BRITISH STANDARD

BS ISO 17356-3:2005

Road vehicles — Open interface for embedded automotive applications —

Part 3: OSEK/VDX Operating System (OS)

ICS 43.040.15

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW

Document Sharing Hub
British Standards

Full document details: BS ISO 17356-3:2005

National foreword

This British Standard reproduces verbatim ISO 17356-3:2005 and implements it as the UK national standard.

The UK participation in its preparation was entrusted to Technical Committee AUE/16, Electrical and electronic equipment, which has the responsibility to:

— aid enquirers to understand the text;

— present to the responsible international/European committee any enquiries on the interpretation, or proposals for change, and keep UK interests informed;

— monitor related international and European developments and promulgate them in the UK.

A list of organizations represented on this committee can be obtained on request to its secretary.

Cross-references

The British Standards which implement international publications referred to in this document may be found in the *BSI Catalogue* under the section entitled "International Standards Correspondence Index", or by using the "Search" facility of the *BSI Electronic Catalogue* or of British Standards Online.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Summary of pages

This document comprises a front cover, an inside front cover, the ISO title page, pages ii to x, pages 1 to 61 and a back cover.

The BSI copyright notice displayed in this document indicates when the document was last issued.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 January 2006

© BSI 31 January 2006

ISBN 0 580 47286 8

Amendments issued since publication

Amd. No.	Date	Comments

Documents
Document Sharing Hub

Full document details: BS ISO 17356-3:2005

INTERNATIONAL STANDARD

ISO 17356-3

First edition
2005-11-01

Road vehicles — Open interface for embedded automotive applications —

Part 3:
OSEK/VDX Operating System (OS)

Véhicules routiers — Interface ouverte pour applications automobiles embarquées —

Partie 3: Système d'exploitation OSEK/VDX

Reference number
ISO 17356-3:2005(E)

Documents
Document Sharing Hub

Full document details: BS ISO 17356-3:2005

Documents
Document Sharing Hub

Full document details: BS ISO 17356-3:2005

Contents

Page

Foreword v
Introduction vi
1 Scope 1
2 Normative references 1
3 Architecture of the operating system "OS" 1
3.1 Processing levels 1
3.2 Conformance classes 3
3.3 Relationship between OS and OSEKtime OS 4
4 Task management 5
4.1 Task concept 5
4.2 Task state model 5
4.3 Activating a task 8
4.4 Task switching mechanism 8
4.5 Task priority 8
4.6 Scheduling policy 9
4.7 Termination of tasks 12
5 Application modes 12
5.1 General 12
5.2 Scope of application modes 12
5.3 Start-up performance 13
5.4 Support for application modes 13
6 Interrupt processing 13
6.1 General 13
7 Event mechanism 14
8 Resource management 16
8.1 General 16
8.2 Behaviour during access to occupied resources 16
8.3 Restrictions when using resources 17
8.4 Scheduler as a resource 17
8.5 General problems with synchronization mechanisms 17
8.6 Priority Ceiling Protocol 18
8.7 Priority Ceiling Protocol with extensions for interrupt levels 19
8.8 Internal resources 21
9 Alarms 22
9.1 General 22
9.2 Counters 22
9.3 Alarm management 22
9.4 Alarm-callback routines 23
10 Messages 24
11 Error handling, tracing and debugging 24
11.1 Hook routines 24
11.2 Error handling 25
11.3 System start-up 26
11.4 System shutdown 28
11.5 Debugging 28
12 Description of system services 29

Documents
Document Sharing Hub

Full document details: BS ISO 17356-3:2005

12.1 Definition of system objects 29
12.2 Conventions 29

13 Specification of OS services 31
13.1 Basics 31
13.2 Common data types 32
13.3 Task management 33
13.4 Interrupt handling 38
13.5 Resource management 41
13.6 Event control 43
13.7 Alarms 46
13.8 OS execution control 50
13.9 Hook routines 51

14 Implementation- and application-specific topics 54
14.1 General 54
14.2 Implementation hints 54
14.3 Application design hints 56
14.4 Implementation-specific tools 60

Documents
Document Sharing Hub
Full document details: BS ISO 17356-3:2005

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 17356-3 was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 3, *Electrical and electronic equipment*.

ISO 17356 consists of the following parts, under the general title *Road vehicles — Open interface for embedded automotive applications*:

— *Part 1: General structure and terms, definitions and abbreviations terms*

— *Part 2: OSEK/VDX specifications for binding OS, COM and NM*

— *Part 3: OSEK/VDX Operating System (OS)*

— *Part 4: OSEK/VDX Communication (COM)*

— *Part 5: OSEK/VDX Network Management (NM)*

— *Part 6: OSEK/VDX Implementation Language (OIL)*

AVA Documents
Document Sharing Hub

Full document details: BS ISO 17356-3:2005

Introduction

0.1 System philosophy

Automotive applications are characterized by stringent real-time requirements. Therefore, the operating system (OS) offers the necessary functionality to support event-driven control systems.

The specified OS services constitute a basis to enable the integration of software modules made by various manufacturers. To be able to react to the specific features of the individual control units as determined by their performance and the requirements of a minimum consumption of resources, the prime focus was not to achieve 100 % compatibility between the application modules, but their direct portability.

