



BSI Standards Publication

Road vehicles — Media Oriented Systems Transport (MOST)

Part 2: Application layer

National foreword

This British Standard is the UK implementation of ISO 21806-2:2020.

The UK participation in its preparation was entrusted to Technical Committee AUE/16, Data Communication (Road Vehicles).

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

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

© The British Standards Institution 2020
Published by BSI Standards Limited 2020

ISBN 978 0 539 03042 6

ICS 43.040.15

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 October 2020.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

**Road vehicles — Media Oriented
Systems Transport (MOST) —**

**Part 2:
Application layer**

*Véhicules routiers — Système de transport axé sur les médias —
Partie 2: Couche d'application*





COPYRIGHT PROTECTED DOCUMENT

© ISO 2020

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword	vi
Introduction	vii
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Symbols and abbreviated terms	3
4.1 Symbols	3
4.2 Abbreviated terms	3
5 Conventions	4
6 APP — Application	4
6.1 APP — Device model	4
6.2 APP — Node kinds	5
6.2.1 APP — MOST nodes	5
6.2.2 APP — Remote-controlled nodes	6
6.2.3 APP — Listen-only nodes	6
6.3 APP — Function block	6
6.3.1 APP — General	6
6.3.2 APP — FBlock library	7
6.3.3 APP — Controller and FBlock	7
6.4 APP — Functions	8
6.4.1 APP — Overview	8
6.4.2 APP — Methods	9
6.4.3 APP — Properties	11
6.5 APP — Events and notification	12
6.5.1 APP — Events	12
6.5.2 APP — Notification	12
6.6 APP — Notification mechanisms	13
6.6.1 APP — General	13
6.6.2 APP — Notification function	14
6.6.3 APP — Implicit notification	15
6.6.4 APP — Automatic notification	15
6.6.5 APP — Errors in the context of the notification function	15
6.6.6 APP — No valid values or property failure	15
6.6.7 APP — Reactions on Configuration.Status events	16
6.7 APP — Requesting FBlock information	16
6.7.1 APP — Function FBlockIDs	16
6.7.2 APP — Function FktIDs	17
6.7.3 APP — Extended FBlock identification	17
6.8 APP — Registry management	18
6.8.1 APP — General	18
6.8.2 APP — Central registry state	19
6.8.3 APP — NetworkMaster	22
6.8.4 APP — NetworkSlave	33
6.9 APP — Network wake-up, startup, and shutdown	40
6.9.1 APP — General	40
6.9.2 APP — Network wake-up and startup	41
6.9.3 APP — Network shutdown	41
6.9.4 APP — Device shutdown	43
6.10 APP — Connection management for streaming data	45
6.10.1 APP — Service for streaming data	45
6.10.2 APP — Source and sink information	46
6.10.3 APP — Streaming connections	48

6.10.4	APP — Compensating MOST network delay	54
6.11	APP — Diagnosis	54
6.11.1	APP — FBlock configuration status information	54
6.11.2	APP — Shutdown reason	56
6.11.3	APP — Shutdown reason analysis	56
6.11.4	APP — Ring break diagnosis	57
6.11.5	APP — Network diagnosis	57
6.12	APP — Error handling	58
6.12.1	APP — General	58
6.12.2	APP — Handling overload in a message receiver	58
6.12.3	APP — Over-temperature management	59
6.13	APP — Timing definitions	62
6.13.1	APP — Definitions	62
6.13.2	APP — NetworkMaster communication	62
6.13.3	APP — PowerMaster communication	66
7	AL — Application layer	68
7.1	AL — Structure of MOST messages	68
7.2	AL — Addressing	68
7.2.1	AL — Overview	68
7.2.2	AL — 16-bit addressing	69
7.2.3	AL — 48-bit addressing (Ethernet MAC address)	70
7.3	AL — Function block identifier (FBlockID)	70
7.4	AL — Instance identifier (InstID)	73
7.4.1	AL — Distinction of FBlock instances	73
7.4.2	AL — Uniqueness of functional addresses	73
7.4.3	AL — Assigning InstID	73
7.4.4	AL — InstID of NetBlock FBlock	73
7.4.5	AL — InstID of NetworkMaster FBlock	73
7.4.6	AL — InstID of FBlock EnhancedTestability	73
7.4.7	AL — Wildcard values for InstID	74
7.5	AL — Function identifier (FktID)	75
7.6	AL — Operation type (OPType)	76
7.6.1	AL — Overview	76
7.6.2	AL — Set, Get and SetGet	77
7.6.3	AL — Increment and Decrement	77
7.6.4	AL — Status	78
7.6.5	AL — Start and StartAck	78
7.6.6	AL — StartResult and StartResultAck	78
7.6.7	AL — Result and ResultAck	79
7.6.8	AL — Processing and ProcessingAck	79
7.6.9	AL — Abort and AbortAck	80
7.6.10	AL — Error and ErrorAck	80
7.6.11	AL — Parameters	86
7.7	AL — Timing definitions	87
7.7.1	AL — Definitions	87
7.7.2	AL — General communication	87
8	PL — Presentation layer	90
8.1	PL — Data and basic data types	90
8.1.1	PL — General	90
8.1.2	PL — Boolean	91
8.1.3	PL — Enum	91
8.1.4	PL — Numeric data types	92
8.1.5	PL — Length-coded String	97
8.1.6	PL — Array Type	98
8.1.7	PL — Record Type	99
8.1.8	PL — Stream	100
8.1.9	PL — BitField	103

8.1.10	PL — String.....	104
8.1.11	PL — Short Stream.....	106
8.1.12	PL — Classified Stream.....	106
8.2	PL — Function classes.....	107
8.2.1	PL — Purpose.....	107
8.2.2	PL — Properties with a single parameter.....	107
8.2.3	PL — Properties with multiple parameters.....	110
8.2.4	PL — Function classes for methods.....	134
9	Service interface definition to transport layer and network layer	135
	Bibliography	136

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

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. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 31, *Data communication*.

A list of all parts in the ISO 21806 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

The Media Oriented Systems Transport (MOST) communication technology was initially developed at the end of the 1990s in order to support complex audio applications in cars. The MOST Cooperation was founded in 1998 with the goal to develop and enable the technology for the automotive industry. Today, MOST¹⁾ enables the transport of high quality of service (QoS) audio and video together with packet data and real-time control to support modern automotive multimedia and similar applications. MOST is a function-oriented communication technology to network a variety of multimedia devices comprising one or more MOST nodes.

Figure 1 shows a MOST network example.

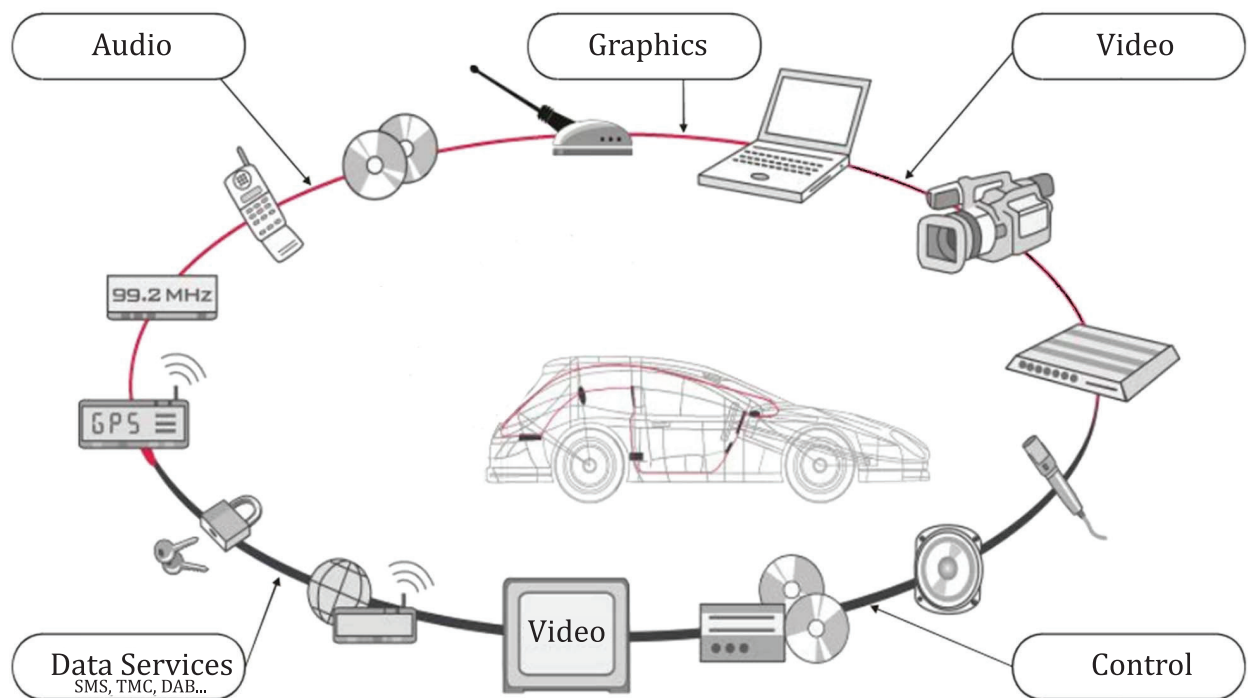


Figure 1 — MOST network example

The MOST communication technology provides:

- synchronous and isochronous streaming,
- small overhead for administrative communication control,
- a functional and hierarchical system model,
- API standardization through a function block (FBlock) framework,
- free partitioning of functionality to real devices,
- service discovery and notification, and
- flexibly scalable automotive-ready Ethernet communication according to ISO/IEC/IEEE 8802-3^[7].

MOST is a synchronous time-division-multiplexing (TDM) network that transports different data types on separate channels at low latency. MOST supports different bit rates and physical layers. The network clock is provided with a continuous data signal.

1) MOST® is the registered trademark of Microchip Technology Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO.

Within the synchronous base data signal, the content of multiple streaming connections and control data is transported. For streaming data connections, bandwidth is reserved to avoid interruptions, collisions, or delays in the transport of the data stream.

MOST specifies mechanisms for sending anisochronous, packet-based data in addition to control data and streaming data. The transmission of packet-based data is separated from the transmission of control data and streaming data. None of them interfere with each other.

A MOST network consists of devices that are connected to one common control channel and packet channel.

In summary, MOST is a network that has mechanisms to transport the various signals and data streams that occur in multimedia and infotainment systems.

The ISO standards maintenance portal (<https://standards.iso.org/iso/>) provides references to MOST specifications implemented in today's road vehicles because easy access via hyperlinks to these specifications is necessary. It references documents that are normative or informative for the MOST versions 4V0, 3V1, 3V0, and 2V5.

The ISO 21806 series has been established in order to specify requirements and recommendations for implementing the MOST communication technology into multimedia devices and to provide conformance test plans for implementing related test tools and test procedures.

To achieve this, the ISO 21806 series is based on the open systems interconnection (OSI) basic reference model in accordance with ISO/IEC 7498-1^[3] and ISO/IEC 10731^[5], which structures communication systems into seven layers as shown in [Figure 2](#). Stream transmission applications use a direct stream data interface (transparent) to the data link layer.

