OSEK/VDX

Binding Specification

Version 1.4

06th September 2002

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 harmonised with VDX. For simplicity OSEK will be used instead of OSEK/VDX in the document.

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 software of control units of various companies.
- Standardised interfacing features for control units with different architectural designs.
- Sequenced utilisation of the intelligence (existing resources) distributed in the vehicle, to enhance the performance of the overall system without requiring additional hardware.
- Provides independence with regards to individual implementation, as the specification does not prescribe implementation aspects.

Scope of the document :

As the standardisation of requirements that are applicable to different OSEK/VDX specifications should not be replicated within the different specifications, this document is therefore set-up to collate all requirements that are owned by the different specifications. This document also provides an overall description of the OSEK/VDX specifications set.

This binding specification is therefore a 'normative' document and intention is laid to prevent a divergence of the OSEK/VDX specifications by giving the possibility to refer to a single binding document.

Table of Contents

List of Figures

List of Tables

1 General description (informative)

The OSEK/VDX specification set consists of 4 normative documents that define the requirements of two choices of operating systems (real-time executive for ECU's) with two choices of communication features (data exchange within and between ECU's) linked to the respective operating systems, and network management strategies (configuration determination and monitoring).

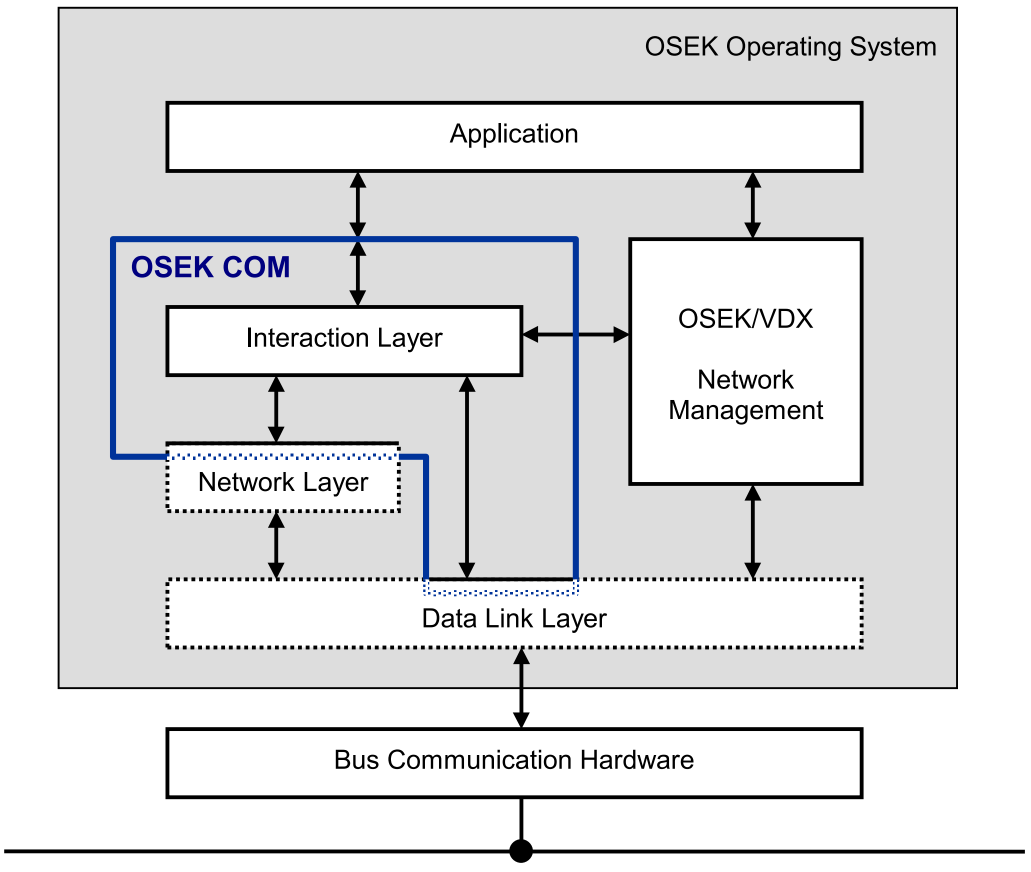

Figure 1-1: Layer model of OSEK/VDX with OSEK OS

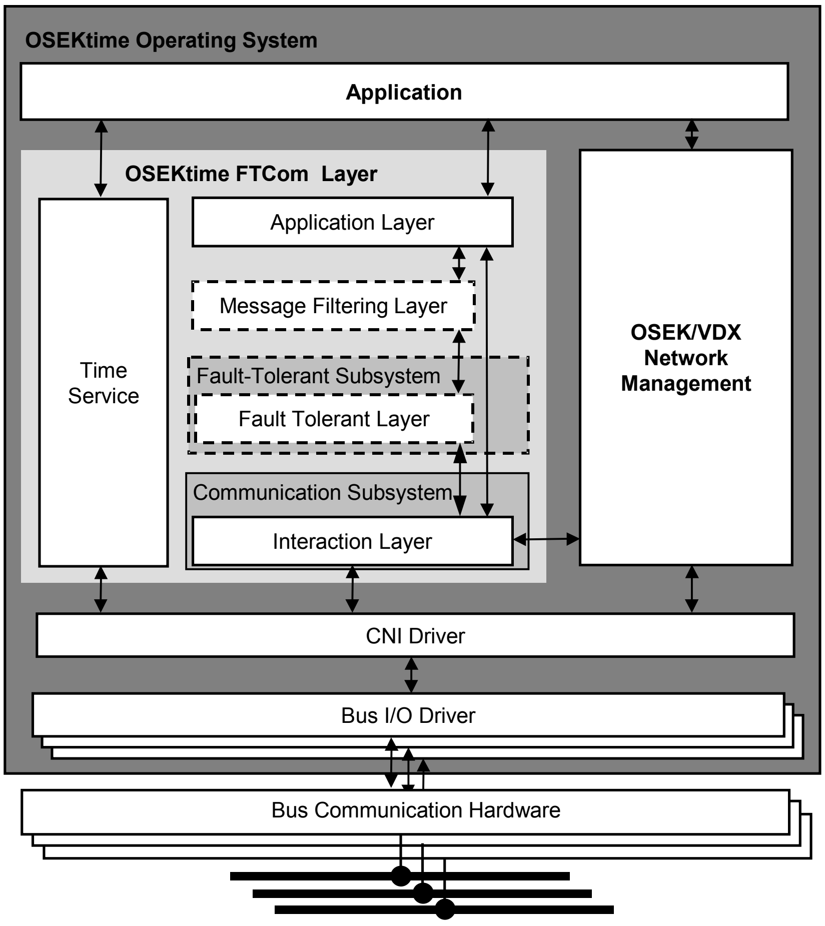

Figure 1-2 Layer model of OSEK/VDX with OSEKtime OS

OSEK/VDX operating system (OS)

The specification of the OSEK/VDX OS provides a pool of services and processing mechanisms. The operating system serves as a basis for the controlled real-time execution of concurrent application and provides their environment on a processor. The architecture of the OSEK/VDX OS distinguishes three processing levels: interrupt level, a logical level for operating systems activities and task level. The interrupt level is assigned higher priorities than the task level. In addition to the management of the processing levels, operating system services are provided for functionality like task management, event management, resource management, counter, alarm and error treatment.

OSEK/VDX communication (COM)

The communication specification provides interfaces for the transfer of data within vehicle networks systems. This communication takes place between and within network stations (ECU's). The positioning of OSEK/VDX COM within the OSEK/VDX architecture is represented in Figure 1-1. It shows the interface with the application, OSEK/VDX network management and the hardware communication bus. This specification defines an interaction layer and requirements to the underlying network layer and/or data link layer. The interaction layer provides the application programming interface (API) of OSEK/VDX COM to support the transfer of messages within and between network stations. For network communication, the interaction layer uses services provided by the lower layers. ECU-internal communication is handled by the interaction layer only.

