1	`ActivateTask(Task13)`	`E_OK`	
Task1	`ActivateTask(Task14)`	`E_OK`	
Task1	`ActivateTask(Task15)`	`E_OK`	
Task1	`ActivateTask(Task16)`	`E_OK`	
Task1	`ClearEvent(Task1_Event1)`	`E_OK`	
Task1	`ClearEvent(Task1_Event2)`	`E_OK`	
Task1	`ClearEvent(Task1_Event3)`	`E_OK`	
Task1	`ClearEvent(Task1_Event4)`	`E_OK`	
Task1	`ClearEvent(Task1_Event5)`	`E_OK`	
Task1	`ClearEvent(Task1_Event6)`	`E_OK`	
Task1	`ClearEvent(Task1_Event7)`	`E_OK`	
Task1	`ClearEvent(Task1_Event8)`	`E_OK`	
Task1	`TerminateTask()`		
Task2	`ClearEvent(Task2_Event1)`	`E_OK`	
Task2	`ClearEvent(Task2_Event2)`	`E_OK`	
Task2	`ClearEvent(Task2_Event3)`	`E_OK`	
Task2	`ClearEvent(Task2_Event4)`	`E_OK`	
Task2	`ClearEvent(Task2_Event5)`	`E_OK`	
Task2	`ClearEvent(Task2_Event6)`	`E_OK`	

Running task	Called OS service	Return status	Test case
Task2	ClearEvent(Task2_Event7)	E_OK	
Task2	ClearEvent(Task2_Event8)	E_OK	
Task2	TerminateTask()		
Task3	ClearEvent(Task3_Event1)	E_OK	
Task3	ClearEvent(Task3_Event2)	E_OK	
Task3	ClearEvent(Task3_Event3)	E_OK	
Task3	ClearEvent(Task3_Event4)	E_OK	
Task3	ClearEvent(Task3_Event5)	E_OK	
Task3	ClearEvent(Task3_Event6)	E_OK	
Task3	ClearEvent(Task3_Event7)	E_OK	
Task3	ClearEvent(Task3_Event8)	E_OK	
Task3	TerminateTask()		
Task4	ClearEvent(Task4_Event1)	E_OK	
Task4	ClearEvent(Task4_Event2)	E_OK	
Task4	ClearEvent(Task4_Event3)	E_OK	
Task4	ClearEvent(Task4_Event4)	E_OK	
Task4	ClearEvent(Task4_Event5)	E_OK	
Task4	ClearEvent(Task4_Event6)	E_OK	
Task4	ClearEvent(Task4_Event7)	E_OK	
Task4	ClearEvent(Task4_Event8)	E_OK	
Task4	TerminateTask()		
Task5	ClearEvent(Task5_Event1)	E_OK	
Task5	ClearEvent(Task5_Event2)	E_OK	
Task5	ClearEvent(Task5_Event3)	E_OK	
Task5	ClearEvent(Task5_Event4)	E_OK	
Task5	ClearEvent(Task5_Event5)	E_OK	
Task5	ClearEvent(Task5_Event6)	E_OK	
Task5	ClearEvent(Task5_Event7)	E_OK	
Task5	ClearEvent(Task5_Event8)	E_OK	
Task5	TerminateTask()		
Task6	ClearEvent(Task6_Event1)	E_OK	
Task6	ClearEvent(Task6_Event2)	E_OK	
Task6	ClearEvent(Task6_Event3)	E_OK	
Task6	ClearEvent(Task6_Event4)	E_OK	
Task6	ClearEvent(Task6_Event5)	E_OK	
Task6	ClearEvent(Task6_Event6)	E_OK	
Task6	ClearEvent(Task6_Event7)	E_OK	
Task6	ClearEvent(Task6_Event8)	E_OK	
Task6	TerminateTask()		
Task7	ClearEvent(Task7_Event1)	E_OK	
Task7	ClearEvent(Task7_Event2)	E_OK	
Task7	ClearEvent(Task7_Event3)	E_OK	
Task7	ClearEvent(Task7_Event4)	E_OK	
Task7	ClearEvent(Task7_Event5)	E_OK	
Task7	ClearEvent(Task7_Event6)	E_OK	
Task7	ClearEvent(Task7_Event7)	E_OK	

Running task	Called OS service	Return status	Test case
Task7	ClearEvent(Task7_Event8)	E_OK	
Task7	TerminateTask()		
Task8	ClearEvent(Task8_Event1)	E_OK	
Task8	ClearEvent(Task8_Event2)	E_OK	
Task8	ClearEvent(Task8_Event3)	E_OK	
Task8	ClearEvent(Task8_Event4)	E_OK	
Task8	ClearEvent(Task8_Event5)	E_OK	
Task8	ClearEvent(Task8_Event6)	E_OK	
Task8	ClearEvent(Task8_Event7)	E_OK	
Task8	ClearEvent(Task8_Event8)	E_OK	
Task8	TerminateTask()		
Task9	ClearEvent(Task9_Event1)	E_OK	
Task9	ClearEvent(Task9_Event2)	E_OK	
Task9	ClearEvent(Task9_Event3)	E_OK	
Task9	ClearEvent(Task9_Event4)	E_OK	
Task9	ClearEvent(Task9_Event5)	E_OK	
Task9	ClearEvent(Task9_Event6)	E_OK	
Task9	ClearEvent(Task9_Event7)	E_OK	
Task9	ClearEvent(Task9_Event8)	E_OK	
Task9	TerminateTask()		
Task10	ClearEvent(Task10_Event1)	E_OK	
Task10	ClearEvent(Task10_Event2)	E_OK	
Task10	ClearEvent(Task10_Event3)	E_OK	
Task10	ClearEvent(Task10_Event4)	E_OK	
Task10	ClearEvent(Task10_Event5)	E_OK	
Task10	ClearEvent(Task10_Event6)	E_OK	
Task10	ClearEvent(Task10_Event7)	E_OK	
Task10	ClearEvent(Task10_Event8)	E_OK	
Task10	TerminateTask()		
Task11	ClearEvent(Task11_Event1)	E_OK	
Task11	ClearEvent(Task11_Event2)	E_OK	
Task11	ClearEvent(Task11_Event3)	E_OK	
Task11	ClearEvent(Task11_Event4)	E_OK	
Task11	ClearEvent(Task11_Event5)	E_OK	
Task11	ClearEvent(Task11_Event6)	E_OK	
Task11	ClearEvent(Task11_Event7)	E_OK	
Task11	ClearEvent(Task11_Event8)	E_OK	
Task11	TerminateTask()		
Task12	ClearEvent(Task12_Event1)	E_OK	
Task12	ClearEvent(Task12_Event2)	E_OK	
Task12	ClearEvent(Task12_Event3)	E_OK	
Task12	ClearEvent(Task12_Event4)	E_OK	
Task12	ClearEvent(Task12_Event5)	E_OK	
Task12	ClearEvent(Task12_Event6)	E_OK	
Task12	ClearEvent(Task12_Event7)	E_OK	
Task12	ClearEvent(Task12_Event8)	E_OK	