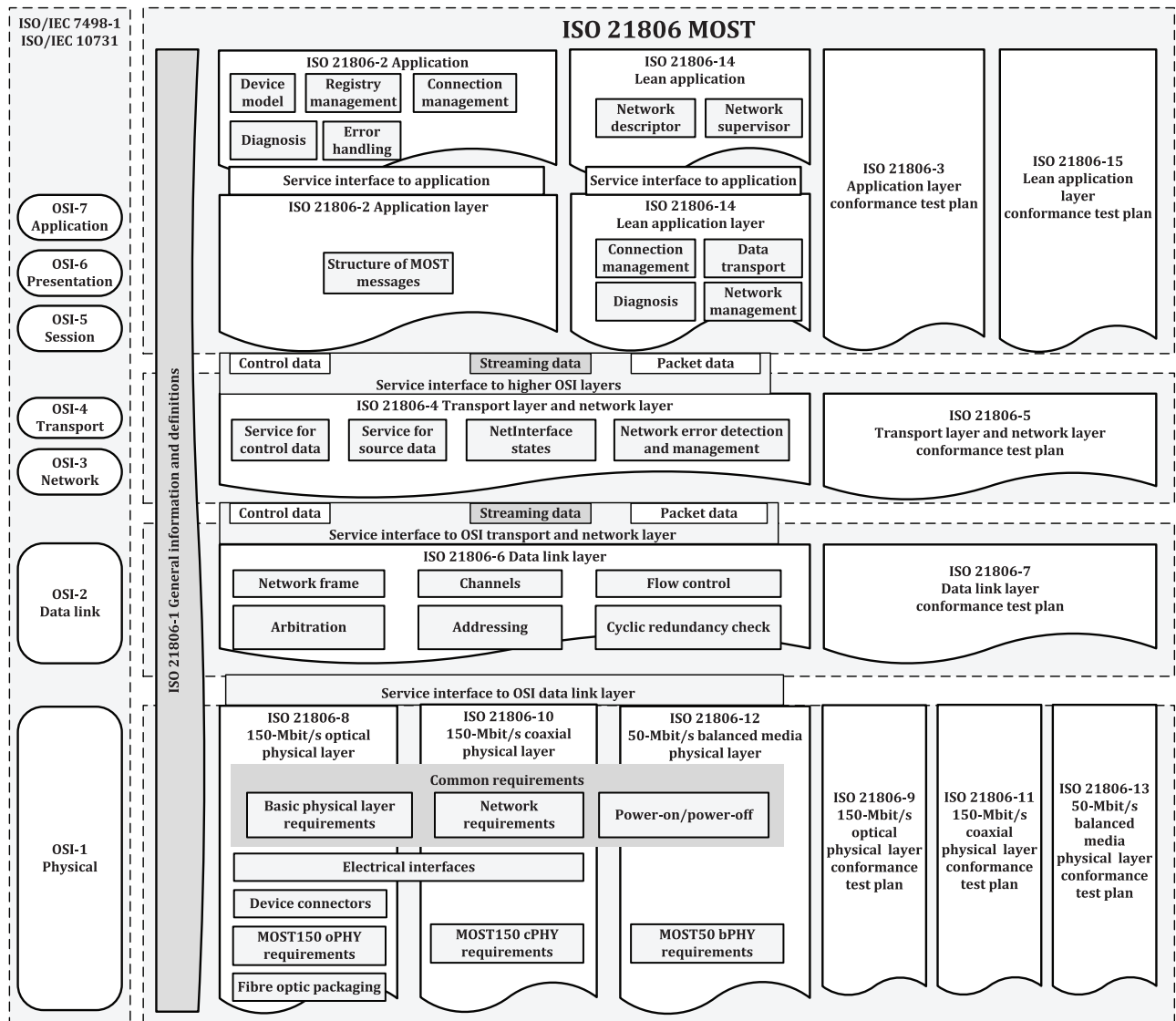


Figure 2 — The ISO 21806 series reference according to the OSI model

The International Organization for Standardization (ISO) draws attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO takes no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO. Information may be obtained from the patent database available at www.iso.org/patents.

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

Road vehicles — Media Oriented Systems Transport (MOST) —

Part 2: Application layer

1 Scope

This document specifies a part of the application, the application layer, and the presentation layer.

The application covers the

- device model,
- registry management,
- connection management for streaming data,
- diagnosis, and
- error handling.

The application layer covers the structure of MOST messages consisting of

- addressing,
- function block identifiers,
- instance identifiers,
- function identifiers,
- operation types, and
- timing definitions.

The presentation layer covers the definition of data, basic data types and function classes.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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 21806-1, *Road vehicles — Media Oriented Systems Transport (MOST) — Part 1: General information and definitions*

IEEE 754-2008, *IEEE Standard for Floating-Point Arithmetic*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 21806-1 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <http://www.electropedia.org/>

3.1 **administrative FBlock**

FBlock (3.10) that has an administrative purpose, which is specific to the MOST network

3.2 **central registry**

lookup table in the *NetworkMaster* (3.13) *FBlock* (3.10) for cross-referencing logical node addresses and functional addresses

3.3 **central registry state**

indicator for the availability of the *central registry* (3.2)

3.4 **connection manager**

entity that manages streaming connections

3.5 **ConnectionMaster**

FBlock (3.10) that provides the interface to the *connection manager* (3.4)

3.6 **controller**

client that uses the services of an *FBlock* (3.10) instance

3.7 **decentral registry**

lookup table in a *NetworkSlave* (3.14) for determining available *FBlocks* (3.10) and cross-referencing logical node addresses and functional addresses

3.8 **FBlock scan**

process of collecting information from the *NetworkSlaves* (3.14), performed by the *NetworkMaster* (3.13)

3.9 **function**

access to a feature of a service in an *FBlock* (3.10)

3.10 **function block**

FBlock
group of *functions* (3.9) that are particular to a specific application

3.11 **method**

function (3.9) that can be started and which leads to a result after a certain period of time

3.12 **MOST message**

message sent with functional address (FBlockID, InstID) as part of the payload

3.13 **NetworkMaster**

node that controls the *central registry state* (3.3) and administrates the *central registry* (3.2)

3.14**NetworkSlave**

node that reports its *FBlocks* (3.10) to the *NetworkMaster* (3.13)

3.15**notification**

mechanism for informing *controllers* (3.6) about changes in a *property* (3.19)

3.16**notification matrix**

table that contains the logical node addresses of *controllers* (3.6) that require status updates when a *property* (3.19) changes

3.17**PowerMaster**

node that controls MOST network startup and shutdown

3.18**PowerSlave**

node that provides *functions* (3.9) for MOST network startup and shutdown

3.19**property**

function (3.9) for reading or writing a value or status within an *FBlock* (3.10)

4 Symbols and abbreviated terms**4.1 Symbols**

--- empty cell/undefined

4.2 Abbreviated terms

AL	application layer
APP	application
CDC	compact disc changer
CRC	cyclic redundancy check
DAB	digital audio broadcasting
DiagID	diagnosis identifier
DSP	digital signal processor
DTCP	digital transmission content protection
DVD	digital video disc
ECL	electrical control line
ESDR	ETSI satellite digital radio
ETSI	European Telecommunications Standards Institute
FBlock	function block

FBlockID	function block identifier
FktID	function identifier
GSM	global system for mobile communications
HDCP	high-bandwidth digital content protection
HMI	human machine interface
InstID	instance identifier
JIS	Japanese Industrial Standard
LSb	least significant bit
MNC	MOST network controller
MOST	Media Oriented System Transport
MSb	most significant bit
NCE	network change event
OPType	operation type
QoS	Quality of Service
PL	presentation layer
RCC	remote control communication
RDS	radio data system
ROM	read-only memory
SMS	short message service
TMC	traffic message channel
TPEG	Transport Protocol Experts Group
TV	television
UTF	Unicode transformation format

5 Conventions

This document is based on OSI service conventions as specified in ISO/IEC 10731^[4].

6 APP — Application

6.1 APP — Device model

The following subclauses describe the logical model of a MOST device. A MOST device is a physical unit that can be connected to a MOST network via a MOST network controller (MNC).

A MOST device contains at least one MOST node, remote-controlled node, or listen-only node. A MOST device implements at least one MNC and MOST transceiver. A device that does not contain a MOST node or a remote-controlled node is beyond the scope of this document.

Between the FBlocks and the MNC, the MOST network service forms an intermediate layer providing routines to simplify the handling of the MNC. A MOST port is the MNC's connection point to the MOST transceiver.

Figure 3 shows a model of a MOST device with one MOST node and one MOST transceiver. In this example, the MOST node contains two application FBlocks, X and Y, and a controller for FBlock Z.

Application FBlocks are used to group functions that are particular to a specific application, for example radio or telephone. Administrative FBlocks group functions that have an administrative purpose, for example NetworkMaster or NetBlock.

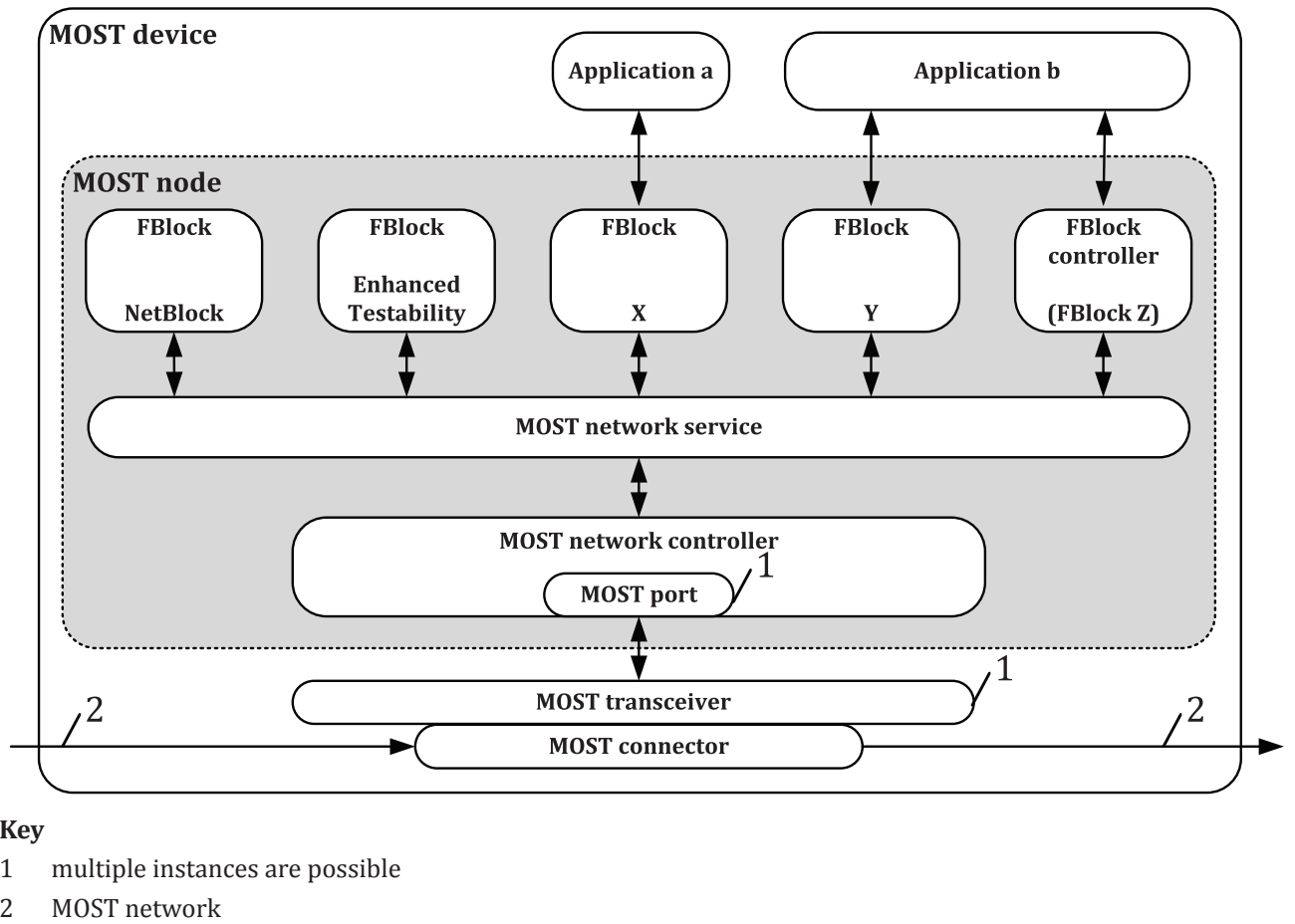


Figure 3 — Example of a MOST device

6.2 APP — Node kinds

6.2.1 APP — MOST nodes

The requirements in this subclause refer to distinctive characteristics of MOST nodes.

REQ	8.1 APP – MOST nodes – Structure
A MOST node shall implement the MOST network service and the MNC.	

REQ	8.2 APP – MOST nodes – Implement NetBlock
A MOST node shall implement the FBlock NetBlock.	

REQ	8.3 APP – MOST nodes – Implement EnhancedTestability
A MOST node shall implement the FBlock EnhancedTestability.	

REQ	8.4 APP – MOST nodes – One TimingMaster in the network
In a MOST network, there shall be exactly one designated TimingMaster.	

REQ	8.5 APP – MOST nodes – TimingMaster in MOST node
The designated TimingMaster shall reside in a MOST node.	

6.2.2 APP — Remote-controlled nodes

For certain use cases that do not require FBlock communication outside the administrative range (FBlockID 00₁₆ to 0F₁₆), a node kind exists which is called remote-controlled node.

REQ	8.6 APP – Remote-controlled nodes – Structure
A remote-controlled node shall implement the MNC and the FBlock NetBlock.	

REQ	8.7 APP – Remote-controlled nodes – No EnhancedTestability
A remote-controlled node shall not implement the FBlock EnhancedTestability.	
NOTE Because the FBlock EnhancedTestability is not implemented a remote-controlled node is treated differently during conformance testing.	

REQ	8.8 APP – Remote-controlled nodes – FBlocks in administrative range
A remote-controlled node shall not implement any FBlock outside the administrative range.	

REQ	8.9 APP – Remote-controlled nodes – Messages in administrative range
A remote-controlled node shall not send MOST messages to any FBlock outside the administrative range.	

6.2.3 APP — Listen-only nodes

For network analysis purposes, a certain node kind can be configured to operate in listen-only mode. Listen-only nodes do not change the content of network frames.

REQ	8.10 APP – Listen-only nodes – Behaviour
A listen-only node shall behave like a MOST node with its bypass closed.	
NOTE Listen-only nodes are typically not part of MOST networks; they are only included during system design and for failure diagnosis.	

6.3 APP — Function block

6.3.1 APP — General

On the application level, a MOST device contains multiple function blocks (FBlocks), for example, tuner, amplifier, or CD player.

Each FBlock contains a number of single functions. For example, a CD player possesses functions such as play, stop, eject, and time played.