OSEK/VDX OSEKtime – Operating System

The OSEKtime operating system provides the necessary services to support distributed fault-tolerant highly dependable real-time applications (e.g., start-up of the system, message handling, state message interface, interrupt processing, synchronisation and error handling).

The operating system is built according to the user's configuration instructions at system generation time. The operating system cannot be modified later at execution time.

Described in the operating system specification is also the Time Service specification. The Time Services support the synchronisation of the task execution due to a global time base during system start-up and repeatedly during normal operation (e.g., at every end of the dispatching table). The Time Service functionality are usually close connected to the communication layer, therefore the Time Service is implemented in FTCom.

If the functionality of both OSEKtime OS and OSEK OS is needed, it is possible to run both systems in parallel (not shown in Figure 1-2).

OSEK/VDX Fault-Tolerant Communication FTCom

FTCom is divided into the layers: Application, Message Filtering, Fault Tolerant, and Interaction. The Application layer provides the Application Programming Interface (API). The Message Filtering layer provides mechanisms for message filtering. The Fault Tolerant layer provides services required to support the fault-tolerant functionality, that includes judgement mechanisms for message instance management and support of message status information. The Interaction layer provides services for the transfer of message instances via network: Resolves issues connected with the presentation of a message instance on different hosts (e.g. different byte ordering), provides a message instance packing/unpacking service and supports a message instance status information. For efficiency reasons the Filter and Fault Tolerant layer are optional.

OSEK/VDX network management (NM)

Network serves as a basis for new distributed control functions that are independent of local ECU platforms. As a consequence of networking, the local station behaviour influences and depends on the global behaviour, and vice versa. The mutual influences and dependencies often require network wide negotiated management. In order to guarantee the reliability and safety of a distributed system, the OSEK/VDX NM gives support for several of such management tasks. The basic concept of OSEK/VDX NM is monitoring network stations. Two alternatives monitoring mechanisms are offered: direct and indirect station monitoring. Direct monitoring is performed by a dedicated communication of the network management. Direct monitoring of the nodes may be impossible or undesirable. This could be the case for example for very simple or time critical applications. The mechanism of indirect monitoring is therefore available. It is based on the use of monitored application messages. This indirect monitoring is limited to nodes that send messages in the course of normal operation.