As the OS is intended for use in any type of control units, it supports time-critical applications on a wide range of hardware. A high degree of modularity and ability for flexible configuration are prerequisites to making the OS suitable for low-end microprocessors and complex control units alike. These requirements have been supported by definition of “conformance classes” (see 3.2) and a certain capability for application specific adaptations.

For time-critical applications, dynamic generation of system objects was left out. Instead, generation of system objects was assigned to the system generation phase. Error inquiries within the operating system are obviated to a large extent, so as not to affect the speed of the overall system unnecessarily. On the other hand, a system version with extended error inquiries has been defined. It is intended for the test phase and for less time-critical applications. Even at that stage, defined uniform system appearance is ensured.

0.1.1 Standardized interfaces

The interface between the application software and the OS is defined by system services. The interface is identical for all implementations of the OS on various processor families.

System services are specified in an ISO/ANSI-C-like syntax, however the implementation language of the system services is not specified.

0.1.2 Scalability

Different conformance classes, various scheduling mechanisms and the configuration features make the OS feasible for a broad spectrum of applications and hardware.

The OS is designed to require only a minimum of hardware resources (RAM, ROM, CPU time) and therefore runs even on 8-bit microcontrollers.

0.1.3 Error checking

The OS offers two levels of error checking, extended status for development phase and standard status for production phase.

The extended status allows for enhanced plausibility checks on calling OS services. Due to the additional error checking, it requires more execution time and memory space than the standard version. However, many errors can be found in a test phase. After all errors have been eliminated, the system can be recompiled with the standard version.

Documents
Document Sharing Hub
Full document details: BS ISO 17356-3:2005

0.1.4 Portability of application software

One of the goals of ISO 17356 is to support the portability and re-usability of application software. Therefore, the interface between the application software and the OS is defined by standardized system services with well-defined functionality. Use of standardized system services reduces the effort to maintain and to port application software and development cost.

Portability means the ability to transfer an application software module from one ECU to another without bigger changes inside the application. The standardized interface (service calls, type definitions and constants) to the operating system supports the portability on source code level. Exchange of object code is not addressed by ISO 17356.

The application software lies on the operating system and in parallel on an application-specific Input/Output System interface which is not standardized in ISO 17356. The application software module can have several interfaces. There are interfaces to the OS for real-time control and resource management, but also interfaces to other software modules to represent a complete functionality in a system, and at least to the hardware, if the application works directly with microcontroller modules.

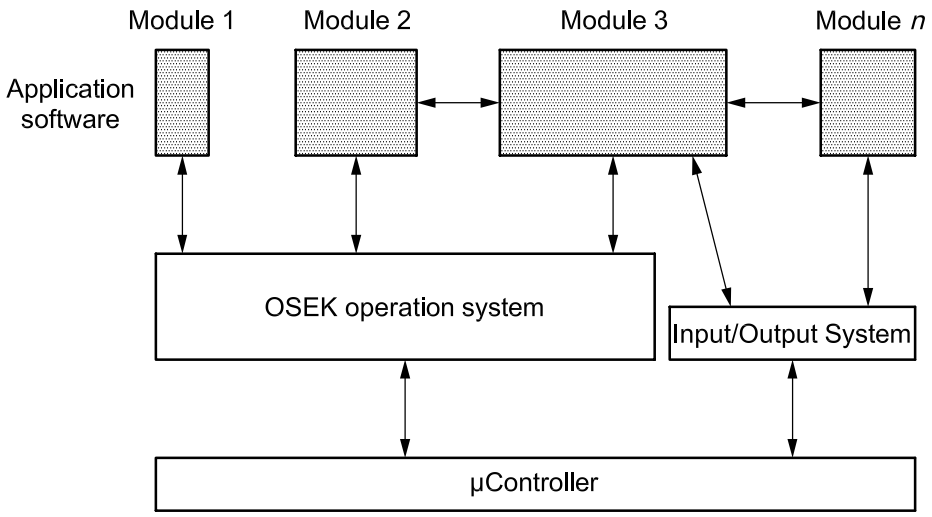

Figure 1 — Software interfaces inside ECU[1)]

During the process to port application software from one ECU to another, it is necessary to consider characteristics of the software development process, the development environment and the hardware architecture of the ECU, for example:

— software development guidelines;

— file management system;

— data allocation and stack usage of the compiler;

— memory architecture of the ECU;

— timing behaviour of the ECU;

— different microcontroller specific interfaces e.g. ports, A/D converter, serial communication and watchdog timer; and

— placement of the API calls.

1) OSEK OS allows direct interfacing between application and the hardware.

Documents
Document Sharing Hub

Full document details: BS ISO 17356-3:2005

This means that the specifications are not enough to describe an implementation completely. The implementation supplies specific documentation.

0.1.5 Support of portability

The certification process ensures the conformance of different implementations to the specification. Clause 14 of this International Standard collects implementation specific details which should be regarded to increase portability of an application between various implementations. Herein, only the OS interface to the application is considered.

0.1.6 Special support for automotive requirements

Specific requirements for the OS arise in the application context of software development for automotive control units. The following features address requirements such as reliability, real-time capability and cost sensitivity:

— The OS is configured and scaled statically. The user statically specifies the number of tasks, resources and services required.

— The specification of the OS supports implementations capable of running on ROM, i.e. the code could be executed from *Read-Only-Memory*.

— The OS supports portability of application tasks.