Running task	Called OS service	Return status	Test case
Task12	TerminateTask()		
Task13	ClearEvent(Task13_Event1)	E_OK	
Task13	ClearEvent(Task13_Event2)	E_OK	
Task13	ClearEvent(Task13_Event3)	E_OK	
Task13	ClearEvent(Task13_Event4)	E_OK	
Task13	ClearEvent(Task13_Event5)	E_OK	
Task13	ClearEvent(Task13_Event6)	E_OK	
Task13	ClearEvent(Task13_Event7)	E_OK	
Task13	ClearEvent(Task13_Event8)	E_OK	
Task13	TerminateTask()		
Task14	ClearEvent(Task14_Event1)	E_OK	
Task14	ClearEvent(Task14_Event2)	E_OK	
Task14	ClearEvent(Task14_Event3)	E_OK	
Task14	ClearEvent(Task14_Event4)	E_OK	
Task14	ClearEvent(Task14_Event5)	E_OK	
Task14	ClearEvent(Task14_Event6)	E_OK	
Task14	ClearEvent(Task14_Event7)	E_OK	
Task14	ClearEvent(Task14_Event8)	E_OK	
Task14	TerminateTask()		
Task15	ClearEvent(Task15_Event1)	E_OK	
Task15	ClearEvent(Task15_Event2)	E_OK	
Task15	ClearEvent(Task15_Event3)	E_OK	
Task15	ClearEvent(Task15_Event4)	E_OK	
Task15	ClearEvent(Task15_Event5)	E_OK	
Task15	ClearEvent(Task15_Event6)	E_OK	
Task15	ClearEvent(Task15_Event7)	E_OK	
Task15	ClearEvent(Task15_Event8)	E_OK	
Task15	TerminateTask()		
Task16	ClearEvent(Task16_Event1)	E_OK	
Task16	ClearEvent(Task16_Event2)	E_OK	
Task16	ClearEvent(Task16_Event3)	E_OK	
Task16	ClearEvent(Task16_Event4)	E_OK	
Task16	ClearEvent(Task16_Event5)	E_OK	
Task16	ClearEvent(Task16_Event6)	E_OK	
Task16	ClearEvent(Task16_Event7)	E_OK	
Task16	ClearEvent(Task16_Event8)	E_OK	
Task16	TerminateTask()		

Test Sequence 15:

Test cases: Number of tasks which are not in the *suspended* state ≥ 8
Conformance Class: BCC1, BCC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1
type: basic

autostart: yes
priority: 8
max. activations: 1
preemptive: non, full

Task2

type: basic
autostart: no
priority: 7;
max. activations: 1
preemptive: non, full

Task3

type: basic
autostart: no
priority: 6
max. activations: 1
preemptive: non, full

Task4

type: basic
autostart: no
priority: 5
max. activations: 1
preemptive: non, full

Task5

type: basic
autostart: no
priority: 4
max. activations: 1
preemptive: non, full

Task6

type: basic
autostart: no
priority: 3
max. activations: 1
preemptive: non, full

Task7

type: basic
autostart: no
priority: 2
max. activations: 1
preemptive: non, full

Task8

type: basic
autostart: no
priority: 1
max. activations: 1
preemptive: non, full

Running task	**Called OS service**	**Return status**	**Test case**
Task1	`ActivateTask(Task2)`	`E_OK`	

Running task	Called OS service	Return status	Test case
Task1	`ActivateTask(Task3)`	`E_OK`	
Task1	`ActivateTask(Task4)`	`E_OK`	
Task1	`ActivateTask(Task5)`	`E_OK`	
Task1	`ActivateTask(Task6)`	`E_OK`	
Task1	`ActivateTask(Task7)`	`E_OK`	
Task1	`ActivateTask(Task8)`	`E_OK`	
Task1	`TerminateTask()`		
Task2	`TerminateTask()`		
Task3	`TerminateTask()`		
Task4	`TerminateTask()`		
Task5	`TerminateTask()`		
Task6	`TerminateTask()`		
Task7	`TerminateTask()`		
Task8	`TerminateTask()`		

3.2. Interrupt processing

The test cases 7 and 8 can not be tested, because more than one ISR is necessary. This leads to priority issues which are not covered by the OSEK OS specification.