OSEK/VDX Implementation Language (OIL)

To reach the original goal of the OSEK/VDX project in having portable software, a way of describing an OSEK/VDX system is defined. This is the motivation for definition of a standardised OSEK/VDX Implementation language, called OIL.

OSEK/VDX Run Time Interface (ORTI)

To provide debugging support on the level of OSEK objects, it is necessary to have debuggers available that are capable of displaying and debugging OSEK components. The ORTI specification provides an interface for debugging and monitoring tools to access OSEK objects in target memory. Tools can evaluate internal data structures of OSEK objects and their location in memory. ORTI is consisting of a language to describe kernel objects (KOIL, Kernel Object Interface Language) and a description of OSEK specific objects and attributes.

2 Bindings configuration (normative)

Multiple bindings of the OSEK/VDX specifications are available, each reflecting development efforts resulting from OSEK/VDX evaluations and introduction of new requirements. Interfaces between the OSEK/VDX specifications are guaranteed within the following binding sets. The following tables list the issues of all OSEK/VDX specifications applicable to each particular binding reference. Two binding references are created to document on one hand the configuration of OSEK/VDX specifications (referred to as 'SB') and on the other hand the configuration of OSEK/VDX certification plans (referred to as 'CB').

2.1 Binding index of OSEK/VDX specifications :

Specification binding identifier:	**SB0**	**SB1**	**SB2**	**SB3**	**SB4**	**SB5**
Binding	-	-	-	**1.2**	**1.3**	**1.4**
Operating System (OS)	**1.0**	**2.0r1**	2.0r1	**2.1r1**	**2.2**	2.2, **2.2.1**
Communication (COM)	**1.0**	**2.0a**	**2.1r1**	**2.2.2**	2.2.2	**3.0, 3.0.1**
Network management (NM)	**1.0**	**2.1**	**2.5**	**2.5.1**	2.5.1	2.5.1, **2.5.2**
OSEK Implementation Language (OIL)	**-**	**2.0**	**2.1**	**2.2**	**2.3**	**2.4**
OSEKtime OS				**1.0**	1.0	1.0
OSEKtime COM (FTCOM)				**1.0**	1.0	1.0

Table 2-1 : Binding index OSEK/VDX specifications

Note: OS 2.2.1, COM 3.0.1 and NM 2.5.2 are planed documentation updates with no functional changes, to reflect wording changes done during ISO standardisation (ISO 17356).

2.2 Binding index of OSEK/VDX certification plans :

Certification binding identifier:	**CB0**	**CB1**	**CB2**	**CB3**	**CB4**	**CB5**
Conformance methodology	**-**	**2.0**	2.0	**3.0**	**4.0**	**4.5**
OS test plan	**-**	**2.0**	2.0	**3.0**	**4.0**	**4.5**
COM test plan	**-**	**-**	**2.0**	**3.0**	**4.0**	**4.5**
NM test plan	**-**	**-**	**2.0**	**3.0**	**4.0**	**4.5**
OSEKtime test plan				**-**	**-**	**-**

Table 2-2 : Binding index OSEK certification plans

3 Common requirements specification (normative)

3.1 Definition of error codes

The different parts of OSEK/VDX (e.g. OSEK OS, OSEK COM) specify return codes of system functions to indicate different conditions which can arise during performing the system function. These return codes of type *StatusType* are defined as define-variables in the respective documentation. However, because these return codes can not be seen locally (e.g. they are used as input parameter to *ShutdownOS*), unique values have to be defined across the different specifications.

To accommodate this, ranges of error code values have been defined which are assigned to the different parts of the specification. Each range consists of 32 values. Within each range, the first up to 16 values are consecutively defined as standard return values. Starting with the second half of the range, the second 16 values may be defined consecutively to inform about detection of implementation specific additional errors (e.g. stack overflow, corruption of internal lists etc.).

Within the first range, the value '0' (E_OK) has a special meaning. It indicates the successful completion of a system function without any specific return indication.

The ranges are assigned as follows:

0	E_OK
1 to 31	OSEK OS error codes
32 to 63	OSEK COM error codes
64 to 95	OSEK NM error codes
96 to 127	OSEKtime OS error codes
128 to 159	OSEKtime FTCom error codes
160 to 255	OSEK RESERVED

3.2 Definition of StatusType

The data type *StatusType* is used within all parts of OSEK/VDX. To be able to combine different parts of OSEK/VDX from different supplies (e.g. OSEK COM from supplier A, OSEK NM from supplier B), the definition of this type has to be handled with care to avoid conflicts.

Conflicts can arise if the definitions are different between the different parts of OSEK/VDX. Moreover, even if the definitions are the same, the compiler will have to create an error if the same type is defined more than once in one translation unit.