— The specification of the OS provides a predictable and documented behaviour to enable OS implementations, which meet automotive real-time requirements.

— The specification of the OS allows the implementation of predictable performance parameters.

0.2 Purpose of this document

The following description is to be regarded as a generic description which is mandatory for any implementation of the OS. This concerns the general description of strategy and functionality, the interface of the calls, the meaning and declaration of the parameters and the possible error codes.

This part of ISO 17356 leaves a certain amount of flexibility. On the one hand, the description is generic enough for future upgrades; on the other hand, part of the description is explicitly specified and implementation-specific.

Any implementation defines all implementation-specific issues. The conformance classes supported by the implementation are indicated precisely, and the issues identified as implementation-specific are documented.

Because this description is mandatory, definitions have only been made where the general system strategy is concerned. In all other respects, it is up to the system implementation to determine the optimal adaptation to a specific hardware type.

0.3 Structure of this document

0.3.1 General

In the following text, the clauses of this International Standard are described briefly:

0.3.2 Clause 3 — Architecture of the operating system “OS”

This clause gives a survey about the design principles and the architecture of the operating system.

Documents
Document Sharing Hub
Full document details: BS ISO 17356-3:2005

0.3.3 Clause 4 — Task management

This clause explains task management with the different task types and scheduling mechanisms.

0.3.4 Clause 5 — Application modes

This clause describes application modes and how they are supported.

0.3.5 Clause 6 — Interrupt processing

This clause provides information about the interrupt strategy and the different types of interrupt service routines.

0.3.6 Clause 7 — Event mechanism

This clause explains the event mechanism and the different behaviour depending on the scheduling.

0.3.7 Clause 8 — Resource management

This clause describes the resource management and discusses the benefits and implementation of the priority ceiling protocol.

0.3.8 Clause 9 — Alarms

This clause describes the two-stage concept to support time-based events (e.g. hardware-timer) as well as non-time-based events (e.g. angle measurement).

0.3.9 Clause 10 — Messages

The message handling for intra-processor communication is added to ISO 17356-3. Full message handling is described in ISO 17356-4. The exact subset to be implemented is described in ISO 17356-4.

0.3.10 Clause 11 — Error handling, tracing and debugging

This clause describes the mechanisms to achieve centralized error handling. It also describes the services to initialize and shut down the system.

0.3.11 Clause 12 — Description of system services

This clause describes the conventions used for description.

0.3.12 Clause 13 — Specification of operating system services

This clause describes all operating system services made available to the user. Structure of the description is identical for any service; it contains all the information the service user requires.

0.3.13 Clause 14 — Implementation and application-specific topics

This clause provides a list of all operating system-specific topics, including services, data types, and constants.

Documents
Document Sharing Hub

Full document details: BS ISO 17356-3:2005

0.4 Summary

0.4.1 General

The OS provides a pool of different services and processing mechanisms. It is built according to the user's configuration instructions at system generation time.

Four conformance classes are described, meant to satisfy different requirements concerning functionality and capability of the OS. Thus, the user can adapt the OS to the control task and the target hardware. It is not possible to modify the OS later at execution time.

Applications which have been written for a certain conformance class are portable to implementations of the same class. This is ensured by a definition of the services, their scope of capabilities, and the behaviour of each conformance class. Only if all the services of a conformance class are offered with the determined scope of capabilities does the OS implementation conform to ISO 17356-3. The service groups are structured in terms of functionality.

0.4.2 Task management

Task management includes:

— activation and termination of tasks; and

— management of task states and task switching.

0.4.3 Synchronization

The OS supports the following means of synchronization effective on tasks:

— resource management;

— access control for inseparable operations to jointly used (logic) resources or devices, or for control of a program flow;

— event control; and

— event management for task synchronization.

0.4.4 Interrupt management

This includes services for interrupt processing.

0.4.5 Alarms

Alarms can be either relative or absolute.

0.4.6 Intra-processor message handling

This includes services for exchange of data.

0.4.7 Error treatment

This includes mechanisms supporting the user in case of various errors.

Documents
Document Sharing Hub
Full document details: BS ISO 17356-3:2005

Road vehicles — Open interface for embedded automotive applications —

Part 3:
OSEK/VDX Operating System (OS)

1 Scope

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.

This part of ISO 17356 describes the concept of a real-time operating system, capable of multitasking, which can be used for motor vehicles. It is not a product description which relates to a specific implementation. It also specifies the operating system Application Program Interface (API).

General conventions, explanations of terms and abbreviations have been compiled in ISO 17356-1. ISO 17356-6 describes implementation and system generation aspects.

The specification of the OS represents a uniform environment which supports efficient utilization of resources for automotive control unit application software. The OS is a single processor operating system meant for distributed embedded control units.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 17356-1, *Road vehicles — Open interface for embedded automotive applications — Part 1: General structure and terms, definitions and abbreviations terms*

ISO 17356-2, *Road vehicles — Open interface for embedded automotive applications — Part 2: OSEK/VDX specifications for binding OS, COM and NM*

ISO 17356-6, *Road vehicles — Open interface for embedded electronic equipment — Part 6: OSEK/VDX Implementation Language (OIL)*

3 Architecture of the operating system “OS”

3.1 Processing levels

The OS serves as a basis for application programs which are independent of each other, and provides their environment on a processor. The OS enables a controlled real-time execution of several processes which appear to run in parallel.