Test Sequence 1:

Test cases: 1, 3, 6
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: yes
- priority: 1
- max. activations: 1
- preemptive: non, full

ISRs: ISR1
- category: 1

ISR2
- category: 2

ISR3
- category: 3

Running task	Called OS service	Return status	Test case
Task1	`EnableInterrupt(ISR1)`	`E_OK|E_OS_NOFUNC`	
Task1	`DisableInterrupt(ISR1)`	`E_OK`	3
Task1	*trigger interrupt ISR1*		
Task1	`EnableInterrupt(ISR1)`	`E_OK`	1
Task1	*trigger interrupt ISR1*		6

Running task	Called OS service	Return status	Test case
ISR1			
Task1	`EnableInterrupt(ISR2)`	`E_OK|E_OS_NOFUNC`	
Task1	`DisableInterrupt(ISR2)`	`E_OK`	3
Task1	*trigger interrupt ISR2*		
Task1	`EnableInterrupt(ISR2)`	`E_OK`	1
Task1	*trigger interrupt ISR2*		6
ISR2			
Task1	`EnableInterrupt(ISR3)`	`E_OK|E_OS_NOFUNC`	
Task1	`DisableInterrupt(ISR3)`	`E_OK`	3
Task1	*trigger interrupt ISR3*		
Task1	`EnableInterrupt(ISR3)`	`E_OK`	1
Task1	*trigger interrupt ISR3*		6
ISR3	`EnterISR()`		
ISR3	`LeaveISR()`		
Task1	`TerminateTask()`		

Test Sequence 2:

Test cases: 2, 4
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: yes
- priority: 1
- max. activations: 1
- preemptive: non, full

ISRs: ISR2
- category: 2

ISR3
- category: 3

Running task	Called OS service	Return status	Test case
Task1	`EnableInterrupt(ISR2)`	`E_OK|E_OS_NOFUNC`	
Task1	`EnableInterrupt(ISR2)`	`E_OS_NOFUNC`	2
Task1	`DisableInterrupt(ISR2)`	`E_OK`	
Task1	`DisableInterrupt(ISR2)`	`E_OS_NOFUNC`	4
Task1	`EnableInterrupt(ISR3)`	`E_OK|E_OS_NOFUNC`	
Task1	`EnableInterrupt(ISR3)`	`E_OS_NOFUNC`	2
Task1	`DisableInterrupt(ISR3)`	`E_OK`	
Task1	`DisableInterrupt(ISR3)`	`E_OS_NOFUNC`	4
Task1	`TerminateTask()`		

Test Sequence 3:

Test cases: 9, 10

Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-preemptive
Hooks: -
Tasks: Task1
type: basic
autostart: yes
priority: 1
max. activations: 1
preemptive: non
Task2
type: basic
autostart: no
priority: 2
max. activations: 1
preemptive: non
Task3
type: basic
autostart: no
priority: 3
max. activations: 1
preemptive: non
ISRs: ISR2
category: 2
ISR3
category: 3

Running task	Called OS service	Return status	Test case
Task1	`EnableInterrupt(ISR2|ISR3)`	`E_OK|E_OS_NOFUNC`	
Task1	*trigger interrupt ISR2*		
ISR2	`ActivateTask(Task2)`	`E_OK`	9
Task1	`TerminateTask()`		
Task2	*trigger interrupt ISR3*		
ISR3	`EnterISR()`		
ISR3	`ActivateTask(Task3)`	`E_OK`	10
ISR3	`LeaveISR()`		
Task2	`TerminateTask()`		
Task3	`TerminateTask()`		

Test Sequence 4:

Test cases 11, 12
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: mixed-, full-preemptive
Hooks: -
Tasks: Task1
type: basic
autostart: yes

priority: 1
max. activations: 1
preemptive: full

Task2

type: basic
autostart: no
priority: 2
max. activations: 1
preemptive: full

ISRs: ISR2

category: 2

ISR3

category: 3

Running task	Called OS service	Return status	Test case
Task1	`EnableInterrupt(ISR2|ISR3)`	`E_OK|E_OS_NOFUNC`	
Task1	*trigger interrupt ISR2*		
ISR2	`ActivateTask(Task2)`	`E_OK`	11
Task2	`TerminateTask()`		
Task1	*trigger interrupt ISR3*		
ISR3	`EnterISR()`		
ISR3	`ActivateTask(Task2)`	`E_OK`	12
ISR3	`LeaveISR()`		
Task2	`TerminateTask()`		
Task1	`TerminateTask()`		

3.3. Event mechanism

Test Sequence 1:

Test case: 1, 2, 3, 11, 12, 13, 15, 16, 17, 21, 22, 23, 24:
Conformance Class: ECC1, ECC2
Return Status: extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1

type: basic
autostart: yes
priority: 1
max. activations: 1
preemptive: non, full

Task2

type: extended
autostart: no
priority: 3
max. activations: 1
preemptive: non, full

resources: Resource1
events: Event1

ISRs: ISR2
category: 2
ISR3
category: 3

Resources: Resource1

Events: Event1

Running task	Called OS service	Return status	Test case
Task1	`EnableInterrupt(ISR2|ISR3)`	`E_OK|E_OS_NOFUNC`	
Task1	`SetEvent(INVALID_TASK, Event1)`	`E_OS_ID`	1
Task1	`SetEvent(Task1,Event1)`	`E_OS_ACCESS`	2
Task1	`SetEvent(Task2, Event1)`	`E_OS_STATE`	3
Task1	`ClearEvent(Event1)`	`E_OS_ACCESS`	11
Task1	*trigger interrupt ISR2*		
ISR2	`ClearEvent(Event1)`	`E_OS_CALLEVEL`	12
ISR2	`WaitEvent(Event1)`	`E_OS_CALLEVEL`	23
Task1	*trigger interrupt ISR3*		
ISR3	`EnterISR()`	`E_OK`	
ISR3	`ClearEvent(Event1)`	`E_OS_CALLEVEL`	13
ISR3	`WaitEvent(Event1)`	`E_OS_CALLEVEL`	24
ISR3	`LeaveISR()`		
Task1	`GetEvent(INVALID_TASK, &EventMask)`	`E_OS_ID`	15
Task1	`GetEvent(Task1, &EventMask)`	`E_OS_ACCESS`	16
Task1	`GetEvent(Task2, &EventMask)`	`E_OS_STATE`	17
Task1	`WaitEvent(Event1)`	`E_OS_ACCESS`	21
Task1	`ChainTask(Task2)`		
Task2	`GetResource(Resource1)`	`E_OK`	
Task2	`WaitEvent(Event1)`	`E_OS_RESOURCE`	22
Task2	`ReleaseResource(Resource1)`	`E_OK`	
Task2	`TerminateTask()`		