Therefore, the definition of *StatusType* and of the constant E_OK have to be done as follows in all parts of OSEK:

```
#ifndef  STATUSTYPEDEFINED
#define STATUSTYPEDEFINED
typedef unsigned char StatusType;
#define E_OK 0
#endif
```

These definitions have to be done in the header files supplied by the OSEK suppliers.

Please note that, if *StatusType* is not set to ‘unsigned char’, there is no guarantee that implementations of different OSEK parts by different suppliers will be able to coexist.

3.3 Support of ‘*internal communication’*

The definition of messages for *internal*, *external* and *internal-external communication* must be consistent and guaranteed. To cope with the situation that both kernels, i.e. COM and OS, are linked within a system, rules are set up clarifying which kernel handles *internal communication*.

If both COM and OS kernels are present but one of CCCA or CCCB only is to be supported (no application message use *external communication*) then the OSEK OS kernel shall provide the functionality to handle internal messages, i.e. those using *internal communication*.

If both COM and OS kernels are present but one of CCC0 onwards is to be supported to handle *external communication* in addition to internal communication then the OSEK COM kernel shall provide the functionality to handle internal messages, i.e. those using *internal communication*.

Thus, it is guaranteed that definitions of data types used within internal and external message handling are consistent within a system.

To internally assure that the stated rules are followed, a define-variable LOCALMESSAGESONLY is defined. *Internal communication* within OSEK OS must be implemented if this define-variable is set.

The define-variable LOCALMESSAGESONLY shall be defined by the tool which generates a system out of an OIL file. As long as the definition of messages in OIL has not been completed, other means of definition may be used.

4 Glossary (INFORMATIVE)

1:1 Connection
logical communication channel between a transmitter and a receiver. A message is sent by exactly one transmitter and is received by exactly one receiver

1:N Connection
logical communication channel between a transmitter and N receivers. A message is sent by exactly one transmitter and is received by *N* receivers

Acceptance Filtering
mechanism which decides whether each received *protocol frame* is to be taken into account by the local *Node* or ignored

Activate
state transition of a *task* from *suspended* to *ready*. The transition is achieved by a system service

Actual Configuration
set of all operable nodes (see *operability of a node*) to which communication access is possible

Address-related Communication
special kind of communication between *nodes* using node addresses (see *node addressing*). Each address-related communication message contains certain data and - either explicitly or implicitly - the node address of the transmitter and the receiver. The communication of the *network management* is completely based on address-related communication

Alarm
alarm is an association between a counter and a task, event or callback. An alarm expires when a predefined counter value is reached. The expiry value can be defined relative to the current counter value or can be an absolute value. Alarms can be defined to be either single-shot or cyclic. An alarm is statically assigned at system generation time to: one counter and a task, event or alarm callback routine

Alarm callback
alarm callback routine is a short function provided by the application that gets called when the alarm expires but before any task is activated or event set

Alarm Management
alarm management is based on the *counter* concept. It lets the user link *alarm callbacks*, *task* activation or *event* setting to counter values. The link is done by use of *alarms*

Alive Message
dedicated *NM message*. An alive message is used to announce an initialised and operable node (see *operability of a node*) for integration in the *actual configuration*

API
Abbreviation of "Application Program Interface", the description of the application's interface to the operating system, communications and network management functions

Application errors
error where the operating system can not execute the requested service correctly, but assumes the correctness of its internal data. In this case, centralised error treatment is called

Arbitration
mechanism which guarantees that a simultaneous access made by multiple *stations* results in contention where one *frame* will survive uncorrupted

Basic Conformance Class
conformance Class of the OSEK operating system in which only *Basic Tasks* are admitted. Two basic conformance classes are distinguished: BCC1 and BCC2

Basic Task
task that has a defined beginning and a defined end. Basic tasks only release the processor if they are being terminated, the operating system is executing a higher-priority task or an *interrupt* occurs. A Basic Task can only enter the task states *suspended, ready* and *running*. It is not possible for a Basic Task to wait for an event

BCC
abbreviation of "*Basic Conformance Class*"

Broadcast
special case of multicast, whereby a single message is addressed to all nodes simultaneously

BNF
abbreviation of "*Backus-Naur Form*"

BT
abbreviation of "*Basic Task*"

BusOff
node is in the BusOff state when it is switched off from the bus. In the BusOff state a node can neither send nor receive any *protocol frames*

CAN
abbreviation of "Controller Area Network". A protocol originally defined for use as a communication network for control application in vehicles

CC
abbreviation of "*Conformance Class*"

CCC
abbreviation of "Communication Conformance Class"

Certification
purpose of certification is to determine whether an implementation is consistent with a given reference model. The scope of this reference model has to be settled according to the objectives of the *OSEK/VDX* project. All constraints necessary to fulfil these objectives must be incorporated in the reference model

COM
abbreviation of *"Communication"*

Communication Layer
set of all entities and elements which constitute a communication layer based on the *ISO/OSI Reference Model (ISO 7498)*

Configurability
ability to set the parameters of a system in terms of static values (e.g. number of tasks, RAM size for stack, size of message buffer, etc.)