Documents
Document Sharing Hub

Full document details: BS ISO 17356-3:2005

The OS provides a defined set of interfaces for the user. These interfaces are used by entities which are competing for the CPU. There are two types of entities:

— interrupt service routines managed by the operating system; and

— tasks (basic tasks and extended tasks).

The hardware resources of a control unit can be managed by OS services. These OS services are called by a unique interface, either by the application program or internally within the OS.

The OS defines three processing levels:

— interrupt level;

— logical level for scheduler; and

— task level.

Within the task level, tasks are scheduled (non, full or mixed preemptive scheduling) according to their user assigned priority. The run time context is occupied at the beginning of execution time and is released again once the task is finished.

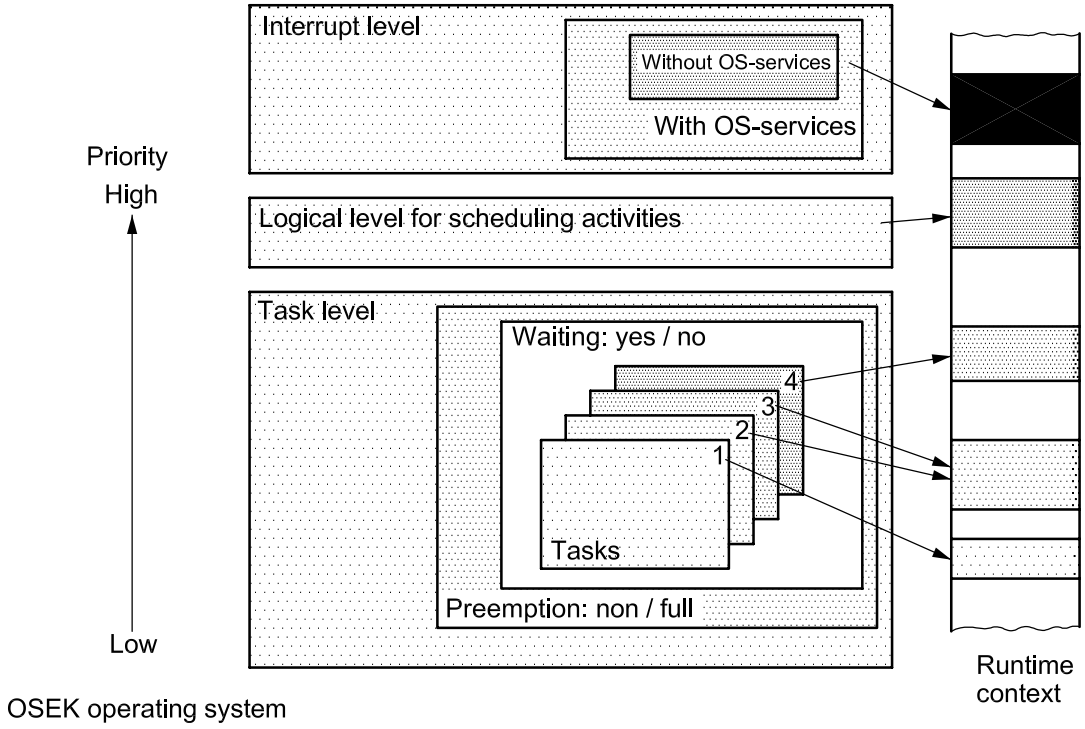

Figure 2 — Processing levels of the operating system

The following priority rules have been established:

— Interrupts have precedence over tasks.

— The interrupt processing level consists of one or more interrupt priority levels.

— Interrupt service routines have a statically assigned interrupt priority level.

Documents
Document Sharing Hub
Full document details: BS ISO 17356-3:2005

— Assignment of interrupt service routines to interrupt priority levels is dependent on implementation and hardware architecture.

— For task priorities and resource ceiling-priorities bigger numbers refer to higher priorities.

— The task's priority is statically assigned by the user. (The meaning of task priorities is described in 4.5.)

Processing levels are defined for the handling of tasks and interrupt routines as a range of consecutive values. Mapping of operating system priorities to hardware priorities is implementation-specific.

NOTE Assignment of a priority to the scheduler is only a logical concept which can be implemented without directly using priorities. Additionally, it does not prescribe any rules concerning the relation of task priorities and hardware interrupt levels of a specific microprocessor architecture.

3.2 Conformance classes

Various requirements of the application software for the system and various capabilities of a specific system (e.g. processor, memory) demand different features of the OS. In the following description, these OS features are described as "conformance classes" (CC).

Conformance classes exist to support the following objectives:

— to provide convenient groups of OS features for easier understanding and discussion of the OS;

— to allow partial implementations (which may be certified as compliant) along pre-defined lines; and

— to create an upgrade path from classes of lesser functionality to classes of higher functionality with no changes to the application using related features.

To be certified, the complete conformance class shall be implemented. However, system generation needs only to link those system services that are required for a specific application. It shall not be possible to change conformance classes during execution.

Conformance classes are determined by the following attributes:

— multiple requesting of task activation, as described in 4.3;

— task types, as described in 4.2.4; and

— number of tasks per priority.

All other features are mandatory if not explicitly stated otherwise.