Test Sequence 2:

Test case 14, 18, 19, 20, 25, 26
Conformance Class: ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1
type: extended
autostart: no
priority: 1
max. activations: 1
preemptive: non, full
events: Event1

Task2
type: extended
autostart: yes
priority: 2
max. activations: 1
preemptive: non, full
events: Event2

Events: Event1, Event2

Running task	Called OS service	Return status	Test case
Task2	`ActivateTask(Task1)`	`E_OK`	
Task2	`GetEvent(Task1, &EventMask)`	`E_OK, EventMask=0x0`	19
Task2	`WaitEvent(Event2)`	`E_OK`	25
Task1	`GetEvent(Task2, &EventMask)`	`E_OK, EventMask=0x0`	20
Task1	`SetEvent(Task1, Event1)`	`E_OK`	
Task1	`GetEvent(Task1, &EventMask)`	`E_OK, EventMask=Event1`	18
Task1	`WaitEvent(Event1)`	`E_OK`	26
Task1	`ClearEvent(Event1)`	`E_OK`	14
Task1	`GetEvent(Task1, &EventMask)`	`E_OK, EventMask=0x0`	
Task1	`SetEvent(Task2, Event2)`	`E_OK`	
Task1	*force scheduling*		
Task2	`TerminateTask()`		
Task1	`TerminateTask()`		

Test Sequence 3:

Test case 4, 5, 9
Conformance Class: ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-preemptive
Hooks: -
Tasks: Task1
type: basic
autostart: yes
priority: 1
max. activations: 1
preemptive: non
Task2
type: extended
autostart: yes
priority: 2
max. activations: 1
preemptive: non
events: Event1, Event2, Event3
Events: Event1, Event2, Event3

Running task	Called OS service	Return status	Test case
Task2	`WaitEvent(Event1)`	`E_OK`	
Task1	`GetTaskState(Task2, &TaskState)`	`E_OK, TaskState=WAITING`	
Task1	`GetEvent(Task2, &EventMask)`	`E_OK, EventMask=0x0`	
Task1	`SetEvent(Task2, Event2)`	`E_OK`	5
Task1	`GetTaskState(Task2, &TaskState)`	`E_OK, TaskState=WAITING`	
Task1	`GetEvent(Task2, &EventMask)`	`E_OK, EventMask=Event2`	
Task1	`SetEvent(Task2, Event1)`	`E_OK`	4
Task1	`GetTaskState(Task2, &TaskState)`	`E_OK, TaskState=READY`	
Task1	`GetEvent(Task2, &EventMask)`	`E_OK, EventMask=Event1|Event2`	
Task1	`SetEvent(Task2, Event3)`	`E_OK`	9
Task1	`GetEvent(Task1, &EventMask)`	`E_OK, EventMask=Event1|Event2|Event3`	
Task1	`TerminateTask()`		
Task2	`TerminateTask()`		

Test Sequence 4:

Test case 6, 7, 8, 10
Conformance Class: ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: mixed-, full-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: yes
- priority: 2
- max. activations: 1
- preemptive: full

Task2
- type: extended
- autostart: yes
- priority: 3
- max. activations: 1
- preemptive: full
- events: Event1, Event2

Task3
- type: extended
- autostart: no
- priority: 1
- max. activations: 1
- preemptive: full
- events: Event3

Task4

type: basic
autostart: no
priority: 4
max. activations: 1
preemptive: full

Events: Event1, Event2, Event3

Running task	Called OS service	Return status	Test case
Task2	`WaitEvent(Event1)`	`E_OK`	
Task1	`SetEvent(Task2, Event2)`	`E_OK`	8
Task1	`GetTaskState(Task2, &TaskState)`	`E_OK, TaskState=WAITING`	
Task1	`GetEvent(Task2, &EventMask)`	`E_OK, EventMask=Event2`	
Task1	`ActivateTask(Task3)`	`E_OK`	
Task1	`GetTaskState(Task3, &TaskState)`	`E_OK, TaskState=READY`	
Task1	`SetEvent(Task3, Event3)`	`E_OK`	10
Task1	`GetEvent(Task3, &EventMask)`	`E_OK, EventMask=Event3`	
Task1	`SetEvent(Task2, Event1)`	`E_OK`	6
Task2	`ClearEvent(Event1)`	`E_OK`	
Task2	`WaitEvent(Event1)`	`E_OK`	
Task1	`ActivateTask(Task4)`	`E_OK`	
Task4	`SetEvent(Task2, Event1)`	`E_OK`	7
Task4	`GetTaskState(Task2, &TaskState)`	`E_OK, TaskState=READY`	
Task4	`TerminateTask()`		
Task2	`TerminateTask()`		
Task1	`TerminateTask()`		
Task3	`TerminateTask()`		

3.4. Resource management

Test Sequence 1:

Test case: 1, 2, 3, 4, 5, 9, 10, 11, 12
Conformance Class: BCC2, ECC1, ECC2
Return Status: extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1

type: basic
autostart: yes
priority: 1
max. activations: 1
preemptive: non, full

RESOURCE =
Resource1, Resource2, Resource3, Resource4, Resource5, Resource6

Task2

type: basic
autostart: no
priority: 2
max. activations: 1
preemptive: non, full
RESOURCE =
ResourceA

ISRs: ISR2

category: 2

ISR3

category: 3

Resources: Resource1, Resource2, Resource3, Resource4, Resource5, Resource6, ResourceA

Running task	Called OS service	Return status	Test case
Task1	`EnableInterrupt(ISR2|ISR3)`	`E_OK|E_OS_NOFUNC`	
Task1	`GetResource(ResourceA)`	`E_OS_ACCESS`	1
Task1	`GetResource(NoResource)`	`E_OS_ID`	2
Task1	`GetResource(Resource1)`	`E_OK`	
Task1	`GetResource(Resource2)`	`E_OK`	
Task1	`GetResource(Resource3)`	`E_OK`	
Task1	`GetResource(Resource4)`	`E_OK`	
Task1	`GetResource(Resource5)`	`E_OK`	
Task1	`GetResource(Resource6)`	`E_OS_LIMIT|E_OK`	5
Task1	`ReleaseResource(Resource6)`	`E_OK`	
Task1	`ReleaseResource(Resource5)`	`E_OK`	
Task1	`ReleaseResource(Resource4)`	`E_OK`	
Task1	`ReleaseResource(Resource3)`	`E_OK`	
Task1	`ReleaseResource(Resource2)`	`E_OK`	
Task1	`ReleaseResource(Resource1)`	`E_OK`	
Task1	*trigger interrupt ISR2*		
ISR2	`GetResource(Resource1)`	`E_OS_CALLEVEL`	3
ISR2	`ReleaseResource(Resource1)`	`E_OS_CALLEVEL`	10
Task1	*trigger interrupt ISR3*		
ISR3	`EnterISR()`	`E_OK`	
ISR3	`GetResource(Resource1)`	`E_OS_CALLEVEL`	4
ISR3	`ReleaseResource(Resource1)`	`E_OS_CALLEVEL`	11
ISR3	`LeaveISR()`		
Task1	`ReleaseResource(Resource1)`	`E_OS_NOFUNC`	12
Task1	`ReleaseResource(NoResource)`	`E_OS_ID`	9
Task1	`TerminateTask()`		

Test Sequence 2:

Test case: 6, 13
Conformance Class: BCC2, ECC1, ECC2

Return Status: standard, extended
Scheduling Policy: non-, mixed-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: yes
- priority: 1
- max. activations: 1
- preemptive: non
- resources: Resource1

Task2
- type: basic
- autostart: no
- priority: 2
- max. activations: 1
- preemptive: non
- resources: Resource1

Task3
- type: basic
- autostart: no
- priority: 3
- max. activations: 1
- preemptive: non

Resources: Resource1

Running task	Called OS service	Return status	Test case
Task1	`GetResource(Resource1)`	`E_OK`	6
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	*force scheduling*		
Task1	`ActivateTask(Task3)`	`E_OK`	
Task1	*force scheduling*	`E_OK`	
Task3	`TerminateTask()`		
Task1	`ReleaseResource(Resource1)`	`E_OK`	13
Task1	*force scheduling*	`E_OK`	
Task2	`TerminateTask()`		
Task1	`TerminateTask()`		

Test Sequence 3:

Test case: 7, 14
Conformance Class: BCC2, ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: mixed-, full-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: yes
- priority: 1
- max. activations: 1

preemptive: full
resources: Resource1

Task2

type: basic
autostart: no
priority: 2
max. activations: 1
preemptive: full
resources: Resource1

Task3

type: basic
autostart: no
priority: 3
max. activations: 1
preemptive: full

Resources: Resource1

Running task	Called OS service	Return status	Test case
Task1	`GetResource(Resource1)`	`E_OK`	7
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	`ActivateTask(Task3)`	`E_OK`	
Task3	`TerminateTask()`		
Task1	`ReleaseResource(Resource1)`	`E_OK`	14
Task2	`TerminateTask()`		
Task1	`TerminateTask()`		

Test Sequence 4:

Test case: 8, 15, 16
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1

type: basic
autostart: yes
priority: 1
max. activations: 1
preemptive: non, full

Task2

type: basic
autostart: no
priority: 2
max. activations: 1
preemptive: non, full

Task3

type: basic
autostart: no
priority: 3

max. activations: 1
preemptive: non, full

Running task	Called OS service	Return status	Test case
Task1	`GetResource(RES_SCHEDULER)`	`E_OK`	8
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	*force scheduling*		
Task1	`ActivateTask(Task3)`	`E_OK`	
Task1	*force scheduling*		
Task1	`ReleaseResource(RES_SCHEDULER)`	`E_OK`	15, 16
Task1	*force scheduling*		
Task3	`TerminateTask()`		
Task2	`TerminateTask()`		
Task1	`TerminateTask()`		

Test Sequence 5:

Test case: 2, 3, 4, 9, 10, 11, 12
Conformance Class: BCC1
Return Status: extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1
type: basic
autostart: yes
priority: 1
max. activations: 1
preemptive: non, full
ISRs: ISR2
category: 2
ISR3
category: 3

Running task	Called OS service	Return status	Test case
Task1	`EnableInterrupt(ISR2|ISR3)`	`E_OK|E_OS_NOFUNC`	
Task1	`GetResource(NoResource)`	`E_OS_ID`	2
Task1	*trigger interrupt ISR2*		
ISR2	`GetResource(Resource1)`	`E_OS_CALLEVEL`	3
ISR2	`ReleaseResource(Resource1)`	`E_OS_CALLEVEL`	10
Task1	*trigger interrupt ISR3*		
ISR3	`EnterISR()`	`E_OK`	
ISR3	`GetResource(Resource1)`	`E_OS_CALLEVEL`	4
ISR3	`ReleaseResource(Resource1)`	`E_OS_CALLEVEL`	11
ISR3	`LeaveISR()`		
Task1	`ReleaseResource(RES_SCHEDULER)`	`E_OS_NOFUNC`	12
Task1	`ReleaseResource(NoResource)`	`E_OS_ID`	9
Task1	`TerminateTask()`		

3.5. Alarms

Test case29 can not be tested, because it is not possible to trigger the alarm's counter while no task is running.