6.3.2 APP — FBlock library

The MOST function catalogue (see ISO 21806-1) contains a set of standard FBlocks, which build the MOST FBlock library. The FBlock library can be used by network owners as a foundation for modelling the architecture of their MOST components and systems. Based on the FBlock library, a network owner creates the specific MOST function catalogue, which contains the set of FBlocks that is used by the network owner.

6.3.3 APP — Controller and FBlock

Application related communication over the MOST network occurs between a controller and an FBlock.

Controllers and FBlocks provide interfaces to applications. The controller sends commands to the FBlock. The FBlock executes these commands and returns reports. The controller processes the reports.

A node which contains controllers can also contain FBlocks and, therefore, be controlled by other nodes. An FBlock may be associated with more than one controller.

A controller interacts with one FBlock instance. Within a node, only one controller exists for one FBlock instance.

[Figure 4](#) illustrates application interaction on the controller and FBlock side, as well as MOST message transport.

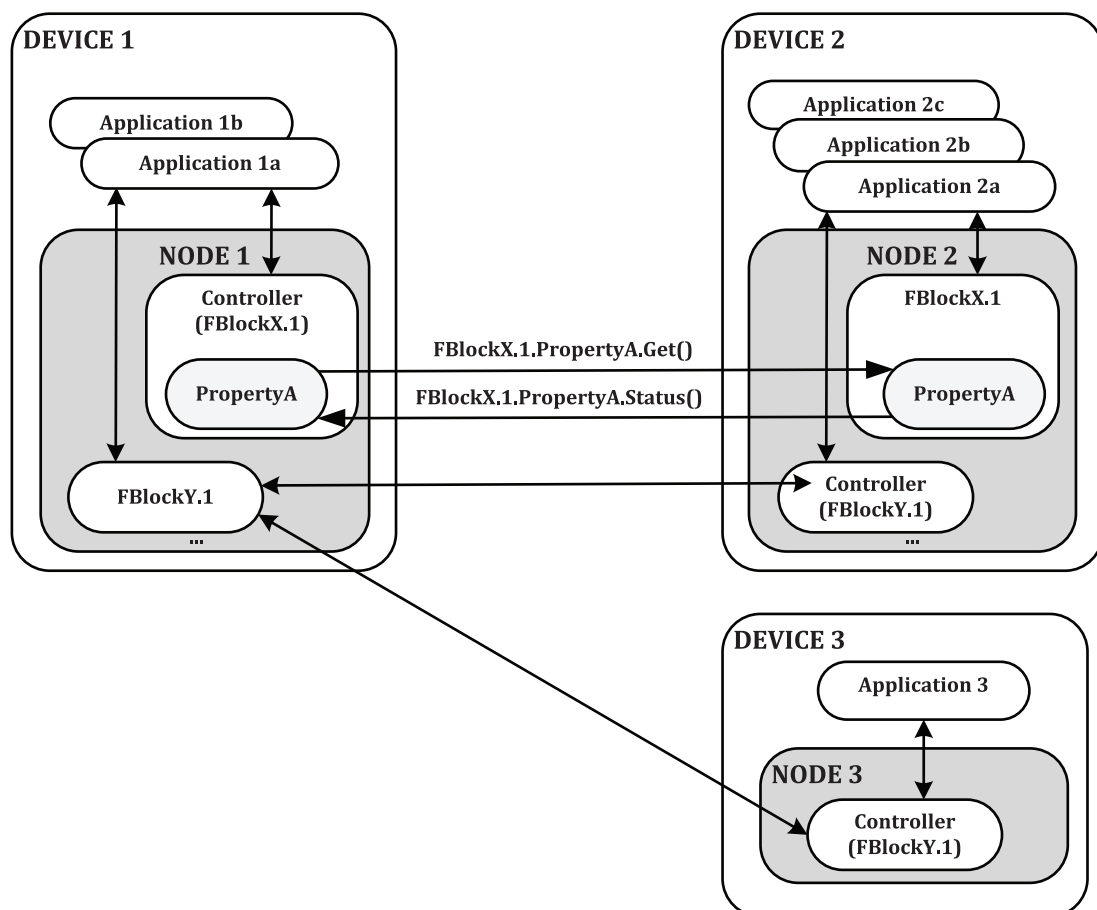


Figure 4 — Application, controller, and FBlock interaction example

6.4 APP — Functions

6.4.1 APP — Overview

A function is a specified attribute of an FBlock. Functions, as shown in [Figure 5](#), are subdivided into two categories:

- Functions that are started and do not lead to a result immediately. These functions are called methods.
- Functions for determining or changing the status of a node, which refer to the current properties of a node. These functions are called properties.

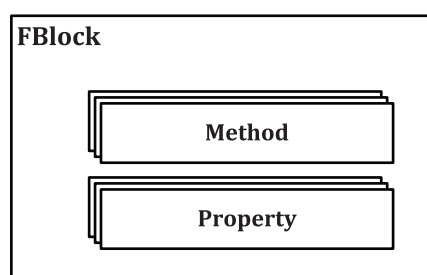


Figure 5 — Structure of an FBlock consisting of functions classifiable as methods and properties

To communicate with a function, a controller needs information about the available parameters, their limits, and the allowed operations.

On the application level, a function is addressed independently of the node it is implemented in. Functions are grouped into FBlocks.

Different FBlocks and functions of a node are distinguished by their identifiers:

`FBlockID.InstID.FktID`

- The `FBlockID` identifies the FBlock according to [Table 20](#) in [7.3](#).
- The `InstID` identifies the instance of an FBlock so that multiple instances of the same FBlock type can be addressed uniquely. One example is the NetBlock FBlock (`FBlockID 0116`), which is implemented by every MOST node.
- The `FktID` identifies the function.

Functions provide operations. The type of operation is specified by the `OPType`. The parameters of the operation follow the `OPType`, resulting in this structure:

`FBlockID.InstID.FktID.OPType(Parameter1, Parameter2,...)`

REQ 8.11 and 8.12 specify naming conventions for functions.

REQ	8.11 APP – Functions – Name without leading number
Function names shall not start with a number.	

REQ	8.12 APP – Functions – Permissible name characters
Function names shall contain no characters other than letters, numbers, and underscores.	

6.4.2 APP — Methods

6.4.2.1 APP — Execution of methods

In general, a method is started and does not immediately lead to a result. An example is the auto-scanning of a tuner.

In the device model, to each method there is an associated process which is executed when an FBlock receives a corresponding command for this method.

Methods can be specified without parameters. For example, a method to perform auto-scanning is designed without parameters or with a parameter that specifies the direction of the auto-scan.

Execution of a method fails, for example, if the addressed FBlock has no method of that kind, if a wrong parameter is found, or if the current status of the FBlock prevents execution. Methods may be designed to not be reentrant, that is, subsequent attempts to start a method are rejected as long as the method is processing the initial request.

REQ	8.13 APP – Methods – Reaction on execution failure
If execution of a method is not possible, the FBlock shall send the respective error message to the initiating controller.	

When finishing a process that is running due to `StartResult` or `StartResultAck`, the FBlock reports execution to the controller. This report may contain results of the process, for example, a frequency found by the tuner.

If a process runs for a long time, it may return intermediate results before finishing, such as informing the controller about the successful start of the process.

The controller may abort execution of a method with the `OPType Abort` or `AbortAck`.

For methods, [Table 1](#) in the column "Controller" shows the `OPTypes` that are used by the controller and in the column "FBlock" the `OPTypes` that are used by the FBlock.

Table 1 — OTypes for methods

Controller	FBlock
Start execution of a method (<code>Start</code> , <code>StartAck</code> , <code>StartResult</code> , <code>StartResultAck</code>)	Error with cause for error (<code>Error</code> , <code>ErrorAck</code>)
Abort a method (<code>Abort</code> , <code>AbortAck</code>)	Execution report with results (<code>Result</code> , <code>ResultAck</code>)
	Intermediate result (<code>Processing</code> , <code>ProcessingAck</code>)

6.4.2.2 APP — Using methods with `SenderHandle`

As several tasks within a node may access one method at the same time, a mechanism is provided to route the answer back to the respective task. This mechanism relies on the `SenderHandle` parameter.

REQ	8.14 APP – Methods – <code>SenderHandle</code> data type
The <code>SenderHandle</code> parameter shall be of data type <code>Unsigned Word</code> .	
NOTE 1 On the FBlock side, the <code>SenderHandle</code> in combination with the source address (of the originator) and <code>FBlockID.InstID</code> is used to identify a method under execution. On the controller side, the <code>SenderHandle</code> is used to dispatch the responses of the FBlock.	

REQ	8.15 APP – Methods – <code>SenderHandle</code> value
A controller shall choose a unique value for <code>SenderHandle</code> .	

REQ	8.16 APP – Methods – SenderHandle relevance
The <code>SenderHandle</code> shall not contain information beyond what is necessary to identify the corresponding method.	
NOTE 2 The <code>SenderHandle</code> is present when one of the <code>OPTypes</code> <code>StartResultAck</code> , <code>AbortAck</code> , <code>StartAck</code> , <code>ErrorAck</code> , <code>ProcessingAck</code> , and <code>ResultAck</code> is used, see 7.6.	

EXAMPLE 1

```
Controller → FBlock: FBlockID.InstID.StartResultAck(SenderHandle, Data)
FBlock → Controller: FBlockID.InstID.ResultAck(SenderHandle, Data)
FBlock → Controller: FBlockID.InstID.ErrorAck(SenderHandle, ErrorCode, ErrorInfo)
```

EXAMPLE 2

One example for the use of `SenderHandles` is the SMS service in a GSM module of a telephone device: In the HMI, three tasks are attempting to send SMS text messages independent of each other via the function `SMSSend`. The message of task 1 is successfully sent, the message of task 2 is buffered, while the message of task 3 is rejected.

Figure 6 illustrates the problem of dispatching an error message when no `SenderHandle` is present.

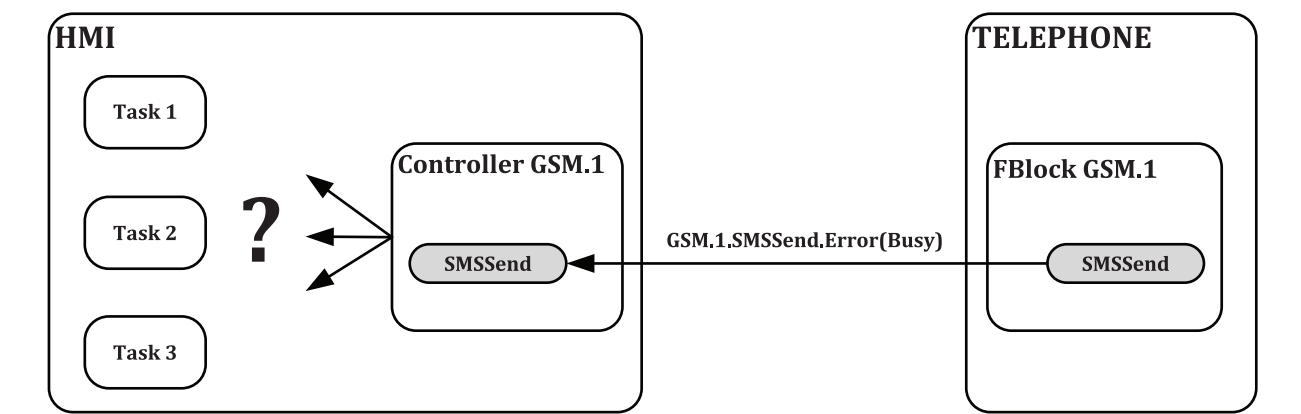


Figure 6 — Erroneous routing without the availability of a `SenderHandle`

Figure 7 illustrates how the error message is dispatched when the `SenderHandle` parameter is used.

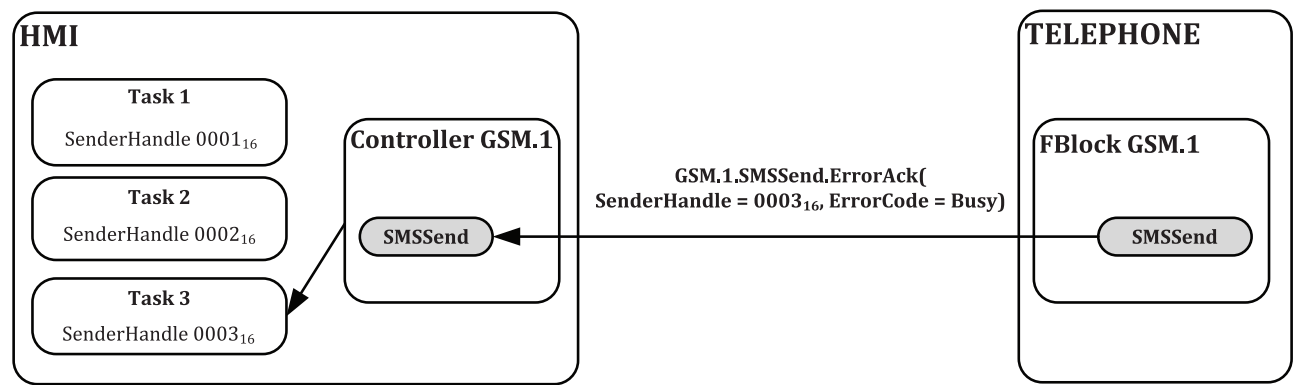


Figure 7 — Routing with the use of a `SenderHandle`

The `SenderHandle` is set by the HMI in `StartResultAck`. The `SenderHandle` is not interpreted by the telephone, but is returned in an answer (`ProcessingAck`, `ResultAck`, or `ErrorAck`).