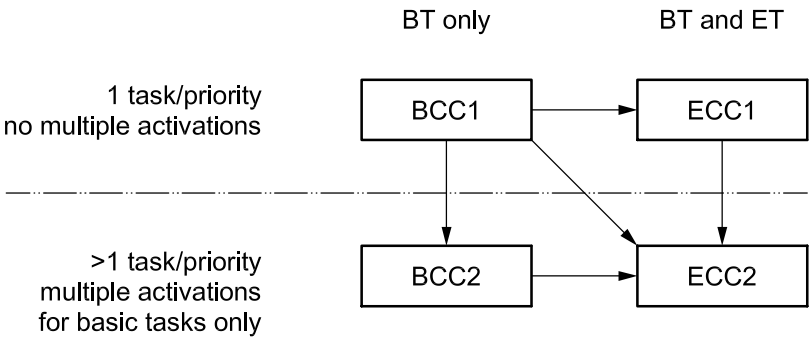

Figure 3 — Restricted upward compatibility for conformance classes

Documents
Document Sharing Hub
Full document details: BS ISO 17356-3:2005

The following conformance classes are defined:

— BCC1 (only basic tasks, limited to one activation request per task and one task per priority, while all tasks have different priorities);

— BCC2 (like BCC1, plus more than one task per priority possible and multiple requesting of task activation allowed);

— ECC1 (like BCC1, plus extended tasks); and

— ECC2 (like ECC1, plus more than one task per priority possible and multiple requesting of task activation allowed for basic tasks).

The portability of applications can only be assumed if the minimum requirements are not exceeded. The minimum requirements for Conformance Classes are shown in Table 1.

Table 1 — The minimum requirements for Conformance Classes

	BCC1	BCC2	ECC1	ECC2
Multiple requesting of task activation	no	yes	Basic task (BT): no Extended task (ET): no	BT: yes ET: no
Number of tasks which are not in the suspended state	8		16 (any combination of BT/ET)	
More than one task per priority	no	Yes	no (both BT/ET)	yes (both BT/ET)
Number of events per task	—		8	
Number of task priorities	8		16	
Resources	RES_SCHEDULER	8 (including RES_SCHEDULER)		
Internal resources	2			
Alarm	1			
Application Mode	1			

3.3 Relationship between OS and OSEKtime OS

OSEKtime OS (www.osek-vdx.org) is an OS especially tailored to the needs of time-triggered architectures. It allows ISO 17356-3 to coexist with OSEKtime OS. Conceptually, OSEKtime assigns its idle time to be used by ISO 17356-3. OS interrupts and tasks have less importance (lower priority) than similar entities in OSEKtime OS.

The OS interfaces, and the definition of system calls, do not change if the OS coexists with OSEKtime. There are minor exceptions with respect to system startup and shutdown due to the fact that OSEKtime is responsible for the overall system, whereas the OS is only locally responsible. These deviations are specifically mentioned within this part of ISO 17356.

On top of this, there is functionality defined within OSEKtime which imposes restrictions on the implementation of the OS if it is intended to coexist with OSEKtime OS. For more information, please refer to the specification of the OSEKtime OS.

Documents
Document Sharing Hub
Full document details: BS ISO 17356-3:2005

4 Task management

4.1 Task concept

4.1.1 General

Complex control software can conveniently be subdivided in parts executed according to their real-time requirements. These parts shall be implemented by means of tasks. A task provides the framework for the execution of functions. The OS provides concurrent and asynchronous execution of tasks. The scheduler organizes the sequence of task execution.

The OS provides a task switching mechanism (see scheduler, 4.4), including a mechanism which is active when no other system or application functionality is active. This mechanism is called idle-mechanism. Two different task concepts are provided by the OS:

— basic tasks; and

— extended tasks.

4.1.2 Basic tasks

Basic tasks only release the processor if:

— they terminate;

— the OS switches to a higher-priority task; or

— an interrupt occurs which causes the processor to switch to an interrupt service routine (ISR).

4.1.3 Extended tasks

Extended tasks are distinguished from basic tasks by being allowed to use the OS *WaitEvent*, which may result in a *waiting* state (see Clause 7 and 13.6.3.4). The *waiting* state allows the processor to be released and to be reassigned to a lower-priority task without the need to terminate the running extended task.

In view of the OS, management of extended tasks is in principal more complex than management of basic tasks, and requires more system resources.

4.2 Task state model

4.2.1 General

The following text describes the task states and the transitions between the states for both task types.

A task shall be able to change between several states, as the processor can only execute one instruction of a task at any time, while several tasks may be competing for the processor at the same time. The OS is responsible for saving and restoring task context in conjunction with task state transitions whenever necessary.

4.2.2 Extended tasks

Extended tasks have four task states:

— **Running**: 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, while all the other states can be adopted simultaneously by several tasks.

Documents
Document Sharing Hub

Full document details: BS ISO 17356-3:2005

— **Ready**: 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.

— **Waiting**: A task cannot continue execution because it shall *wait* for at least one event (see Clause 7).

— **Suspended**: In the *suspended* state, the task is passive and can be activated.