Test Sequence 1

Test case: 1, 3, 7, 10, 11, 12, 13, 16, 19, 20, 21, 22, 25
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: yes
- priority: 1
- max. activations: 1
- preemptive: non, full

Counters: Counter1
- max. allowed: 16
- ticks per base: 1
- min. cycle: 2

Alarms: Alarm1
- counter: Counter1
- action: activate task
- task: Task1

Running task	Called OS service	Return status	Test case
Task1	`GetAlarmBase(NoAlarm, &AlarmBase)`	`E_OS_ID`	1
Task1	`GetAlarm(NoAlarm, &Tick)`	`E_OS_ID`	3
Task1	`GetAlarmBase(Alarm1, &AlarmBase)`	`E_OK`	
Task1	`SetRelAlarm(NoAlarm, 0, 0)`	`E_OS_ID`	7
Task1	`SetRelAlarm(Alarm1, -1, 0)`	`E_OS_VALUE`	10
Task1	`SetRelAlarm(Alarm1, AlarmBase.maxallowedvalue+1, 0)`	`E_OS_VALUE`	11
Task1	`SetRelAlarm(Alarm1, 0, AlarmBase.mincycle-1)`	`E_OS_VALUE`	12
Task1	`SetRelAlarm(Alarm1, 0, AlarmBase.maxallowedvalue+1)`	`E_OS_VALUE`	13
Task1	`SetAbsAlarm(NoAlarm, 0, 0)`	`E_OS_ID`	16
Task1	`SetAbsAlarm(Alarm1, -1, 0)`	`E_OS_VALUE`	19
Task1	`SetAbsAlarm(Alarm1, AlarmBase.maxallowedvalue+1, 0)`	`E_OS_VALUE`	20
Task1	`SetAbsAlarm(Alarm1, 0, AlarmBase.mincycle-1)`	`E_OS_VALUE`	21
Task1	`SetAbsAlarm(Alarm1, 0, AlarmBase.maxallowedvalue+1)`	`E_OS_VALUE`	22
Task1	`CancelAlarm(NoAlarm)`	`E_OS_ID`	25
Task1	`TerminateTask()`		

Test Sequence 2:

Test cases:	2, 4, 5, 8, 14, 17, 23, 26, 27	
Conformance Class:	BCC1, BCC2, ECC1, ECC2	
Return Status:	standard, extended	
Scheduling Policy:	non-, mixed-, full-preemptive	
Hooks:	-	
Tasks:	Task1	
	type:	basic
	autostart:	yes
	priority:	1
	max. activations:	1
	preemptive:	non, full
	Task2	
	type:	basic
	autostart:	no
	priority:	2
	max. activations:	1
	preemptive:	non, full
Counters:	Counter1	
	max. allowed:	16
	ticks per base:	1
	min. cycle:	1
Alarms:	Alarm1	
	counter:	Counter1
	action:	activate task
	task:	Task2

Running task	Called OS service	Return status	Test case
Task1	`GetAlarmBase(Alarm1, &AlarmBase)`	`E_OK`	2
Task1	`GetAlarm(Alarm1, &Tick)`	`E_OS_NOFUNC`	4
Task1	`CancelAlarm(Alarm1)`	`E_OS_NOFUNC`	26
Task1	*initialize alarm counter*		
Task1	`SetAbsAlarm(Alarm1, 1, 1)`	`E_OK`	23
Task1	`SetAbsAlarm(Alarm1, 3, 0)`	`E_OS_STATE`	17
Task1	*increment alarm counter*		
Task1	*force scheduling*		
Task2	`TerminateTask()`		
Task1	*increment alarm counter*		
Task1	*force scheduling*		
Task2	`TerminateTask()`		
Task1	`CancelAlarm(Alarm1)`	`E_OK`	27
Task1	`SetRelAlarm(Alarm1, 1, 0)`	`E_OK`	14
Task1	`SetRelAlarm(Alarm1, 2, 0)`	`E_OS_STATE`	8
Task1	`GetAlarm(Alarm1, &Tick)`	`E_OK, Tick=1`	5
Task1	*increment alarm counter*		
Task1	*force scheduling*		
Task2	`TerminateTask()`		

Running task	Called OS service	Return status	Test case
Task1	`TerminateTask()`		

Test Sequence 3

Test cases: 6, 9, 15, 18, 24, 28, 34
Conformance Class: ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: yes
- priority: 1
- max. activations: 1
- preemptive: non, full

Task2
- type: extended
- autostart: yes
- priority: 2
- max. activations: 1
- preemptive: non, full
- events: Event2

Events: Event2
Counters: Counter1
- max. allowed: 16
- ticks per base: 1
- min. cycle: 1

Alarms: Alarm1
- counter: Counter1
- action: set event
- task: Task2
- event: Event2