The SMS call in task 1, for example, looks like:

```
HMI → Telephone: GSM.1.SMSSend.StartResultAck(SenderHandle = 000116, SMSData)
```

After successful transmission, task 1 receives:

```
Telephone → HMI: GSM.1.SMSSend.ResultAck(SenderHandle1)
```

Task 3 attempts to send while task 1 and 2 are active:

```
HMI → Telephone: GSM.1.SMSSend.StartResultAck(SenderHandle = 000316, SMSData)
```

It then receives this answer:

```
Telephone → HMI: GSM.1.SMSSend.ErrorAck(SenderHandle = 000316, ErrorCode = "Busy")
```

6.4.2.3 APP — Using methods without SenderHandle

Apart from not containing a `SenderHandle` parameter, methods without `SenderHandle` behave like methods with `SenderHandle`.

6.4.3 APP — Properties

6.4.3.1 APP — General

Within an FBlock, a property represents a value or a status.

Properties can be read (e.g. outside temperature), written (e.g. passwords), or both (e.g. desired value for speed control). For each property, the allowed operations are specified in the MOST FBlock library or the function catalogue of the network owner.

For changing and reading of properties, the OPTypes in [Table 2](#) are exchanged.

Table 2 — Messages for properties

Controller	FBlock
Writing a property (Set/SetGet)	Status of a property (Status)
Reading a property (Get)	Error message with cause of error (Error)
Incrementing / decrementing a property (Increment/Decrement)	

6.4.3.2 APP — Writing a property

The process of writing a property is illustrated in [Figure 8](#), which is an example of the temperature setting of a heating. In this scenario, a reasonable function interface chooses the data type Signed Byte for the Value parameter with a minimum value of -40 and a maximum value of 80.

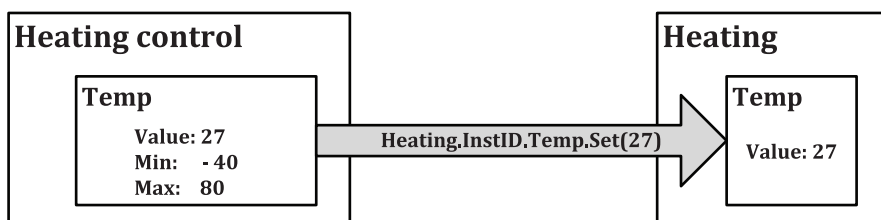


Figure 8 — Writing a property (temperature setting of a heating)

The `Temp` function is a member of the FBlock `Heating`, so the heating control sends the instruction `Heating.InstID.Temp.Set(27)` to FBlock `Heating`.

6.4.3.3 APP — Reading a property

In order for the HMI to display the current temperature, the value of function `Temp` in `FBlock Heating` is read.

Therefore, as shown in [Figure 9](#), the heating control sends the instruction `Heating.InstID.Temp.Get`.

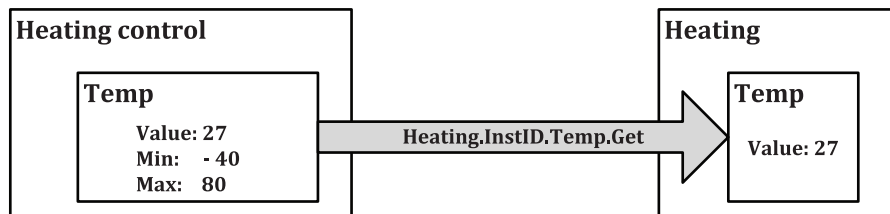


Figure 9 — Reading a property (temperature setting of a heating)

The heating replies, as shown in [Figure 10](#), by sending the status message `Heating.InstID.Temp.Status(27)`.

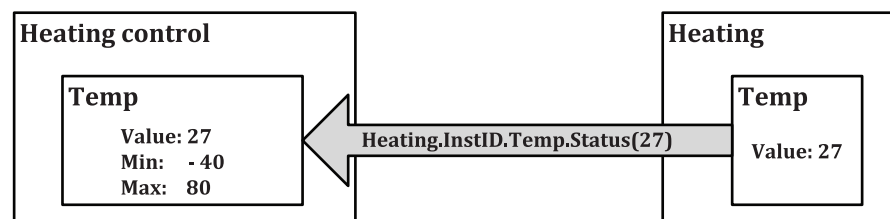


Figure 10 — Status report of a property (temperature setting of a heating)

6.5 APP — Events and notification

6.5.1 APP — Events

An event is the spontaneous transmission of a MOST report. The event is described as follows:

- After the request to report changes of a property, new values of the property are announced. This is called notification.
- MOST system event, which is based on node or central registry states that are reported for reasons of registry management or MOST network stability. Examples for such events are changes to the central registry or over-temperature of a device.

6.5.2 APP — Notification

Properties of an FBlock may change without a corresponding command, for example, when a CD player automatically moves to the next track after playing a track. To obtain current values of properties, repeated reading of the properties (polling) is not efficient.

To prevent polling and reduce communication with FBlocks, the notification mechanism is available. A controller may apply for notification of a particular property or a group of properties of an FBlock. The FBlock then sends status reports about changes in those properties without further requests.

Controllers registered for notification have a corresponding entry in the notification matrix of the FBlock (see [6.6](#)). All controllers that have corresponding entries for that particular property in the notification matrix receive a `Status` message with the updated value.

When using notification, it is not recommended to change properties with the `SetGet OPType`. The `Set OPType` should be used instead to avoid the potential transmission of multiple identical status messages.

6.6 APP — Notification mechanisms

6.6.1 APP — General

To avoid polling, notification events are specified to inform about changes of properties. Changes are announced by the `Status OPType`.

Such events are often sent to several controllers (e.g. two HMIs). Consequently, a notification matrix is implemented in every FBlock that supports notification. The controllers that are notified of changes to properties that the FBlock implements have corresponding entries in this matrix.

By design, methods do not support notification.

REQ	8.17 APP – Notification mechanisms – No entries for methods
A notification matrix shall not contain entries for methods.	

[Table 3](#) illustrates the structure of a notification matrix.

Table 3 — Notification matrix (x = notification activated)

Entry	FktID 1	FktID 2	FktID 3	FktID 4	FktID 5
TargetAddress1	x	x	x	x	x
TargetAddress2	---	x	---	x	---
Free for entry	---	---	---	---	---
Free for entry	---	---	---	---	---
Free for entry	---	---	---	---	---
Free for entry	---	---	---	---	---

The size of a notification matrix depends on the number of properties and the number of entries. When taking into consideration that a target address has 16 bits, an `FktID` has 12 bits, and that possibly entries for the maximum 64 nodes of the MOST network are contained, the notification matrix may be large.

The following subjects should be kept in mind:

- The notification matrix is only a model. It does not dictate the software implementation method.
- In most cases, it is sufficient if the notification matrix has only a few entries.
- Group addresses are allowed as target address in the notification matrix. For example, by using a group address, all HMIs in the MOST network can be notified of status changes.
- The debug address is not allowed as target address in the notification matrix.

Administration of the notification matrix is done via function `Notification` or by implicit notification.

REQ	8.18 APP – Notification mechanisms – FBlock discards irrelevant Status messages
A node shall discard <code>Status</code> messages if it does not implement the corresponding controller for the FBlock.	

REQ	8.19 APP – Notification mechanisms – Delete notification matrix entries for invalid targets
If an FBlock sends a status message based on a notification and the MNC indicates to the MOST network service that the transmission failed due to an invalid target address, all entries for this target address shall be deleted from the notification matrix of the FBlock sending this status message.	

This applies to notifications that were set by the `Notification` function as well as implicit notifications that were specified by the network owner.

6.6.2 APP — Notification function

To add or to remove an entry, a controller sends this command:

```
Controller → FBlock: FBlockID.InstID.Notification.Set(Control, TargetAddress, FktIDList)
```

To facilitate handling of potential errors, when using `Notification.Set`, not more than one `FktID` should be provided.

[Table 4](#) lists the values of the parameter `Control` of `Notification.Set`.

Table 4 — Parameter Control

Value	Name	Description
00 ₁₆	SetAll	Entry is done for all functions. The <code>FktIDList</code> is empty.
01 ₁₆	SetFunction	Entry is done for the following functions.
02 ₁₆	ClearAll	<code>TargetAddress</code> is deleted for all functions. The <code>FktIDList</code> is empty.
03 ₁₆	ClearFunction	<code>TargetAddress</code> is deleted for the specified functions.
04 ₁₆ to FF ₁₆	Reserved	Reserved by this document.

Starting with an empty notification matrix, the following commands result in a matrix as shown in [Table 3](#).

```
Notification.Set(SetAll, TargetAddress1)
Notification.Set(SetFunction, TargetAddress2, FktID4)
Notification.Set(SetFunction, TargetAddress2, FktID2)
```

Deleting entries is done in a similar way.

REQ	8.20 APP – Notification mechanisms – No error messages for inexistent entries
If the FBlock receives a command to delete an entry that does not exist, it shall not send an error message.	

To read information from the notification matrix, a controller sends:

```
Controller → FBlock: FBlockID.InstID.Notification.Get(FktID)
```

As an answer to this request, a list is returned that contains all target addresses with entries for the provided `FktID`:

```
FBlock → Controller: FBlockID.InstID.Notification.Status(FktID, <TargetAddress1, TargetAddress2,...,TargetAddressN>)
```

REQ	8.21 APP – Notification mechanisms – Initial report after Notification.Set
When an entry is added to the notification matrix, the FBlock shall send the report of the corresponding function to the target address that is contained in <code>Notification.Set</code> .	

REQ	8.22 APP – Notification mechanisms – Initial report within $t_{\text{NotificationProperty}}$
After receiving a <code>Notification.Set</code> message, the FBlock shall send the first report of any of the affected functions within $t_{\text{NotificationProperty}}$.	

REQ	8.23 APP – Notification mechanisms – Initial report for duplicate entry
If a node sends <code>Notification.Set</code> for a property that already has an entry in the notification matrix for this target address:	
— the report of the property shall be sent anyway, — the existing entry shall remain unmodified, and — no second entry shall be created for the requested target address.	

This also applies to group addresses.

6.6.3 APP — Implicit notification

The network owner or device/ECU supplier may specify implicit notifications when necessary.

The implicit notification is set within the node like a normal notification on transition from central registry state NotOK to OK. So, the target address that should be notified has an entry in the notification matrix. Therefore, on this transition, the initial event is sent over MOST. Updates are sent upon changes of the property. The (implicitly set) notification can be requested by `NotificationCheck` and may be removed by `FBlock.Notification.Set(Clear|ClearAll)` as with a notification set by function call. On transition from central registry state OK to NotOK, the implicit notification is cleared, as well.

REQ	8.24 APP – Notification mechanisms – Implicit notification limitations
	Implicit notification shall not be set for addresses that depend on dynamic address calculation (group addresses that do not depend on dynamic address calculation, blocking broadcast address, non-blocking broadcast address, static address, node position address of TimingMaster).

After transition from central registry state NotOK to OK, the controller receives the status reports of all functions which are activated as events.

6.6.4 APP — Automatic notification

Automatic notification is performed in conjunction with function class LongArray. Automatic notification is bound to the existence of a corresponding ArrayWindow function.

When using ArrayWindows, notification sets in when the controller creates an ArrayWindow.

Automatic notification does not necessarily affect the notification matrix.

6.6.5 APP — Errors in the context of the notification function

The `Notification` property reports one of the following error messages, if any error occurred:

- FBlock not registered in the notification service. This happens when the corresponding FBlock is not registered in the notification service

```
FBlock → Controller: FBlockID.InstID.Notification.Error(2016, 2016)
```

- No more entries available. If no additional entries can be made, function Notification answers:

```
FBlock → Controller: FBlockID.InstID.Notification.Error(2016, 2116)
```

- `Notification.Set` rejected. The corresponding properties (`FktIDList`) reject the `Notification.Set` command because of a notification matrix overflow or because the property is not registered in the notification service.

```
FBlock → Controller: FBlockID.InstID.Notification.Error(2016, 1016, FktIDList)
```

- `Notification.Get` not possible. On a received `Notification.Get` command, whenever the respective property is not registered in the notification service, the following is reported:

```
FBlock → Controller: FBlockID.InstID.Notification.Error(0716, 0116, FktID)
```

6.6.6 APP — No valid values or property failure

REQ	8.25 APP – Notification mechanisms – Temporarily unavailable property
	If a valid notification matrix entry is received at a time when the respective property is temporarily unavailable, the FBlock shall send the following message to the target address that corresponds to the entry: FBlock Controller: FBlockID.InstID.FktID.Error(41 ₁₆).