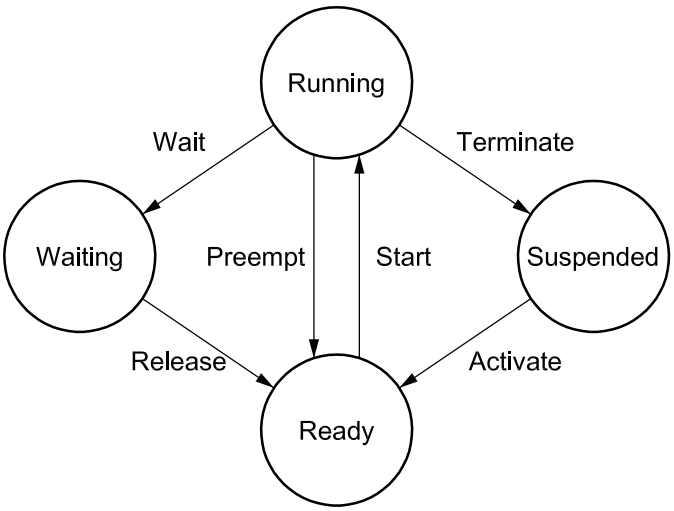

Figure 4 — Extended task state model

Table 2 — States and status transitions for extended tasks

Transition	Former state	New state	Description
activate	*suspended*	*ready*	A new task is set into the *ready* state by a system service. The operating system ensures that the execution of the task starts with the first instruction.
start	*ready*	*running*	A *ready* task selected by the scheduler is executed.
wait	*running*	*waiting*	The transition into the waiting state is caused by a system service. To be able to continue operation, the *waiting* task requires an event.
release	*waiting*	*ready*	At least one event has occurred which a task has *waited* for.
preempt	*running*	*ready*	The scheduler decides to start another task. The *running* task is put into the *ready* state.
terminate	*running*	*suspended*	The *running* task causes its transition into the *suspended* state by a system service.

Termination of a task is only possible if the task terminates itself ("self-termination"). This restriction reduces complexity of an OS. There is no provision for a direct transition from the *suspended* state into the *waiting* state. This transition is redundant and would add to the complexity of the scheduler.

Documents
Document Sharing Hub
Full document details: BS ISO 17356-3:2005

4.2.3 Basic tasks

The state model of basic tasks is nearly identical to the extended tasks state model. The only exception is that basic tasks do not have a *waiting* state. Basic tasks consist of the following task states:

— **Running**: 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, while all the other states can be adopted simultaneously by several tasks.

— **Ready**: 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.

— **Suspended**: In the *suspended* state, the task is passive and can be activated.

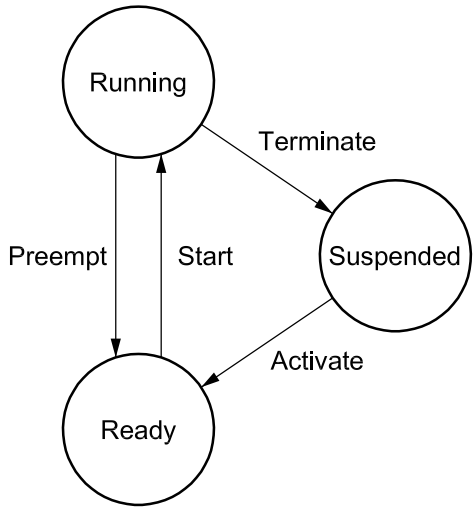

Figure 5 — Basic task state model

Table 3 — States and status transitions for basic tasks

Transition	Former state	New state	Description
activate	*suspended*	*ready*	A new task is set into the *ready* state by a system service. The operating system ensures that the execution of the task starts with the first instruction.
start	*ready*	*running*	A *ready* task selected by the scheduler is executed.
preempt	*running*	*ready*	The scheduler decides to start another task. The *running* task is put into the *ready* state.
terminate	*running*	*suspended*	The *running* task causes its transition into the *suspended* state by a system service.
NOTE Task activation will not immediately change the state of the task in case of multiple activation requests. If the task is not suspended, the activation will only be recorded and performed later.			

AVA Documents
Document Sharing Hub

Full document details: BS ISO 17356-3:2005

4.2.4 Comparison of the task types

Basic tasks have no *waiting* state, and thus only comprise synchronization points at the beginning and the end of the task. Application parts with internal synchronization points shall be implemented by more than one basic task. An advantage of basic tasks is their moderate requirement regarding run time context (RAM).

An advantage of extended tasks is that they can handle a coherent job in a single task, no matter which synchronization requests are active. Whenever current information for further processing is missing, the extended task switches over into the *waiting* state. It exits this state whenever corresponding events signal the receipt or the update of the desired data or events. Extended tasks also comprise more synchronization points than basic tasks.

4.3 Activating a task

4.3.1 General

Task activation is performed using the OS services *ActivateTask* or *ChainTask*. After activation, the task is ready to execute from the first statement.

The OS does not support C-like parameter passing when starting a task. Those parameters should be passed by message communication (see Clause 10) or by global variables.

4.3.2 Multiple requesting of task activation

Depending on the conformance class, a basic task can be activated once or multiple times. “Multiple requesting of task activation” means that the OS receives and records parallel activations of a basic task already activated.

The number of multiple requests in parallel is defined in a basic task specific attribute during system generation. If the maximum number of multiple requests has not been reached, the request is queued. The requests of basic task activations are queued per priority in activation order.

4.4 Task switching mechanism

Unlike conventional sequential programming, the principle of multitasking allows the OS to execute various tasks concurrently. Therefore, the scheduling policy has to be defined clearly (see 4.6).