Confirmation
service primitive defined in the *ISO/OSI Reference model (ISO 7498)*. With the 'confirmation' service primitive a service provider informs a service user about the result of a preceding service request of the service user

Conformance Class
in each module (operating system, communication, network management) a pool of services is provided, each being divided into a number of subsets. Applications can choose to use different

subsets of the services in order to reduce demands on the CPU and memory. These subsets are upwardly compatible and are described as conformance classes

Constructional Element
generic term for all definition and declaration services for system objects

Counter
counters are system objects that register recurring events, e.g. time, angle. A counter is represented by a count and some counter-specific constants

CPU
abbreviation of "Central Processing Unit"

Critical Section
sequence of instructions where *mutual exclusion* must be ensured. Such a section is called 'critical' because shared data is modified within it

Data Consistency
data consistency means that the content of a given message correlates unambiguously to the operation performed onto the message by the application. This means that no unforeseen sequence of operations may alter the content of a message hence rendering a message inconsistent with respect to its allowed and expected value

Data Link Layer
communication layer which provides services for the transfer of *I-PDUs*. The data link layer consists of the communication hardware and the communication driver software

Deadlock
state in which *tasks* block one another so that further processing of the tasks concerned is no longer possible. A deadlock between two tasks occurs, e.g. if both tasks wait for the reception of a message which is to be sent by the other task before sending its own message

Direct Node Monitoring
active monitoring of a *node* by another node in the network. For this purpose the monitored node sends a *NM message* according to a dedicated and uniform algorithm. For the network-wide synchronisation of NM messages a *logical ring* is used

Deadline Monitoring
in deadline monitoring the application is informed via the *notification mechanism* if: a message is not received from another node within a specified interval, or if a request to send an *I-PDU* is not completed by the *DLL* within a specified interval

DLL
abbreviation of "*Data Link Layer*"

ECC
abbreviation of "*Extended Conformance Class*"

ECU
abbreviation of "Electronic Control Unit" (see *station*)

EPROM
abbreviation of "Erasable Programmable Read Only Memory"

Error Handling
error service is provided to handle errors detected by the operating system. Its basic framework is predefined and has to be completed by the user. This gives the user a choice of efficient centralised or decentralised error handling

Error Hook
the error hook routine (*ErrorHook*) is called if a system service returns a StatusType value not equal to E_OK. *ErrorHook* is also called if an error is detected during task activation or event setting

ET
abbreviation of "*Extended Task*"

Event
events are a method of task synchronisation. *Extended tasks* may suspend their execution without terminating by waiting for events. The task continues when an appropriate event is set. *Basic tasks* may not use events

Event Mechanism
means of task synchronisation by using *events*

Extended Conformance Class
conformance Class of the OSEK operating system in which *Basic* and *Extended Tasks* are permitted. Two extended conformance classes are distinguished: ECC1, ECC2

Extended Task
extended tasks are distinguished from *Basic Tasks* by being allowed to use additional operating system services which may result in a *waiting* state. An Extended Task can enter the task states *suspended, ready, running,* and *waiting*

Fatal Error
error where the operating system can no longer assume correctness of its internal data. In this case the operating system calls the centralised system shutdown

FIFO
abbreviation of "First In First Out"

Frame
data unit according to the data link protocol specifying the arrangement and meaning of bits or bit fields in the sequence of transfer across the transfer medium (see data link message)

Full-preemptive Scheduling
full preemptive scheduling means that a task which is presently *running* may be rescheduled at any instruction by the occurrence of trigger conditions pre-set by the operating system. Full-preemptive scheduling will put the *running* task into the *ready* state as soon as a higher-priority task has become *ready*. The preemptee's context is saved so that it can be continued at the location where it was preempted2.54

Group Addressing
addressing of several receiver *nodes* in a single address-related *NM message* (see *address-related communication*). Group addressing is implemented by using *multicast* connections

the message objects are identified by a local (*Node*-wide) reference named handle. The handle is attached to both a logical and physical address

Hook Routine
a user defined function which will be called by the operating system under certain circumstances and in a defined context. Hook routines may be used for tracing or application dependent debugging purposes, user defined extensions to context switches, and in error handling. Most operating system services are not allowed in hook routines

IL
abbreviation for *Interaction Layer*

Indication
service primitive defined in the *ISO/OSI Reference Model (ISO 7498)*. With the service primitive 'indication' a service provider informs a service user about the occurrence of either an internal event or a service request issued by another service user

Indirect Node Monitoring
monitoring a *node* by "listening" to dedicated application communication messages. Indirect node monitoring is based on monitored *state messages* which are sent periodically

Interaction Layer
communication layer that implements the interface between the application and other potential communication layers (DLL, Network layers). The communication services of the interaction layer are independent of both microcontroller and network protocol. The interaction layer enables *internal* and network-wide communication by means of *UnQueued messages* and *Queued messages*

Internal Communication
exchange of messages between tasks belonging to the same node