If a node responds with `ErrorCode 4116` to `Notification.Set` or `Notification.SetGet`, the node should insert the target address from the `Notification` message into the notification matrix.

This `ErrorCode` can be used to indicate the failure of a sensor. If the sensor signal disappears, the sensor function reports the error “not available” (`4116`). If the function has an implemented notification mechanism, the error is distributed to the controllers that have corresponding entries in the notification matrix.

6.6.7 APP — Reactions on `Configuration.Status` events

- `Configuration.Status(NotOK)`
After reception of `NotOK` (and every time the central registry is regarded in central registry state `NotOK`, for example, after startup) the notification matrix is deleted.
- `Configuration.Status(NewExt)`
After receiving `Configuration.Status(NewExt)`, nodes check whether it is necessary to add entries in the notification matrices of the new `FBlocks`.

Notification matrix entries are requested with `Notification.Set`. Only those which are related to the new `FBlocks` should be added. It is not necessary to rebuild other notifications.
- `Configuration.Status(Invalid)`
After receiving `Configuration.Status(Invalid)`, nodes check whether they are registered for notification with the disappeared `FBlocks`. If that is the case, the affected nodes should take the appropriate actions.

6.7 APP — Requesting `FBlock` information

6.7.1 APP — Function `FBlockIDs`

The property `FBlockIDs` is used to obtain information about the `FBlocks` contained in a node, shown in [Figure 11](#). This command reads `FBlockIDs`:

```
NetworkMaster → NetworkSlave: NetBlock.FBlockIDs.Get
```

The `NetworkSlave` answers with a list of the contained `FBlockIDs`.

The `FBlocks` `NetBlock`, `EnhancedTestability`, and `DebugMessages` are not listed:

```
Node → Controller: NetBlock.FBlockIDs.Status(FBlockID 1, InstID 1,
FBlockID 2, InstID 1,... FBlockID N, InstID 1)
```

The term “controller” is used because the `NetworkSlave` cannot verify in all cases that the sender is the `NetworkMaster`.

[Figure 11](#) shows the information contained in `FBlockIDs`.

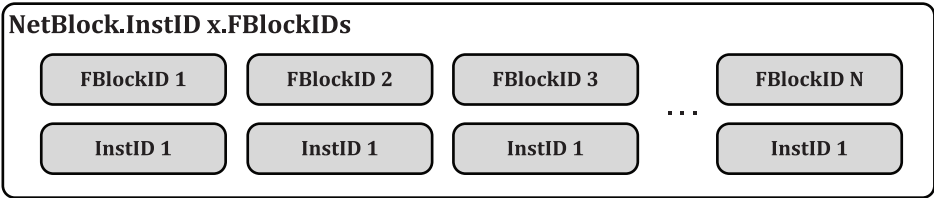


Figure 11 — Reading the `FBlocks` of a node from `NetBlock FBlock`

REQ	8.26 APP – Requesting <code>FBlock</code> information – <code>NetBlock</code> implements <code>FBlockIDs</code>
Every <code>NetBlock FBlock</code> shall implement the property <code>FBlockIDs (000₁₆)</code> .	

6.7.2 APP — Function FktIDs

In an adaptable system, it may happen that a controller does not know which functions are available in an FBlock (e.g. simple or high-end audio amplifier).

Therefore, every application FBlock that is listed in the central registry has the function `FktIDs` (`00016`).

It is read as follows:

```
Controller → Node: FBlockID.InstID.FktIDs.Get
```

Within an FBlock, `FktIDs` between `00016` and `FFF16` (4 096 different `FktIDs`) can be available. The `FktIDs` are assigned as described in [Clause 7](#). This raises the problem of a compact response, if the functions contained in an FBlock are requested. It is solved by a mechanism derived from the run length encoding (RLE). A bit field is built where the first bit is set to 1 if `FktID 00016` is available; the second bit is set to 1 if `FktID 00116` is available, and so on. [Figure 12](#) shows an example of such a bit field.

FktID	---	000 ₁₆	001 ₁₆	002 ₁₆	003 ₁₆	004 ₁₆	005 ₁₆	006 ₁₆	...	021 ₁₆	022 ₁₆	023 ₁₆	024 ₁₆	...	A00 ₁₆	A01 ₁₆	A02 ₁₆	A03 ₁₆	...	FFF ₁₆
Bit field	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1	0	0	0	0

Figure 12 — Run length encoding example

The answer lists only the positions (`FktIDs`) where the bit state changes, beginning with an initial bit state of 1.

For the example shown above, the result is:

```
Node → Controller: FBlockID.InstID.FktIDs.Status(<00216 00416 00616 02216 02416 A0016 A0216 016>)
```

The trailing `016` represents a stuffing nibble.

[Figure 13](#) shows how the information from `FBlockIDs` is used to request `FktIDs` from an FBlock.

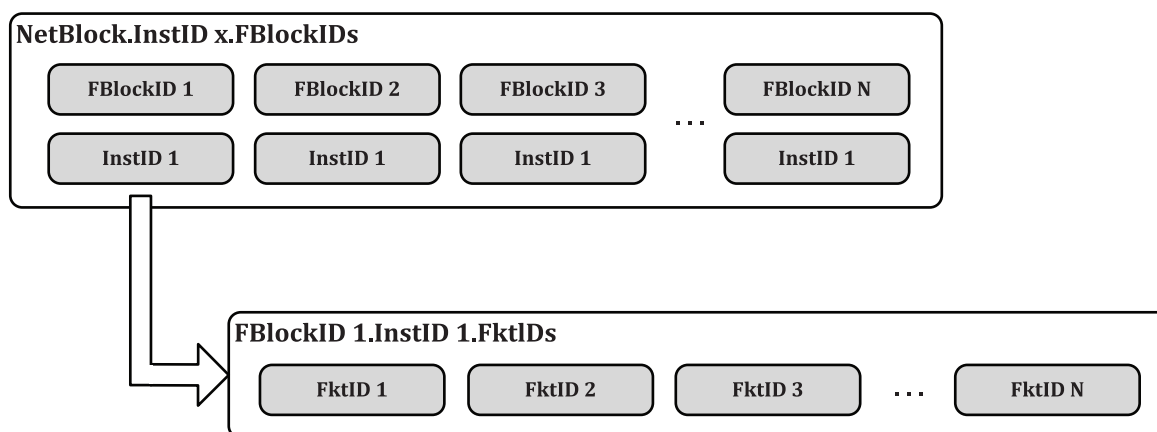


Figure 13 — Requesting the functions contained in an application FBlock

6.7.3 APP — Extended FBlock identification

The property `FBlockInfo` is used to distinguish between the functionality of similar FBlocks with the same `FBlockID`.

The function `FBlockInfo` provides information about the FBlock name, the name of the instance, the corresponding MOST version, and the version of the FBlock itself. This information is provided in text form.

The name of the instance is a string, which is assigned by the network owner to allow further differentiation of the application, for example, `DISP_L` for a left display or `DISP_R` for a right display.

Additionally, `FBlockInfo` can provide maturity information for each function that is implemented by the FBlock. For those FBlocks that do not appear in the central registry, maturity status is usually reported as `Unknown`.

REQ	8.27 APP – Requesting FBlock information – FBlockInfo
	Every FBlock, except administrative FBlocks and device/ECU supplier specific FBlocks, shall implement the property <code>FBlockInfo (FktID 011₁₆)</code> .

6.8 APP — Registry management

6.8.1 APP — General

6.8.1.1 APP — Purpose

Registry management is the process by which the NetworkMaster enables secure communication between applications over the MOST network.

This subclause describes how the NetworkMaster and the NetworkSlaves enable safe registry management. The tools used for this process include the control of the central registry state and the administration of the central registry (see [6.8.3.4](#)), as well as the decentral registries (see [6.8.4.2](#)).

[6.8.2](#) contains the detailed description of central registry states.

The central registry represents the FBlock configuration in the MOST network. When an application has insufficient information to address a communication partner, that is, an FBlock, it requests that information from the NetworkMaster's central registry.

In NetInterface state `s_NetInterface_Normal_Operation`, a MOST network is in central registry state OK or NotOK. The central registry state reflects the validity of the central registry. In central registry state OK, the central registry is available; in central registry state NotOK, the central registry is being initialised or updated. The NetworkMaster builds and maintains the central registry, as well as distributes the central registry state to all NetworkSlaves.

The NetworkMaster builds the central registry by collecting logical node addresses and FBlock configuration from all NetworkSlaves. The MOST network relies on a valid central registry, not only because it contains the information used to find communication partners, but also because it is crucial that a node is informed if one of its communication partners disappears.

A node that contains controllers may store the information concerning its communication partners in a decentral registry. The benefit of having a decentral registry is that the NetworkSlave does not need to request the logical node address of its communication partners before sending a message.

Detailed requirements of the behaviour of MOST devices regarding registry management are described in [6.8.2](#) and [6.8.4](#).

6.8.1.2 APP — Initialisation

6.8.1.2.1 APP — Configuration

This subclause describes the startup following the `ev_Init_Ready` event.

Examples for initialising a higher layer due to the `ev_Init_Ready` event:

- Check of FBlock configuration and building of the central registry.
- Setting of the logical node address and group address.

— Initialisation of the sending and receiving parts of the MOST network service.

6.8.1.2.2 APP — Initialisation of the central registry

The NetworkMaster is responsible for initialising the central registry after `ev_Init_Ready` event. It collects the FBlock configuration by requesting the configuration of each individual NetworkSlave; this is referred to as an FBlock scan. The collected information is entered into the central registry.

The NetworkMaster sets the central registry state to OK to indicate that the central registry is valid or NotOK to indicate that the central registry is invalid. When the central registry state is OK, MOST nodes may communicate freely. When the central registry state is NotOK, communication is limited.

Setting the central registry state to NotOK resets any MOST network related information in all nodes.

The NetworkMaster sets the central registry state to NotOK whenever an error is caused by a NetworkSlave announcing its FBlocks. The NetworkMaster performs an FBlock scan as described in [6.8.3.6](#).

A transition to central registry state OK indicates the completion of the central registry initialisation.

6.8.1.2.3 APP — Initialisation on application level

After the NetworkMaster has set the central registry state to OK, initializations that set up network communication on the application layer should be performed.

Such initializations may include building of streaming connections, establishing MOST high protocol connections, building the decentral registry, and applying for notification.

6.8.1.3 APP — General operation — MOST network monitoring

The NetworkMaster monitors the MOST network for changes and errors. When a network change event (NCE) is detected, the NetworkMaster determines if this is due to a node entering or leaving the network. The NetworkMaster scans the network and reports any new information to all NetworkSlaves in the MOST network. Thereby, nodes are notified if one of its communication partners is missing or if new potential communication partners enter the MOST network.

The NetworkMaster may scan the MOST network at any time.

Nodes may activate and deactivate FBlocks at any time; these changes are reported to the NetworkMaster. The NetworkMaster then updates the central registry and informs all NetworkSlaves.

6.8.2 APP — Central registry state

6.8.2.1 APP — Distribution of the central registry state

The NetworkMaster FBlock distributes the central registry state to the NetworkSlaves by broadcasting `Configuration.Status` messages, using the blocking broadcast address.

It is crucial that `Configuration.Status` broadcast messages are received by all nodes in the MOST network. If at least one MOST NetworkSlave does not receive a `Configuration.Status` broadcast message, the MNC of the NetworkMaster performs low-level retries.

If these are not successful, the NetworkMaster should perform additional retries. The number of retries is network owner specific.

The sequence of additional retries is stopped when an event occurs that requires the NetworkMaster to send `Configuration.Status(NotOK)`.

`Configuration.Status` has the format `NetworkMaster.Configuration.Status(ConfigurationControl, DeltaFBlockList)`

ConfigurationControl is of data type Enum and specifies these values:

- 0₁₆: NotOK
- 1₁₆: OK
- 2₁₆: Invalid
- 3₁₆: Reserved
- 4₁₆: NewExt

The structure of the DeltaFBlockList parameter depends on ConfigurationControl:

- It contains pairs of FBlockID and InstID values for ConfigurationControl = Invalid.
- It contains groups of logical node address, FBlockID, and InstID for ConfigurationControl = NewExt.
- In all other cases, DeltaFBlockList is not used.

Figure 14 shows the states of the central registry in NetInterface state s_NetInterface_Normal_Operation. It shows the view of the NetworkMaster on the central registry states and which events occur. The view of the NetworkSlaves is shown in Figure 18.