The entity deciding which task shall be started and the triggering of all necessary OS internal activities is called the “scheduler”. The scheduler is activated whenever a task switch is possible according to the implemented scheduling policy. The scheduler can be considered as a resource which can be occupied and released by tasks. Thus, a task can reserve the scheduler to avoid a task switch until it is released (see 8.4).

4.5 Task priority

The scheduler decides on the basis of the task priority (precedence) which is the next of the *ready* tasks to be transferred into the *running* state.

The value 0 is defined as the lowest priority of a task. Accordingly bigger numbers define higher priorities.

To enhance efficiency, a dynamic priority management is not supported. Accordingly, the priority of a task is defined statically, i.e. the user shall not be able to change it at the time of execution. However, in particular cases the OS can treat a task with a defined higher priority (see 8.6).

Tasks of identical priority are supported in the conformance classes BCC2 and ECC2 (see 3.2).

Tasks on the same priority level are started depending on their order of activation, whereby extended tasks in the *waiting* state do not block the start of subsequent tasks of identical priority.

Documents
Document Sharing Hub
Full document details: BS ISO 17356-3:2005

A preempted task is considered to be the first (oldest) task in the *ready* list of its current priority.

A task being released from the *waiting* state is treated like the last (newest) task in the *ready* queue of its priority.

Figure 6 shows an example implementation of the scheduler using a queue for each priority level. Several tasks of different priorities are in the *ready* state; i.e. three tasks of priority 3, one of priority 2 and one of priority 1, plus two tasks of priority 0. The task which has waited the longest time, depending on its order of requesting, is shown at the bottom of each queue. The processor has just processed and terminated a task. The scheduler selects the next task to be processed (priority 3, first queue). Before priority 2 tasks can be processed, all tasks of higher priority shall have left the *running* and *ready* state, i.e. started and then removed from the queue either due to termination or due to transition into waiting state.

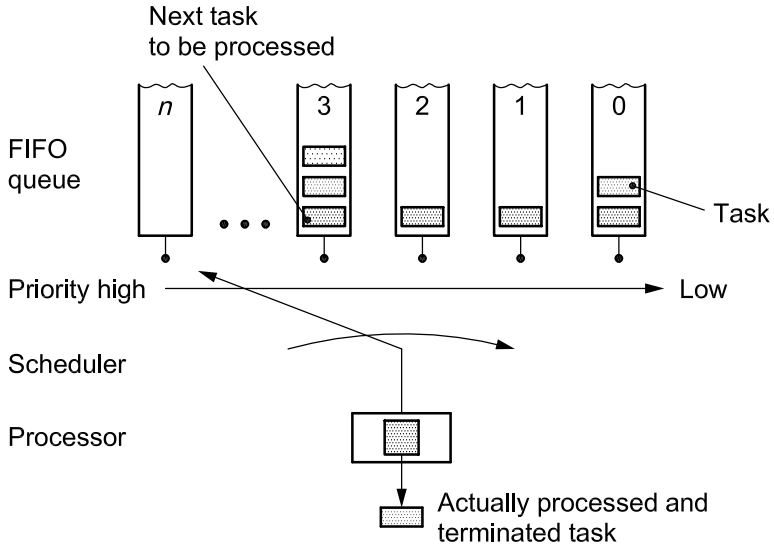

Figure 6 — Scheduler: order of events

The following fundamental steps are necessary to determine the next task to be processed:

— The scheduler searches for all tasks in the *ready/running* state.

— From the set of tasks in the *ready/running* state, the scheduler determines the set of tasks with the highest priority.

— Within the set of tasks in the *ready/running* state and of highest priority, the scheduler finds the oldest task.

4.6 Scheduling policy

4.6.1 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 OS. Full preemptive scheduling puts the *running* task into the *ready* state, as soon as a higher-priority task has gotten *ready*. The task context is saved so that the preempted task can be continued at the location where it was preempted.

With full preemptive scheduling, the latency time is independent of the run time of lower priority tasks. Certain restrictions are related to the increased (RAM) memory space required for saving the context, and the enhanced complexity of features necessary for synchronization between tasks. As each task can be theoretically rescheduled at any location, access to data which are used jointly with other tasks shall be synchronized.

AVA Documents
Document Sharing Hub

Full document details: BS ISO 17356-3:2005

In Figure 7, task T2 with the lower priority does not delay the scheduling of task T1 with higher priority.

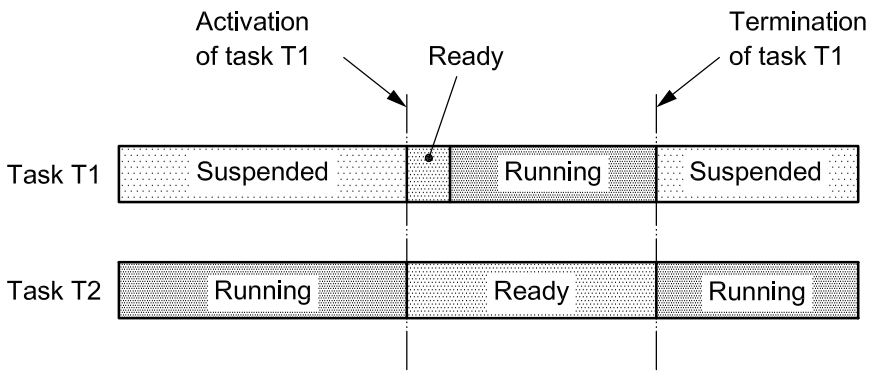

Figure 7 — Full preemptive scheduling

In the case of a full preemptive system, the user shall constantly expect preemption of the *running* task. If a task fragment shall not be preempted, this can be achieved by blocking the scheduler temporarily via the system service *GetResource*.