Internal resource
internal resources are resources which are not visible to the user and therefore can not be addressed by the system functions *GetResource* and *ReleaseResource*. They are managed strictly internally within a clearly defined set of system functions

Interrupt
processor-specific event which can interrupt the execution of the current program section

Interrupt Latency
time between the moment an *interrupt* occurs and the execution of the first instruction of the *Interrupt Service Routine*

Interrupt Level
processing level provided for ISRs. To keep the *interrupt latency* brief, only absolutely indispensable actions should be performed at interrupt level

Interrupt Service Routine
function that provides the main processing of an *interrupt*

Intertask Communication
mode of information interchange between *tasks*. In the course of intertask communication, messages are logically copied from the local area of a task (transmitter) to the local area of another task (receiver)

I-PDU
collection of *messages* for transfer between nodes in a network. At the sending node the *IL* is responsible for packing *message* into an I-PDU and then sending it to the *DLL* for transmission. At the receiving node the DLL passes each I-PDU the IL which then unpacks the messages sending their contents to the application

ISO/OSI Reference Model
model to standardize interfaces and protocols for communication. ISO/OSI is the abbreviation of "International Organization for Standardization / Open Systems Interconnection" (ISO 7498)

ISR
abbreviation of "Interrupt Service Routine"

ISR Category
interrupt processing is subdivided into two categories of ISRs. ISR category 1 comprises all ISRs which do not use operating system services and are, therefore, typically faster for entry and exit than category 2 ISRs. Category 2 ISRs are allowed to use a restricted set of operating system services

Latency Time
time delay between the request of an activity and its execution

LIFO
abbreviation of "Last In First Out"

Limp Home
NM operating mode which is entered in case of an error which cannot be remedied

Limp Home Configuration
set of all *nodes* which cannot participate in *direct node monitoring* due to failure

Limp Home Message
dedicated *NM message* used for notifying a *node* that the system has entered the Limp Home state

Logical Ring
structure to order the *nodes* within a network. The nodes are arranged in terms of a ring. The logical ring is used for the networkwide synchronisation of *NM messages*. In a logical ring the communication sequence is defined independent of the network structure. Therefore each node is assigned a logical successor. The logically first node is the successor of the logically last node in the ring. A ring message always is sent from a node to its logical successor

Mixed-preemptive Scheduling
scheduling policy which enables the use of both scheduling policies, *full-preemptive* and *non-preemptive scheduling,* for the execution of different *tasks* on the same system. The distinction is made via a task attribute (preemptable / non-preemptable)

MSB
abbreviation of "Most Significant Bit"

Multiple Task Requesting
property of a *task* that allows it to have more than one activation outstanding (see *activate*). The operating system receives and records activations. On terminating the task (see *terminate*), the operating system checks whether any activations are outstanding. If so, the task immediately re-enters the running state

Mutual Exclusion
to modify shared data, a *task* must be able to get exclusive access for a limited time, i.e. all other tasks must be excluded to access this data. All tasks modifying shared data must be able to perform this exclusion. Therefore it is called mutual exclusion

Network Configuration
set of *nodes* in the network. Within the *NM* two configurations are distinguished: *actual configuration* and *limp home configuration*

Network Management
network management serves to ensure the safety and availability of the communications network of autonomous control units. OSEK-NM distinguishes between node-related (local) activities, e.g. initialisation of the node, and network-related (global) activities, e.g. coordination of global *NM operating modes*

NM
abbreviation of "*Network Management*"

NMBus-Sleep
NM operating mode. A *node* in NM Bus-Sleep mode does not participate in NM communication. This mode request must be confirmed by all nodes in the network

NM Infrastructure
All order structures (e.g. *logical ring*-) and addressing mechanisms (*Window*), which are accessed by the *network management*. This especially includes a communication infrastructure for the exchange of *NM messages*, so that each *node* is able to communicate with any other node on the network in a straightforward fashion

NMLimp Home
NM operating mode which is entered in case of an error which cannot be remedied

NM Message
NMPDU exchanged between NM entities. The NM distinguishes between *regular ring messages*, *alive messages* and *limp home messages*

NM Mode
see *NM operating mode*

NM Operating Mode
the *NM* can enter different local operating modes, e.g. NMoff, and global operating modes, e.g. sleep-mode. For each mode a specific behaviour of the NM is defined. The transition to global operating mode requires a network-wide coordination, i.e. the local NM of all nodes has to enter the same global mode. Local operating modes only affect the local NM of a node and are transparent for all the other nodes. Operating modes of the application are not managed by the NM

NMSleep Mode
NM operating mode. A node in NM Sleep mode does not participate in NM communication. The NM distinguishes between a local sleep mode and a global sleep mode. In both cases the transition into the sleep mode is notified network-wide. The difference is that a local sleep mode request must not be confirmed by the other nodes in the network. Whereas a global sleep mode request must be confirmed by all nodes in the network

NMPDU
abbreviation of NM Protocol Data Unit. A NMPDU represents an *NM message* communicated between the sending and receiving NM entities. The NMPDU contains an address field with source and destination address, a control field with an opcode and an optional data field with application specific ring data