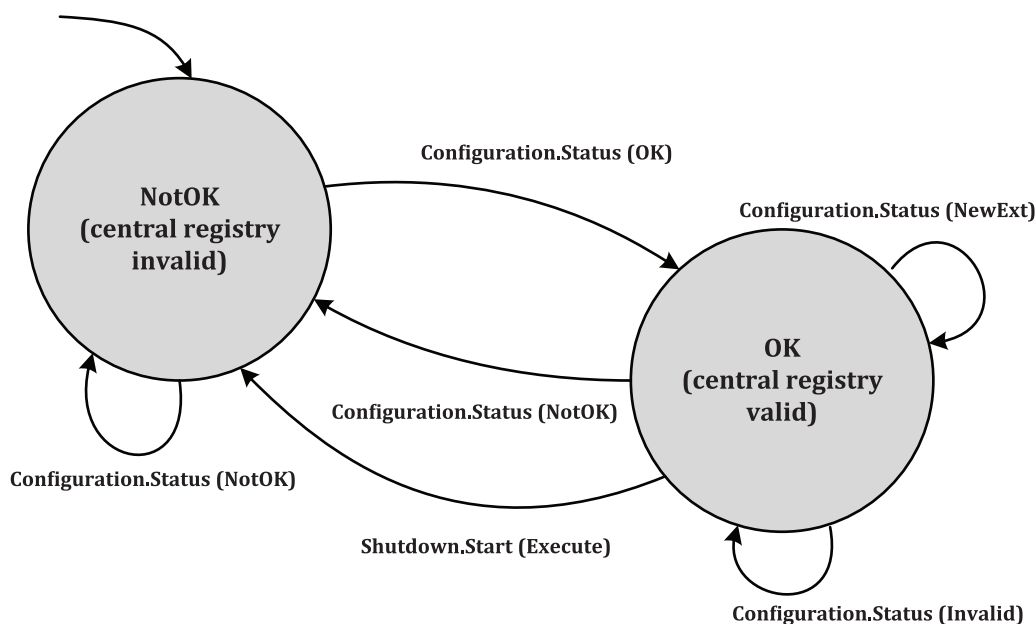


Figure 14 — States of the central registry in s_NetInterface_Normal_Operation

A network configuration reset consists of the following actions:

- Every FBlock containing streaming sinks sets the Status parameter of the Mute property to true for all sinks and disconnects them.
- Every FBlock containing streaming sources deallocates all sources.
- MOST high protocol connections are closed.
- Every NetworkSlave clears its decentral registry.
- The NetworkMaster clears the central registry.
- The connection manager deletes its connection table.
- Every FBlock that supports notification empties the notification matrix.

— All ArrayWindows on LongArrays are destroyed.

REQ	8.28 APP – Registry management – Network configuration reset on NotOK
Following <code>Configuration.Status(NotOK)</code> , all nodes shall perform a network configuration reset and set the logical node address (see 7.2.2).	

REQ	8.29 APP – Registry management – Network Configuration Reset during shutdown
Following <code>Shutdown.Start(Execute)</code> all nodes shall perform a network configuration reset.	

[6.8.3](#) and [6.8.4](#) describe node specific behaviour in the different central registry states, as well as making transitions between states.

6.8.2.2 APP — Central registry state NotOK

Central registry state NotOK is always entered after an `ev_Init_Ready` event. In this state, communication does not take place except for special applications that do not rely on a valid central registry, in particular the FBlock scan performed by the NetworkMaster and other optional features that may be done on a per-node, position-dependent basis. The MOST network can fall back into central registry state NotOK at any time by declaration of the NetworkMaster.

REQ	8.30 APP – Registry management – Events in central registry state NotOK
Table 5 specifies the events in central registry state NotOK (refer to Figure 14).	

Table 5 — Events in central registry state NotOK

Event	Transition to	Cause	Effect
<code>Configuration.Status(NotOK)</code>	No transition	— Uninitialised logical node address in NetworkMaster — Erroneous announcement by NetworkSlave	— Network configuration reset (see 6.8.2) — Set logical node address (see 7.2.2) — Notify application
<code>Configuration.Status(OK)</code>	Central registry state OK	— Central registry verified	— Network configuration available in central registry — Set up decentral registries, where necessary — (Re-) initialise applications — Notify application

6.8.2.3 APP — Central registry state OK

While in central registry state OK, the central registry is valid. So, the exact set of FBlocks in the MOST network, each with its attributes `InstID` and target address is specified. Therefore, application communication (i.e. messages with `FBlockIDs` other than NetBlock or NetworkMaster are allowed) may take place on the control channel and packet channel. All communication on the application level within the MOST network should be done only in central registry state OK.

The MOST network is regarded as being in central registry state NotOK after the PowerMaster sends `NetBlock.Shutdown.Start (Execute)` to the blocking broadcast address.

REQ	8.31 APP – Registry management – Do not set logical node addresses during shutdown
A node with a logical node address in the dynamic address range shall not set the address upon the implicit change of the central registry state to NotOK that is caused by the reception of <code>NetBlock.Shutdown.Start (Execute)</code> .	

Dynamic logical node addresses are set when the message `NetworkMaster.Configuration.Status (NotOK)` is received.

REQ	8.32 APP – Registry management – Events in central registry state OK
Table 6 specifies the events in central registry state OK (refer to Figure 14).	

Table 6 — Events in central registry state OK

Event	Transition to	Cause	Effect
<code>NetBlock.Shutdown.Start (Execute)</code>	Central registry state NotOK	— Shutdown by PowerMaster	— Network configuration reset (see 6.8.2) — Notify application
<code>Configuration.Status (NotOK)</code>	Central registry state NotOK	— Erroneous announcement by NetworkSlave.	— Network configuration reset (see 6.8.2) — Set logical node address (see 7.2.2) — Notify application
<code>Configuration.Status (NewExt)</code>	No transition	— New FBlocks are available	— Notify application
<code>Configuration.Status (Invalid)</code>	No transition	— FBlocks were removed	— Notify application

In central registry state OK, the NetworkMaster FBlock should not send `Configuration.Status (OK)`.

6.8.3 APP — NetworkMaster

6.8.3.1 APP — Overview

The node that contains the NetworkMaster FBlock is referred to as the NetworkMaster.

REQ	8.33 APP – NetworkMaster – Uniqueness
There shall be exactly one NetworkMaster in a MOST network.	

The NetworkMaster controls the central registry state and administrates the central registry. The NetworkMaster monitors the MOST network for certain events and continuously manages incoming information from NetworkSlaves about their current FBlock configuration and whenever necessary informs all NetworkSlaves about updates to the central registry.

6.8.3.2 APP — Network change event

An NCE occurs if a node opens or closes its bypass, that is, enters or leaves the network.

REQ	8.34 APP – NetworkMaster – Network change event
If an NCE occurs, the NetworkMaster shall perform an FBlock scan and send <code>Configuration.Status</code> to the blocking broadcast address.	

6.8.3.3 APP — Setting the central registry state

6.8.3.3.1 APP — Distributing the central registry state

The NetworkMaster FBlock distributes the central registry state by broadcasting `Configuration.Status` messages, using the blocking broadcast address. More information about the different central registry states and `Configuration.Status` messages is available in [6.8.2](#).

6.8.3.3.2 APP — Setting the central registry state to OK

By setting the central registry state to OK, the NetworkMaster confirms the validity of the central registry.

6.8.3.3.3 APP — Setting the central registry state to NotOK

By sending `Configuration.Status(NotOK)` to the blocking broadcast address, the NetworkMaster FBlock resets any MOST network related information in all nodes. The NetworkMaster then executes the actions that are described in [6.8.2](#).

REQ	8.35 APP – NetworkMaster – Terminate FBlock scan on NotOK
If the NetworkMaster sends <code>NetworkMaster.Configuration.Status(NotOK)</code> while performing an FBlock scan, the NetworkMaster shall terminate the FBlock scan.	

REQ	8.36 APP – NetworkMaster – Start FBlock scan after NotOK
After sending <code>NetworkMaster.Configuration.Status(NotOK)</code> to the blocking broadcast address, the NetworkMaster shall start an FBlock scan (see 6.8.3.6).	

The NetworkMaster notifies the NetworkSlaves with a `Configuration.Status(NotOK)` message if the central registry becomes unavailable due to an internal failure of the NetworkMaster.

6.8.3.4 APP — Central registry

6.8.3.4.1 APP — Overview

The NetworkMaster generates the central registry during the initialisation of the network and continues to administrate it until network shutdown (see [6.9.3](#)). The central registry is an image of the MOST network configuration. It contains the logical node address and the respective FBlocks of each node, as illustrated in [Table 7](#).

In addition to the central registry, the NetworkMaster stores the node position corresponding to every logical node address.

Table 7 — Example of a central registry

Logical node address	FBlockID (FBlock name)	InstID
0100 ₁₆	31 ₁₆ (AudioDiskPlayer)	1
	02 ₁₆ (NetworkMaster)	1
	03 ₁₆ (ConnectionMaster)	1
0101 ₁₆	31 ₁₆ (AudioDiskPlayer)	2
0102 ₁₆	40 ₁₆ (AMFMTuner)	1
	30 ₁₆ (AudioTapeRecorder)	1
0103 ₁₆	22 ₁₆ (AudioAmplifier)	2
:	:	:
0100 ₁₆ + MaxNode ^a	10 ₁₆ (HumanMachineInterface)	1

^a MaxNode represents the maximum node position information decreased by 1.

6.8.3.4.2 APP — Purpose

The central registry is used when the NetworkMaster checks the FBlock configuration and when nodes are searching for communication partners.

6.8.3.4.3 APP — Contents

The central registry contains the logical node address of each node and the respective functional addresses (combination of FBlockIDs and InstIDs) of its FBlocks. The NetworkMaster makes this information available to all NetworkSlaves.

REQ	8.37 APP – NetworkMaster – FBlocks excluded from central registry
	The NetworkMaster shall not include the FBlocks 01 ₁₆ (NetBlock), 09 ₁₆ (DebugMessages), and 0F ₁₆ (EnhancedTestability) in the central registry.

6.8.3.4.4 APP — Responsibility

Any new information gained regarding the FBlock configuration is entered into the central registry and distributed to all NetworkSlaves as described in [6.8.3.9](#).

6.8.3.4.5 APP — Responding to requests for information from the central registry

The NetworkMaster FBlock responds to requests for `CentralRegistry.Get` from the NetworkSlaves while the central registry state is OK.

If the NetworkMaster FBlock receives `CentralRegistry.Get` from a NetworkSlave while the MOST network is in central registry state NotOK, the NetworkMaster FBlock answers the request with the `ErrorCode 4116` and sends `Configuration.Status(NotOK)` to the blocking broadcast address.

Broadcasting the central registry state again ensures that every node in the MOST network is updated on the current central registry state, including and in particular the NetworkSlave attempting to read the central registry.

To obtain, for example, all instances of a specific FBlock, a controller sends the following command to the NetworkMaster node (NWM):

```
Controller → NWM: NetworkMaster.CentralRegistry.Get(FBlockID, InstID = FF16)
```

InstID FF₁₆ is a wildcard to address all instances. The NetworkMaster answers with a list of all matching entries in the central registry, containing logical node address and functional address:

```
NWM → Controller: NetworkMaster.CentralRegistry.Status(FBlockInfoList =  
<LogicalNodeAddress, FBlockID, InstID,
```

LogicalNodeAddress, FBlockID, InstID,...>)

To search for a particular instance of an FBlock, the requesting controller provides the InstID of that instance in CentralRegistry.Get.

REQ	8.38 APP – NetworkMaster – Requested FBlockID.InstID does not exist
If the FBlockID.InstID combination requested in CentralRegistry.Get does not exist, the NetworkMaster FBlock shall reply with ErrorCode 07 ₁₆ (parameter not available).	
REQ	8.39 APP – NetworkMaster – Requested FBlockID does not exist
If the FBlockID requested in CentralRegistry.Get does not exist, in ErrorInfo, the NetworkMaster FBlock shall return 01 ₁₆ and the FBlockID.	
REQ	8.40 APP – NetworkMaster – Requested InstID does not exist
If the FBlockID requested in CentralRegistry.Get exists but the InstID does not, in ErrorInfo, the NetworkMaster FBlock shall return 02 ₁₆ and the InstID.	
REQ	8.41 APP – Registry management – FBlockID, InstID combinations
Table 8 specifies the FBlockID, InstID combinations for reading the central registry.	

Table 8 — FBlockID, InstID combinations for reading the central registry

FBlockID	InstID 00 ₁₆	InstID 01 ₁₆ to FE ₁₆	InstID FF ₁₆
00 ₁₆	These FBlockID.InstID combinations are not allowed. The NetworkMaster FBlock responds with ErrorCode 06 ₁₆ (parameter out of range).		
01 ₁₆ to FE ₁₆	If InstID 00 ₁₆ of the requested FBlock exists, it is reported. Otherwise, one existing InstID of that FBlock is reported. However, while the MOST network remains in central registry state OK and the InstID is still present, the same InstID is reported in subsequent calls with identical parameters.	If it exists, the specific FBlockID.InstID combination is returned. If the FBlockID.InstID combination does not exist, ErrorCode 07 ₁₆ (parameter not available) is returned.	All existing InstIDs of the requested FBlock are reported.
FF ₁₆	These FBlockID.InstID combinations are not allowed. The NetworkMaster FBlock responds with ErrorCode 06 ₁₆ (parameter out of range). FBlockID FF ₁₆ requires InstID FF ₁₆ .		The entire central registry is returned.

6.8.3.5 APP — Specific behaviour after ev_Init_Ready

6.8.3.5.1 APP — Validity of logical node addresses

After the ev_Init_Ready event, the NetworkMaster initialises the MOST network.