Summarized, rescheduling is performed in all of the following cases:

— successful termination of a task (system service *TerminateTask*, see 13.3.3.2);

— successful termination of a task with explicit activating of a successor task (system service *ChainTask*, see 13.3.3.3);

— activating a task at task level (e.g. system service *ActivateTask*, see 13.3.3.1, message notification mechanism, alarm expiration; if task activation is defined, see 9.3);

— explicit *wait* call if a transition into the *waiting* state takes place (extended tasks only, system service *WaitEvent*, see 13.6.3.4);

— setting an event to a *waiting* task at task level (e.g. system service *SetEvent*, see 13.6.3.1, message notification mechanism, alarm expiration; if event setting defined, see 9.3);

— release of resource at task level (system service *ReleaseResource*, see 13.5.3.2); and

— return from interrupt level to task level.

During interrupt service routines, no rescheduling is performed (see Figure 2).

Applications using the scheduling strategy “full preemptive scheduling” do not need the system service *Schedule*, but other scheduling policies make use of this system service. To enable portable applications to be written in spite of the different scheduling policies, the user can enforce a rescheduling via the system service *Schedule* at locations where he/she assumes a correct assignment of the CPU.

4.6.2 Non-preemptive scheduling

The scheduling policy is described as non-preemptive, if task switching is only performed via one of a selection of explicitly defined system services (explicit points of rescheduling).

Non-preemptive scheduling imposes particular constraints on the possible timing requirements of tasks. Specifically, the non-preemptable section of a *running* task with lower priority delays the start of a task with higher priority up to the next point of rescheduling.

Documents
Document Sharing Hub
Full document details: BS ISO 17356-3:2005

In Figure 8, task T2 with the lower priority delays task T1 with higher priority up to the next point of rescheduling (in this case, termination of task T2).

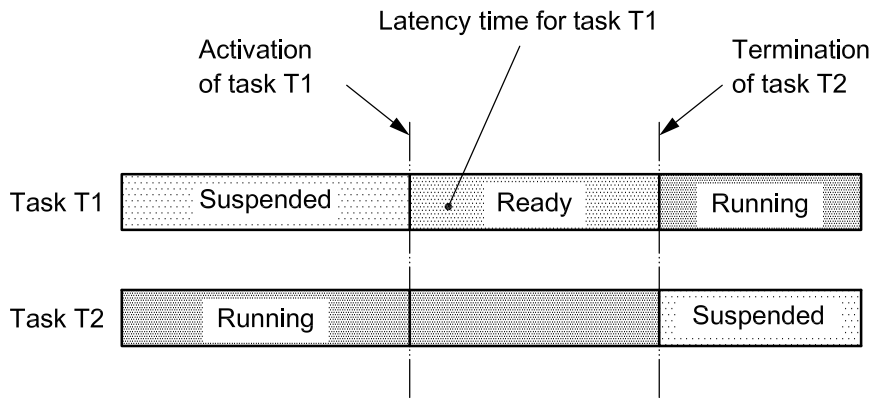

Figure 8 — Non-preemptive scheduling

4.6.3 Points of rescheduling

In the case of a non-preemptable task, rescheduling shall take place exactly in the following cases:

— Successful termination of a task (system service *TerminateTask*, see 13.3.3.2).

— Successful termination of a task with explicit activation of a successor task (system service *ChainTask*, see 13.3.3.3).

— Explicit call of scheduler (system service *Schedule*, see 13.3.3.4).

— A transition into the *waiting* state takes place (system service *WaitEvent*, see 13.6.3.4).

NOTE 1 The call of WaitEvent does not lead to a *waiting* state if one of the events passed in the event mask to WaitEvent is already set. In this case, WaitEvent does not lead to a rescheduling.

Implementations of non-preemptive systems may prescribe that OS services which cause rescheduling may only be called at the highest task program level (not in task subfunctions).

NOTE 2 A task switch at these points of scheduling usually requires saving of less task context information.

4.6.4 Groups of tasks

The OS allows tasks to combine aspects of preemptive and non-preemptive scheduling by defining groups of tasks. For tasks which have the same or lower priority as the highest priority within a group, the tasks within the group behave like non-preemptable tasks: rescheduling shall only take place at the points of rescheduling described in 4.6.2. For tasks with a higher priority than the highest priority within the group, tasks within the group behave like preemptable tasks (see 4.6.1).

Subclause 8.8 describes the mechanism of defining groups by using internal resources. Non-preemptable tasks are the most common usage of the concept of internal resources; they are tasks with a special internal resource of highest task priority assigned.

4.6.5 Mixed preemptive scheduling

If preemptable and non-preemptable tasks are mixed on the same system, the resulting policy is called “mixed preemptive” scheduling. In this case, scheduling policy depends on the preemption properties of the running

Documents
Document Sharing Hub

Full document details: BS ISO 17356-3:2005