Node
network topological entity. Nodes are connected by data links forming the network. Each node is separately addressable on the network

Node Addressing
each *node* has a unique identification, i.e. an address, which is known in the network. The addresses are used to transmit *NM messages* address-related from one node to another node. Individual node addressing is implemented using *1:1 connections*. Several nodes can be addressed using *group addressing*

Non-preemptive Scheduling
scheduling policy in which a *task* switch is only performed via one of a selection of explicitly defined system services (explicit *rescheduling points*)

Non-preemptable Task
task which can not be preempted by other tasks (see *preempt*). Such a task only releases the processor at *rescheduling points*

Offline
state of the *data link layer*. In the Offline state, no application communication is possible. Only the *network management* is allowed to communicate

OIL
abbreviation of "OSEK Implementation Language"

Online
(normal) state of the *data link layer*. Application *and network management* communication are possible

Operability of a Node
station is considered operable in terms of *NM*, if the node participates in *direct* or *indirect node monitoring*

OS
abbreviation of *Operating System*

OS Processing Level
processing level for the execution of services of the operating system. To enable optimum coordination between the processing options of various actions, the *OS* distinguishes three processing levels which are, by descending priority (high, medium, low): the *interrupt level*, the OS processing level and the *task level*

Overrun
attempting to store data in memory beyond its capacity, e.g. Queued message object

PostTaskHook
system hook routine called upon leaving a task either due to *pre-emption* by another task or by *termination*

Preempt
state transition of a *task* from *running* to *ready*. The scheduler decides to start another task. The *running* task is put into the *ready* state. In case of a non-preemptive scheduling policy, preemption only occurs at explicit *rescheduling points*

Preemptable Task
task which can be preempted by any task of higher priority (see *preempt*)

PreTaskHook
system hook routine called before entering or returning to a task

Priority Ceiling Protocol
mechanism used to prevent deadlocks and priority inversion in the framework of *resource management*

Protocol
formal set of conventions or rules governing the exchange of information between *protocol entities*. Protocol comprises syntax and semantics of the protocol messages as well as the instructions on how to react to them

Protocol Entity
task or a procedure for handling a *protocol*

Queued Message
Queued messages are contained in per-message FIFO buffers. Therefore the message at the head of the buffer is consumed by the receive operation

Ready
task state. All functional prerequisites for a transition into the *running* state exist, and the *task* only waits for allocation of the processor. The scheduler decides which *ready* task is executed next. The state is reached via the state transitions *Activate*, *Release* and *Preempt*, and is exited by *Start*

Re-entrant
function is " re-entrant" if the same function can be called again during an interruption of its execution, and both calls are executed correctly

Rescheduling Points
operating system calls which cause the activation of the *scheduler*. Rescheduling points exist not only in *full-preemptive* and *mixed preemptive systems*, but also in *non-preemptive systems*, e.g. explicit call of the scheduler, or successful termination of a task

Regular Ring Message
normal *NM message* containing the network status information. The regular ring message is also used to indicate a station *logoff* or local sleep mode or to request for global sleep mode (see *NM Sleep Mode*)

Release
state transition of a *task* from *waiting* to *ready*. At least one *event* has occurred which a task has waited on

Reply Message
dedicated *NM message* for replying to the reception of a *request message*. The reply message can be used by a slave of a *logical star*

Request
service primitive in compliance with the *ISO/OSI Reference Mode (ISO 7498)*. With the 'request' service primitive a service user requests a service from a service provider

Request Message
dedicated *NM message* for requesting the transmission of a *reply message*. The request message can be used by a master of a *logical star*

Resource
the OSEK operating system provides resources to support *task and ISR* coordination by *mutual exclusion* of *critical sections*. A task or ISR that locks a resource can not be preempted or interrupted by any other task or ISR that also might lock that resource. The assignment of resources to tasks and ISRs is performed at system generation time and cannot be changed by the application

Resource Management
resources are managed either implicitly (in the case of *internal resources*) or via a set of lock and unlock calls

Response
service primitive defined in the *ISO/OSI Reference Model (ISO 7498)*. The service primitive 'response' is used by a service user in order to reply to a preceding *indication* from service provider

Ring data
(see NMPDU) The application is able to send and receive specific data via the NM infrastructure. The data consistency of the data is guaranteed

Ring Message
normal NM message containing the network status information. The ring message is also used to indicate a node local sleep mode or to request for global sleep mode (see NM Sleep Mode)

Running
task state. In the *running* state, the *CPU* is assigned to the *task*, so that its instructions can be executed. Only one task can be in this state at any point in time. The state is entered by the state transition *Start* and can be exited via the state transitions *Wait*, *Preempt* or *Terminate*

Scalability
setting the scope of capabilities of a system as determined by its functionality (see *Conformance Class*)

Scheduler
the Scheduler decides whether a task switch should be made according to the selected *scheduling policy*. The Scheduler can be considered to occupy a resource which can also be occupied and

released by tasks. Thus a task can block the Scheduler to achieve arbitrary periods where it is the only task that can run