This process depends on the availability of a valid logical node address. A valid logical node address is any address within the dynamic or static address ranges as specified in [7.2.2](#).

6.8.3.5.2 APP — Valid logical node address not available

REQ	8.42 APP – NetworkMaster – No valid logical node address
If the NetworkMaster does not have a valid logical node address available when the <code>ev_Init_Ready</code> event occurs, it shall:	
— set a valid logical node address, — set the central registry state to NotOK (see 6.8.3.3.3), and — start an FBlock scan (see 6.8.3.6).	

6.8.3.5.3 APP — Valid logical node address available

REQ	8.43 APP – NetworkMaster – Valid logical node address
If the NetworkMaster has a valid logical node address when the <code>ev_Init_Ready</code> event occurs, the NetworkMaster shall use this logical node address.	

REQ	8.44 APP – NetworkMaster – Starting FBlock scan
If the NetworkMaster has a valid logical node address when the <code>ev_Init_Ready</code> event occurs, the NetworkMaster shall start an FBlock scan (see 6.8.3.6).	

6.8.3.6 APP — Scanning the MOST network**6.8.3.6.1 APP — FBlock scan**

The NetworkMaster scans the MOST network after the `ev_Init_Ready` event and after an NCE. It may also be instructed to scan the MOST network at any other time.

REQ	8.45 APP – NetworkMaster – Performing FBlock scan
When performing an FBlock scan, the NetworkMaster shall request <code>NetBlock.FBlockIDs.Get</code> from each NetworkSlave.	

When performing an FBlock scan, the NetworkMaster should not wait for the response from a node before sending the next request.

The responses from the NetworkSlaves are interpreted as described in [6.8.3.7](#). Any information gained concerning the configuration of the network is written to the central registry and reported to all NetworkSlaves as described in [6.8.3.9](#).

The NetworkMaster sets the central registry state to NotOK whenever an error is caused by a NetworkSlave announcing its FBlocks.

REQ	8.46 APP – NetworkMaster – Setting the central registry state to OK
If during the FBlock scan in central registry state NotOK all nodes respond with <code>FBlockIDs.Status</code> and no invalid announcements occurs (see 6.8.3.7 and 6.8.3.8), the NetworkMaster shall send <code>Configuration.Status(OK)</code> to the blocking broadcast address.	

6.8.3.6.2 APP — Configuration request description

During a network scan, the NetworkMaster requests `NetBlock.FBlockIDs.Get` from each NetworkSlave.

REQ	8.47 APP – NetworkMaster – Processing messages in order
For each node, the NetworkMaster shall process <code>NetBlock.FBlockIDs.Status</code> messages in the order they are received.	

REQ	8.48 APP – NetworkMaster – InstID MSb evaluation
The NetworkMaster shall evaluate whether the MSb of the <code>InstID</code> of the <code>NetBlock</code> is set to 1 in the message <code>NetBlock.FBlockIDs.Status</code> (see 6.8.4.4.3).	

A NetworkSlave requests the removal of its FBlocks from the central registry by setting the MSb of the `InstID` of the `NetBlock` to 1 in the message `NetBlock.FBlockIDs.Status`. For example, the NetworkSlave requests the removal of its FBlocks from the central registry when it just woke up from reset.

REQ	8.49 APP – NetworkMaster – Setting the InstID MSb
When sending <code>NetBlock.FBlockIDs.Get</code> , the NetworkMaster shall set the most significant bit (MSb) of the <code>InstID</code> to 1.	

REQ	8.50 APP – NetworkMaster – No InstID wildcard FF_{16}
The NetworkMaster shall not use the <code>InstID</code> wildcard FF_{16} in <code>NetBlock.FBlockIDs.Get</code> .	

Thus, there is no danger of generating a conflict between setting the MSb of the `InstID` and the usage of the wildcard FF_{16} .

6.8.3.6.3 APP — NetworkMaster performs initial FBlock scan

In the initial FBlock scan, the NetworkMaster can ignore the MSb because the central registry is empty and no NetworkSlave has announced its FBlocks in the MOST network yet.

REQ	8.51 APP – NetworkMaster – Ignoring InstID MSb during initial FBlock scan
In the initial FBlock scan (FBlock scan while in central registry state NotOK), the NetworkMaster shall ignore the MSb of the <code>InstID</code> (see 6.8.4.4.3) of all received <code>NetBlock.FBlockIDs.Status</code> messages.	

6.8.3.6.4 APP — NetworkMaster performs rescan during central registry state OK

REQ	8.52 APP – NetworkMaster – Deleting FBlocks due to InstID MSb
In central registry state OK, when the NetworkMaster receives a <code>NetBlock.FBlockIDs.Status</code> message where the MSb of the <code>InstID</code> is set to 1, the NetworkMaster shall delete all FBlocks that belong to the issuing node from the central registry.	

The NetworkMaster then sends the message `NetworkMaster.Configuration.Status(Invalid, DeltaFBlockList)` to the blocking broadcast address (see [6.8.3.9.2](#)).

This usually leads to an “own configuration invalid” handling (see [6.8.4.4.11](#)) in the affected NetworkSlave. This is the recommended behaviour.

6.8.3.6.5 APP — Addressing

REQ	8.53 APP – NetworkMaster – Scanning with node position addressing
The NetworkMaster shall scan the MOST network using node position addressing.	

The logical node address of the requested NetworkSlave is contained in the response message.

6.8.3.6.6 APP — Non-responding NetworkSlaves

The NetworkMaster waits until the expiration of $t_{\text{WaitForAnswer}}$ for a reply from a NetworkSlave.

When a NetworkSlave does not respond to a request, the NetworkMaster tries again after $t_{\text{DelayCfgRequest1}}$ or $t_{\text{DelayCfgRequest2}}$. $t_{\text{DelayCfgRequest1}}$ is used after each of the first 20 requests after entering `NetInterface` state `s_NetInterface_Normal_Operation`; then, $t_{\text{DelayCfgRequest2}}$ is used.

Refer to [6.13.2](#) for more information about timers.

A NetworkSlave that answers a request from the NetworkMaster with an error is treated as a non-responding NetworkSlave.

6.8.3.6.7 APP — Duration of FBlock scanning

REQ	8.54 APP – NetworkMaster – Scan duration
The NetworkMaster shall continue to scan the MOST network until all NetworkSlaves have answered (see 6.8.3.6.6).	

6.8.3.6.8 APP — Reporting the results of an FBlock scan

REQ	8.55 APP – NetworkMaster – Reporting new information
The NetworkMaster shall report the result of the FBlock scan if it has any new information to distribute.	

New information includes changes in the central registry state or changes in the FBlock configuration of one or more NetworkSlaves.

If $t_{\text{WaitForAnswer}}$ expires and no error condition exists (see [6.8.3.7](#)), the NetworkMaster sets the central registry state to OK.

Refer to [6.8.3.3](#), [6.8.3.9](#), and [6.8.3.10.1](#).

6.8.3.7 APP — Invalid announcements — Node address

The NetworkMaster interprets the incoming FBlock announcements by NetworkSlaves and determines if the announcement is accepted. The following are considered to be invalid announcements.

A “conflicting node address” is a logical node address that is either outside the dynamic and static address range or a duplicate logical node address.

A “duplicate logical node address” refers to an FBlock announcement from a NetworkSlave in which its logical node address is used by another NetworkSlave.

Refer to [7.2.2](#) for more information about the valid address range.

The respective nodes are identified by the `InstIDs` of their NetBlock FBlocks (representing the node position).

The maximum number of permitted FBlock announcements with conflicting node address by a NetworkSlave is specified by the network owner.

REQ	8.56 APP – NetworkMaster – Counting conflicting node address announcements
The NetworkMaster shall count the occurrence of FBlock announcements with conflicting node address by each NetworkSlave.	

REQ	8.57 APP – NetworkMaster – Increasing the error counter
In the case of duplicate logical node addresses, the NetworkMaster shall increase the error counter for all NetworkSlaves that share the address by one.	

REQ	8.58 APP – NetworkMaster – Setting NotOK due to conflicts
If a NetworkSlave announces FBlocks with a conflicting node address and the maximum number of announcements with conflicting node address is not exceeded, the NetworkMaster shall send <code>Configuration.Status(NotOK)</code> to the blocking broadcast address.	

Sending `Configuration.Status(NotOK)` terminates any ongoing FBlock scan.

If the NetworkSlave's current node address is outside the static address range, the NetworkSlave sets the calculated dynamic logical node address.

REQ	8.59 APP – NetworkMaster – Ignoring NetworkSlave due to conflicts
After a NetworkSlave announces its FBlocks with a conflicting node address for the permitted maximum number of times, the NetworkMaster shall ignore the NetworkSlave.	

In this context, “ignore” means that the NetworkMaster:

- does not request FBlockIDs from the respective NetworkSlave,
- does not include the FBlocks of the respective NetworkSlave in the central registry.

Other messages from ignored NetworkSlaves may still be processed.

REQ	8.60 APP – NetworkMaster – Ignoring NetworkSlaves with duplicate address
The NetworkMaster shall ignore all NetworkSlaves that share a duplicate logical node address.	

REQ	8.61 APP – NetworkMaster – Resetting the error counter
If an NCE or shutdown occurs, the NetworkMaster shall reset the conflicting node address error counters for all NetworkSlaves to zero.	

The network owner may specify other events so that the NetworkSlave is no longer ignored.

[Figure 15](#) shows how the NetworkMaster reacts when detecting node address conflicts.

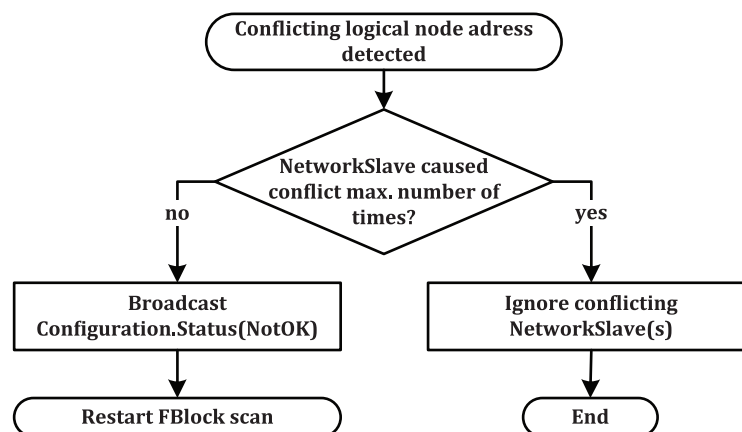


Figure 15 — NetworkMaster behaviour in the case of an address conflict

6.8.3.8 APP — Invalid announcements — InstID

The NetworkMaster is responsible for the uniqueness of functional addresses (FBlockID.InstID) in the MOST network. The NetworkMaster attempts to resolve the issue of two or more NetworkSlaves announcing identical functional addresses.

REQ	8.62 APP – NetworkMaster – InstID setting strategy
For duplicate InstIDs, the NetworkMaster shall not change the InstID of the FBlock that is already contained in the central registry.	

REQ	8.63 APP – NetworkMaster – Assigning valid InstID
The NetworkMaster shall assign a valid InstID for FBlock announcements by NetworkSlaves that contain the InstID value 00 ₁₆ or FF ₁₆ .	

If a NetworkSlave incorrectly reports 00_{16} or FF_{16} as `InstID` and the NetworkMaster assigns a new `InstID` value, 00_{16} or FF_{16} are not interpreted as wildcards but as actual `InstID` values.

The NetworkMaster determines a new `InstID` for the last FBlock to be included. It then sets the new `InstID` in the corresponding NetworkSlave by using the logical node address. If the new `InstID` is accepted by the NetworkSlave, the NetworkMaster enters this `InstID` into the central registry.

The NetworkMaster informs all NetworkSlaves as described in [6.8.3.9](#) or by ultimately setting the central registry state to OK.