Running task	Called OS service	Return status	Test case
Task2	`WaitEvent(Event2)`	`E_OK`	
Task1	*initialize alarm counter*		
Task1	`SetAbsAlarm(Alarm1, 1, 1)`	`E_OK`	24
Task1	`SetAbsAlarm(Alarm1, 3, 0)`	`E_OS_STATE`	18
Task1	*increment alarm counter*		34
Task1	*force scheduling*		
Task2	`ClearEvent(Event2)`	`E_OK`	
Task2	`WaitEvent(Event2)`	`E_OK`	
Task1	*increment alarm counter*		
Task1	*force scheduling*		
Task2	`ClearEvent(Event2)`	`E_OK`	
Task2	`WaitEvent(Event2)`	`E_OK`	
Task1	`CancelAlarm(Alarm1)`	`E_OK`	28

Running task	Called OS service	Return status	Test case
Task1	`SetRelAlarm(Alarm1, 1, 0)`	`E_OK`	15
Task1	`SetRelAlarm(Alarm1, 2, 0)`	`E_OS_STATE`	9
Task1	`GetAlarm(Alarm1, &Tick)`	`E_OK, Tick = 1`	6
Task1	*increment alarm counter*		34
Task1	*force scheduling*		
Task2	`TerminateTask()`	`E_OK`	
Task1	`TerminateTask()`		

Test Sequence 4:

Test cases: 30
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-preemptive
Hooks: -
Tasks: Task1
- type: basic
- autostart: yes
- priority: 1
- max. activations: 1
- preemptive: non

Task2
- type: basic
- autostart: no
- priority: 2
- max. activations: 1
- preemptive: non

Counters: Counter1
- max. allowed: 16
- ticks per base: 1
- min. cycle: 1

Alarms: Alarm1
- counter: Counter1
- action: activate task
- task: Task2

Running task	Called OS service	Return status	Test case
Task1	`SetRelAlarm(Alarm1, 1, 0)`	`E_OK`	
Task1	*increment alarm counter*		30
Task1	`GetTaskState(Task2, &TaskState)`	`E_OK, TaskState=READY`	
Task1	`TerminateTask()`		
Task2	`TerminateTask()`		

Test Sequence 5:

Test case 31, 32
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: standard, extended

Scheduling Policy: mixed-, full-preemptive
Hooks: -
Tasks: Task1
type: basic
autostart: yes
priority: 1
max. activations: 1
preemptive: full
Task2
type: basic
autostart: no
priority: 2;
max. activations: 1
preemptive: full
Task3
type: basic
autostart: no
priority: 3
max. activations: 1
preemptive: full
Counters: Counter1
max. allowed: 16
ticks per base: 1
min. cycle: 1
Alarms: Alarm1
counter: Counter1
action: activate task
task: Task2

Running task	**Called OS service**	**Return status**	**Test case**
Task1	`SetRelAlarm(Alarm1, 1, 0)`	`E_OK`	
Task1	*increment alarm counter*		31
Task2	`TerminateTask()`		
Task1	`ChainTask(Task3)`		
Task3	`SetRelAlarm(Alarm1, 1, 0)`	`E_OK`	
Task3	*increment alarm counter*		32
Task3	`GetTaskState(Task2, &State)`	`E_OK, TaskState=READY`	
Task3	`TerminateTask()`		
Task2	`TerminateTask()`		

Test Sequence 6:

Test cases: 33, 34
Conformance Class: ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-preemptive
Hooks: -
Tasks: Task1
type: basic

	autostart:	yes
	priority:	1
	max. activations:	1
	preemptive:	non
Task2		
	type:	extended
	autostart:	no
	priority:	2
	max. activations:	1
	preemptive:	non
	events:	Event1
Events:	Event1	
Counters:	Counter1	
	max. allowed:	16
	ticks per base:	1
	min. cycle:	1
Alarms:	Alarm1	
	counter:	Counter1
	action:	set event
	task:	Task2
	event:	Event1

Running task	Called OS service	Return status	Test case
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	`SetRelAlarm(Alarm1, 1, 0)`	`E_OK`	
Task1	*increment alarm counter*		33
Task1	`GetEvent(Task2, &EventMask)`	`E_OK, EventMask=Event1`	
Task1	`Schedule()`	`E_OK`	
Task2	`ClearEvent(Event1)`	`E_OK`	
Task2	`WaitEvent(Event1)`	`E_OK`	
Task1	`SetRelAlarm(Alarm1, 1, 0)`	`E_OK`	
Task1	*increment alarm counter*		34
Task1	`GetTaskState(Task2, &TaskState)`	`E_OK, TaskState=READY`	
Task1	`TerminateTask()`		
Task2	`TerminateTask()`		

Test Sequence 7:

Test cases:	35, 36	
Conformance Class:	ECC1, ECC2	
Return Status:	standard, extended	
Scheduling Policy:	mixed-, full-preemptive	
Hooks:	-	
Tasks:	Task1	
	type:	basic
	autostart:	yes
	priority:	1

max. activations: 1
preemptive: full

Task2

type: extended
autostart: yes
priority: 2
max. activations: 1
preemptive: full
events: Event2

Task3

type: basic
autostart: no
priority: 3
max. activations: 1
preemptive: full

Task4

type: basic
autostart: no
priority: 4
max. activations: 1
preemptive: full

Events: Event2

Counters: Counter1

max. allowed: 16
ticks per base: 1
min. cycle: 1

Alarms: Alarm1

counter: Counter1
action: set event
task: Task2
event: Event2

Running task	**Called OS service**	**Return status**	**Test case**
Task2	`WaitEvent(Event2)`	`E_OK`	
Task1	`ActivateTask(Task3)`	`E_OK`	
Task3	`SetRelAlarm(Alarm1, 1, 0)`	`E_OK`	
Task3	*increment alarm counter*		36
Task3	`GetTaskState(Task2, &TaskState)`	`E_OK, TaskState=READY`	
Task3	`TerminateTask()`		
Task2	`ClearEvent(Event2)`	`E_OK`	
Task2	`ActivateTask(Task4)`	`E_OK`	
Task4	`SetRelAlarm(Alarm1, 1, 0)`	`E_OK`	
Task4	*increment alarm counter*		35
Task4	`GetEvent(Task2, &EventMask)`	`E_OK, EventMask=Event2`	
Task4	`TerminateTask()`		
Task2	`TerminateTask()`		
Task1	`TerminateTask()`		

3.6. Error handling, hook routines and OS execution control

Test case 2 (call `StartOS()` to start OSEK OS) can not be tested, because the startup code is implementation specific.