Scheduling Policy
the scheduling policy is used by the *scheduler* to determine whether a task may be preempted by another tasks or not. Three Scheduling policies are distinguished: *non-preemptive*, *full-preemptive* and *mixed-preemptive scheduling*

Segmented Communication
enables the transfer of application data *(see I-PDU)* which cannot fit into a single physical bus frame. Data has to be disassembled into *segments* that are small enough to fit into bus frames. These segments are then transferred separately and the message reassembled upon reception

Segmented Data Transfer
see *Segmented communication*

Semaphore
means for the synchronization of access to shared data. (see *resource management*)

Severe Error
error where the operating system could not achieve the requested service, but assumes the correctness of its internal data. In this case centralized error treatment is called. Additionally the operating system returns the error by the *status* information for decentralized *error handling*

Start
state transition of a *task* from *ready* to *running*. A *ready* task selected by the *scheduler* is executed

StartupHook
system hook routine called after the operating system start-up and before the scheduler is running

Suspended
task state. In the *suspended* state, the *task* is passive and does not occupy any dynamic *resource*. A task can be in this state on system start-up, or can reach it via the status transition *Terminate*. To exit the state, *Activate* the task

System Generation Services
definitions and directives which are necessary to set-up OSEK modules at compile time

ShutdownHook
system hook routine called when a system shutdown is requested by the application or by the operating system

Task
a task provides the framework for the execution of the application. A task can be executed concurrently with other tasks (see *Concurrency*). A task is executed under the control of the *Scheduler* according to the *task priority* assigned to it and the selected *scheduling policy*. A distinction is made between *Basic Tasks* and *Extended Task*

Task Level
processing level where most application software is executed, although some is also executed in *ISRs*. Tasks are executed according to the priority assigned to them and the selected *scheduling policy*. Other processing levels are: Interrupt level and *Operating System Level*

Task Management
this comprises the following tasks: Activation (see *activate)* and Termination (see *terminate*) of *tasks* as well as management of task states and task switches

Task Priority
the priority of a *task* is a measure for the precedence with which the task is to be executed. Initial priorities are defined statically. However, as the application runs, tasks may change their priority (see *Priority Ceiling Protocol*). Depending upon the *CC*, tasks of the same priority are admissible within a system. Tasks of equal priority are started according to the order in which they are acivated

Task States
the *tasks* of the OSEK operating system can assume the states *running*, *ready*, *waiting,* and *suspended*. *Basic Tasks* can not change to the state *waiting*. A task can only be in one state at any point in time

Task Switching Time
time between the occurrence of the "task switch event" up to the execution of the first instruction of the "new" *task*, i.e. including context switch

Task Switching Mechanism
mechanism, managed by the Scheduler, that performs a context switch to a selected Task

Terminate
state transition of a *task* from *running* to *suspended*. The *running* task causes its transition into the *suspended* state by a system service. A task can only terminate itself

Unacknowledged Communication
the transmitter receives no data from the receiver confirming that the message has been received

Unacknowledged Data Transfer
see *Unacknowledged Communication*

Unidirectional Communication
data transfer mode characterised by data being exchanged only in one direction

Unqueued Message
an unqueued message is overwritten upon arrival of a new message. The receive operation reads the last occurrence of an unqueued message. Therefore the message data can be read by the application more than once

Unsegmented Communication
the transfer of data that fits within a single bus frame

Unsegmented Data Transfer
see *Unsegmented communication*

UML
abbreviation of "*Unified Modeling Language*"

UUDT
abbreviation of "*Unacknowledged Unsegmented Data Transfer*"

Validation
ensuring the correctness of a specification

Wait
state transition of a *task* from *running* to *waiting*. The running task requires an *event* to continue operation. Event reception causes the task to make the transition into the *waiting* state

Waiting
task state. A *task* cannot be executed (any longer), because it has to wait for at least one *event*. The *waiting* state allows the processor to be freed and to be reassigned to a lower priority task without the need to terminate the *Extended Task*. Only Extended Tasks can assume this state. The state is reached by the status transition *Wait* and can be exited by *Release* of the task

Warning
corresponds to a return value, not equivalent to an error, giving complementary information related to a system service execution

5 History

Issue	Description	Date
1.0	Original issue 1.0 from candidate release 1 with no requirement change.	28th July 2000
1.1	Replacement of COM 2.2 with COM 2.2.1 in section 2.1.	11th September 2000
1.2	Replacement of OS 2.1 by OS 2.1r1 and COM 2.2.1 by COM 2.2.2 in section 2.1.	08th December 2000
1.3	Replacement of OS 2.1r1 by OS 2.2 and OIL 2.2 by OIL 2.3. Add OSEKtime/ORTI.	14th September 2001
1.4	Replaces: OIL 2.3 by OIL 2.4, COM 2.2.2 by COM 3.0. Provision for OS 2.2.1, COM 3.0.1, NM 2.5.2. Includes OSEK Overall Glossary.	6th September 2002