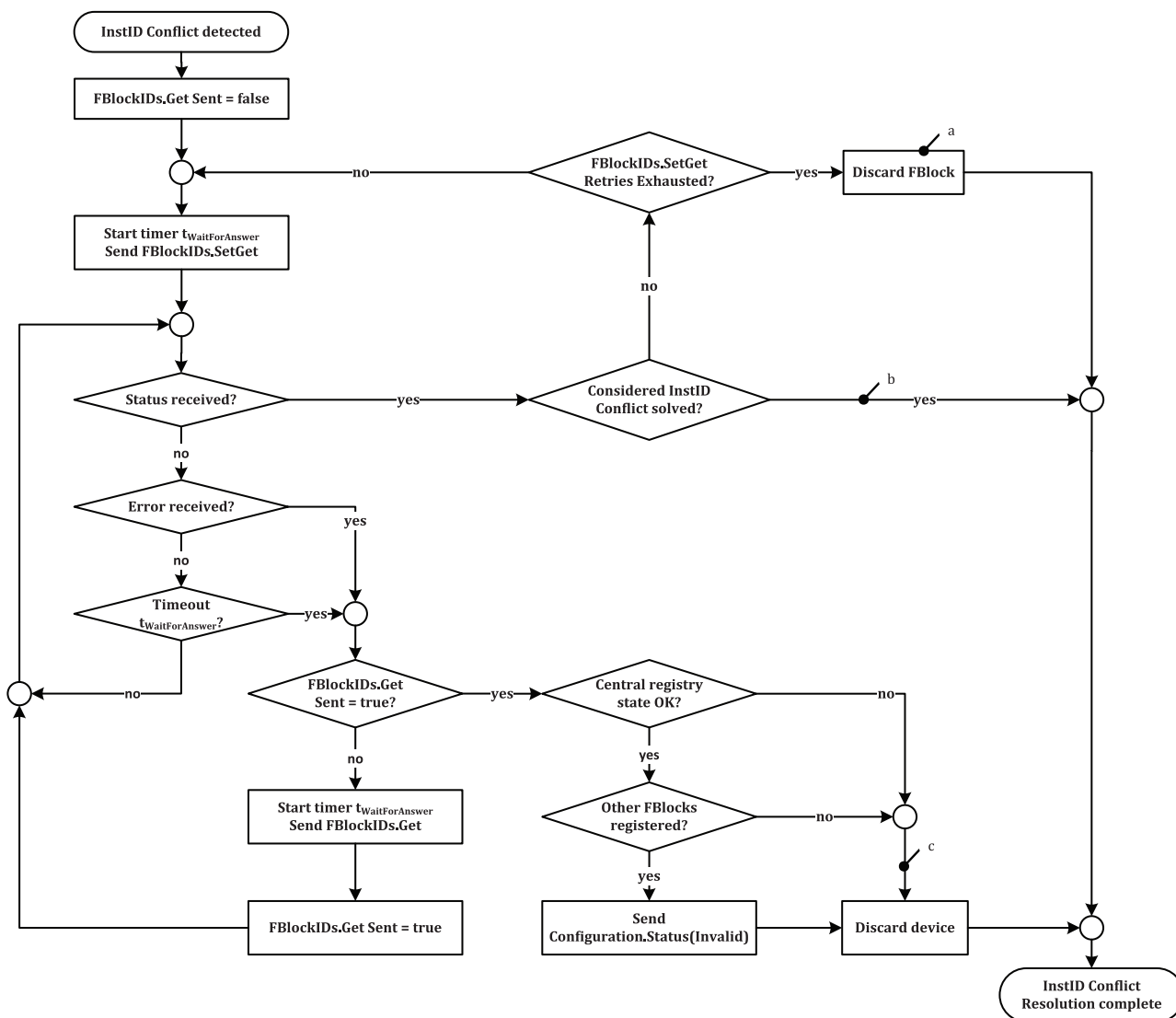
If a NetworkSlave does not accept a new `InstID`, it responds with an unchanged `NetBlock.FBlockIDs.Status` list.

The NetworkMaster repeats the procedure with the same or another `InstID` value. The maximum number of attempts per instance is specified by the network owner.

If the request to change the `InstID` of a conflicting FBlock is not successful, the NetworkMaster either excludes the FBlock or the entire node from the central registry.

When a status message is received, only the FBlock is discarded. The whole node is discarded only in the case that the NetworkSlave is considered non-responding following [6.8.3.6.6](#).

[Figure 16](#) shows the `InstID` conflict resolution by the NetworkMaster.



a Status received.

b Good case.

c Error received or no answer.

Figure 16 — InstID conflict resolution by the NetworkMaster

6.8.3.9 APP — Updates to the central registry

6.8.3.9.1 APP — Changes in central registry state OK

The NetworkMaster informs all NetworkSlaves about changes of the FBlock configuration. This information may become available during an FBlock scan or as NetworkSlaves make additional FBlock announcements, which are not requested by the NetworkMaster.

This subclause describes how the NetworkMaster handles changes to the FBlock configuration while in central registry state OK.

REQ	8.64 APP - NetworkMaster - Single transfer
The NetworkMaster shall not put more entries into <code>DeltaFBlockList of Configuration.Status</code> than a single transfer can transport.	

If there are more FBlocks, several single transfers are performed.

6.8.3.9.2 APP — Disappearing FBlocks in central registry state OK

6.8.3.9.2.1 APP — NetworkMaster reaction

REQ	8.65 APP – NetworkMaster – Reporting removed FBlocks
If the NetworkMaster receives a <code>NetBlock.FBlockIDs.Status</code> message in which one or more FBlocks are missing, compared to the previous message from the same NetworkSlave, the NetworkMaster shall update the central registry and send this message to the blocking broadcasting address: <code>Configuration.Status(Invalid, DeltaFBlockList).</code>	

The `DeltaFBlockList` parameter is a list of the previously contained but now invalid FBlocks.

REQ	8.66 APP – NetworkMaster – ConnectionMaster removal causes NotOK
If the FBlock <code>03₁₆</code> (ConnectionMaster) is removed from the central registry, the NetworkMaster shall send <code>Configuration.Status(NotOK)</code> to the blocking broadcast address.	

Afterwards, an FBlock scan is performed (see [6.8.3.3.3](#)).

When one or more FBlocks disappear, the NetworkMaster should inform all NetworkSlaves about the missing FBlocks as quickly as possible, even if this information is gained while scanning the MOST network.

6.8.3.9.2.2 APP — Own configuration invalid handling (optional)

If the NetworkMaster FBlock receives an `OwnConfigInvalid.StartResultAck(..., State = active, LogicalNodeAddress)` from a NetworkSlave, the NetworkMaster starts the own configuration invalid handling.

The end of the own configuration invalid state is reached when the NetworkMaster completes the handling of own configuration invalid by removing all FBlocks of the node from the central registry, in which case it sends `OwnConfigInvalid.ResultAck(State = finished)` to the NetworkSlave that initiated the handling. The end of the own configuration invalid state is also reached when a transition to `NetInterface` state `s_NetInterface_Off` occurs or the central registry state is set to NotOK.

REQ	8.67 APP – NetworkMaster – OwnConfigInvalid
The NetworkMaster shall be capable of handling <code>OwnConfigInvalid</code> messages from multiple nodes.	

6.8.3.9.3 Appearing FBlocks in central registry state OK

REQ	8.68 APP – NetworkMaster – Reporting additional FBlocks
If the NetworkMaster receives a <code>NetBlock.FBlockIDs.Status</code> message which contains one or more additional FBlocks, compared to the previous message from the same NetworkSlave, the NetworkMaster shall update the central registry and send this message to the blocking broadcast address: <code>Configuration.Status(NewExt, DeltaFBlockList).</code>	

The `DeltaFBlockList` parameter is a list of the new FBlocks, `InstIDs`, and the corresponding logical node addresses.

If this information is gained while scanning the MOST network, the NetworkMaster may continue to scan the MOST network before it informs all NetworkSlaves.

If a NetworkSlave announces FBlocks and the NetworkMaster cannot include them in the central registry because it is “full”, the NetworkMaster does not send a `Configuration.Status` message.

In the case that some but not all of the new FBlocks can be included in the central registry, the broadcast list is not empty but contains the subset of the instances that were included in the central registry:

```
Configuration.Status(NewExt, <partial list>)
```

6.8.3.9.4 APP — FBlock scan without any change in central registry

REQ	8.69 APP – NetworkMaster – Empty list after NCE
The NetworkMaster shall send <code>Configuration.Status(NewExt)</code> with an empty list to the blocking broadcast address when a network scan triggered by an NCE completes without any changes to the central registry.	

6.8.3.9.5 APP — Non-responding nodes in central registry state OK

REQ	8.70 APP – NetworkMaster – Removing NetworkSlave after $t_{\text{WaitForAnswer}}$
If an update of the central registry is triggered because $t_{\text{WaitForAnswer}}$ expires due to a NetworkSlave, which is included in the central registry and does not respond to a request, the NetworkMaster shall remove this NetworkSlave from the central registry.	

Subsequently, the NetworkMaster informs all NetworkSlaves as described in [6.8.3.9.2](#).

6.8.3.10 APP — Miscellaneous NetworkMaster requirements

6.8.3.10.1 APP — Network change event (NCE)

REQ	8.71 APP – NetworkMaster – Starting FBlock scan on NCE
When the NetworkMaster detects an NCE, it shall start an FBlock scan (see 6.8.3.6).	

This also applies to a scan after the `ev_Init_Ready` event (see [6.8.3.5](#)).

When an NCE is detected, the NetworkMaster interrupts and restarts any ongoing scan.

6.8.3.10.2 APP — Positioning of the FBlock NetworkMaster in the MOST network

REQ	8.72 APP – NetworkMaster – TimingMaster in the same node
The NetworkMaster FBlock shall reside in the same node as the TimingMaster.	

6.8.4 APP — NetworkSlave

6.8.4.1 APP — General

All nodes that do not contain the NetworkMaster FBlock are called NetworkSlaves.

A NetworkSlave keeps the NetworkMaster informed about its current FBlock configuration. In particular, the NetworkSlave is responsible for ensuring that `FBlockIDs.Status` messages, which announce or remove FBlocks, are successfully transmitted to the node containing the NetworkMaster FBlock.

6.8.4.2 APP — Decentral registry

6.8.4.2.1 APP — General

Every node that controls other nodes should build a decentral registry in which it lists its communication partners. The node that contains the NetworkMaster FBlock, which implements the central registry, does not necessarily build a decentral registry.

A decentral registry contains the functional addresses and the respective logical node address, as illustrated in [Table 9](#).

Table 9 — Example of a decentral registry

Functional address (FBlockID.InstID)	Logical node address
AudioAmplifier.1	0105 ₁₆
AudioAmplifier.2	0103 ₁₆
AMFMTuner.1	0107 ₁₆
AudioDiskPlayer.1	0107 ₁₆

Since applications know only functional but not node addresses, a MOST message is complemented by the target address. There are two ways to achieve this:

- When an FBlock answers a request it already has the node address of the controller from the request.
- When a controller sends a message and does not know the node address of the target, the target address is obtained from the central registry (see 6.8.3.4) or the decentral registry (see 6.8.4.2).

Figure 17 shows how a NetworkSlave seeks the logical node address of a communication partner.

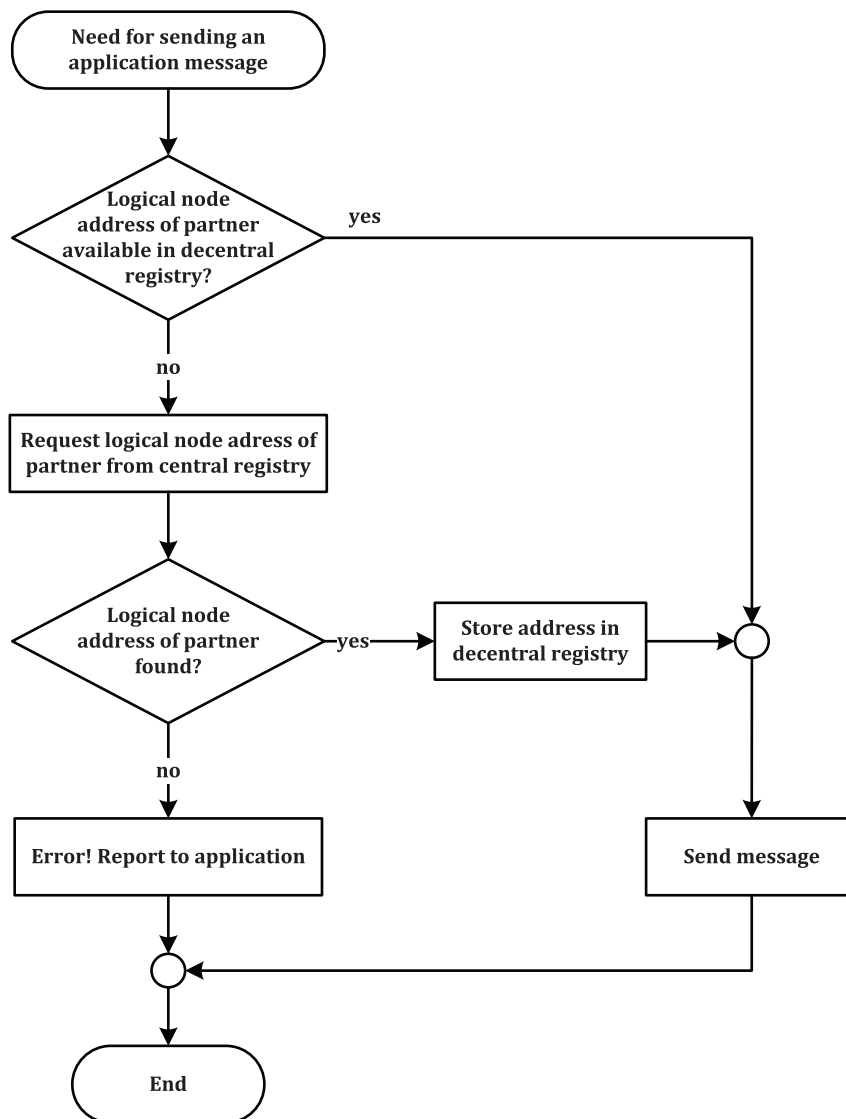


Figure 17 — Seeking the logical node address of a communication partner