Test Sequence 1:

Test cases: 4, 5, 6
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: StartupHook, ErrorHook, PreTaskHook, PostTaskHook
Tasks: Task1
- type: basic
- autostart: no
- priority: 1
- max. activations: 1
- preemptive: non, full

Task2
- type: basic
- autostart: no
- priority: 2
- max. activations: 1
- preemptive: non, full

Counters: Counter1
- max. allowed: 16
- ticks per base: 1
- min. cycle: 1

Alarms: Alarm1
- counter: Counter1
- action: activate task
- task: Task2

Running task	Called OS service	Return status	Test case
Startup-Hook	`ActivateTask(Task1)`	`E_OK`	6
PreTask-Hook	`GetTaskID(&TaskID)`	`E_OK, TaskID=Task1`	4
PreTask-Hook	`GetTaskState(Task1, &TaskState)`	`E_OK, TaskState=RUNNING`	
PreTask-Hook	`GetTaskState(Task2, &TaskState)`	`E_OK, TaskState=SUSPENDED`	
Task1	`ActivateTask(Task2)`	`E_OK`	
Task1	*force scheduling*		
PostTask-Hook	`GetTaskID(&TaskID)`	`E_OK, TaskID=Task1`	4
PostTask-Hook	`GetTaskState(Task1, &TaskState)`	`E_OK, TaskState=RUNNING`	

Running task	Called OS service	Return status	Test case
PostTask-Hook	GetTaskState(Task2, &TaskState)	E_OK, TaskState=READY	
PreTask-Hook	GetTaskID(&TaskID)	E_OK, TaskID=Task2	4
PreTask-Hook	GetTaskState(Task1, &TaskState)	E_OK, TaskState=READY	
PreTask-Hook	GetTaskState(Task2, &TaskState)	E_OK, TaskState=RUNNING	
Task2	GetAlarm(Alarm1, &Tick)	E_OS_NOFUNC	
Error-Hook			5
Task2	TerminateTask()		
PostTask-Hook	GetTaskID(&TaskID)	E_OK, TaskID=Task2	4
PostTask-Hook	GetTaskState(Task1, &TaskState)	E_OK, TaskState=READY	
PostTask-Hook	GetTaskState(Task2, &TaskState)	E_OK, TaskState=RUNNING	
PreTask-Hook	GetTaskID(&TaskID)	E_OK, TaskID=Task1	4
PreTask-Hook	GetTaskState(Task1, &TaskState)	E_OK, TaskState=RUNNING	
PreTask-Hook	GetTaskState(Task2, &TaskState)	E_OK, TaskState=SUSPENDED	
Task1	TerminateTask()	E_OK	
PostTask-Hook	GetTaskID(&TaskID)	E_OK, TaskID=Task1	4
PostTask-Hook	GetTaskState(Task1, &TaskState)	E_OK, TaskState=RUNNING	
PostTask-Hook	GetTaskState(Task2, &TaskState)	E_OK, TaskState=SUSPENDED	

Test Sequence 2:

Test cases: 1, 3, 7
Conformance Class: BCC1, BCC2, ECC1, ECC2
Return Status: standard, extended
Scheduling Policy: non-, mixed-, full-preemptive
Hooks: ShutdownHook
Tasks: Task1

- type: basic
- autostart: yes
- priority: 1
- max. activations: 1
- preemptive: non, full

Running task	Called OS service	Return status	Test case
Task1	`GetActiveApplicationMode(&Mode)`	`E_OK`	1
Task1	`ShutdownOS()`		3
Shutdown Hook			7

4. Abbreviations

API	Application Programming Interface
COM	Communication
DLL	Data Link Layer
ECU	Electronic Control Unit
ISO	International Standard Organization
ISR	Interrupt Service Routine
IUT	Implementation Under Test
LT	Lower Tester
NM	Network Management
OPDU	OSEK Protocol Data Unit
OS	Operating System
PDU	Protocol Data Unit
PCO	Point of Control and Observation
SDL	Specification and Description Language
TMP	Test Management Protocol
TM_PDU	Test Management - Protocol Data Unit
TTCN	Tree and Tabular Combined Notation
UT	Upper Tester

5. References

[1] OSEK/VDX Conformance Testing Methodology - Version 1.0 - 19th of December 1997

[2] OSEK/VDX OS Test Plan - Version 1.0 - 4th of March 1998

[3] OSEK/VDX Certification Procedure - F. Kaag, J. Minuth, K.J. Neumann, H. Kuder - Proceedings of the 1st International Workshop on Open Systems in Automotive Networks - October 1995.

[4] OSEK/VDX Operating System - Version 2.0 revision 1- 15th of October1997

[5] ISO/IEC 9646-1 - Information technology, Open Systems Interconnection, Conformance testing methodology and framework, part 1 : General Concepts, 1992.

[6] ISO/IEC 9646-3 - Information technology, Open Systems Interconnection, Conformance testing, methodology and framework, part 3 : The Tree and Tabular Combined Notation (TTCN), 1992.

[7] Benutzerdokumentation "Classification-Tree Editor - CTE für MS-Windows", Version 1.2 - ATS Automated Testing Solutions GmbH, Daimler-Benz AG, 1st of February 1998.