INTERNATIONAL STANDARD

ISO 21806-2

First edition
2020-09

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

Reference number
ISO 21806-2:2020(E)

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

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, MOST[1)] 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.

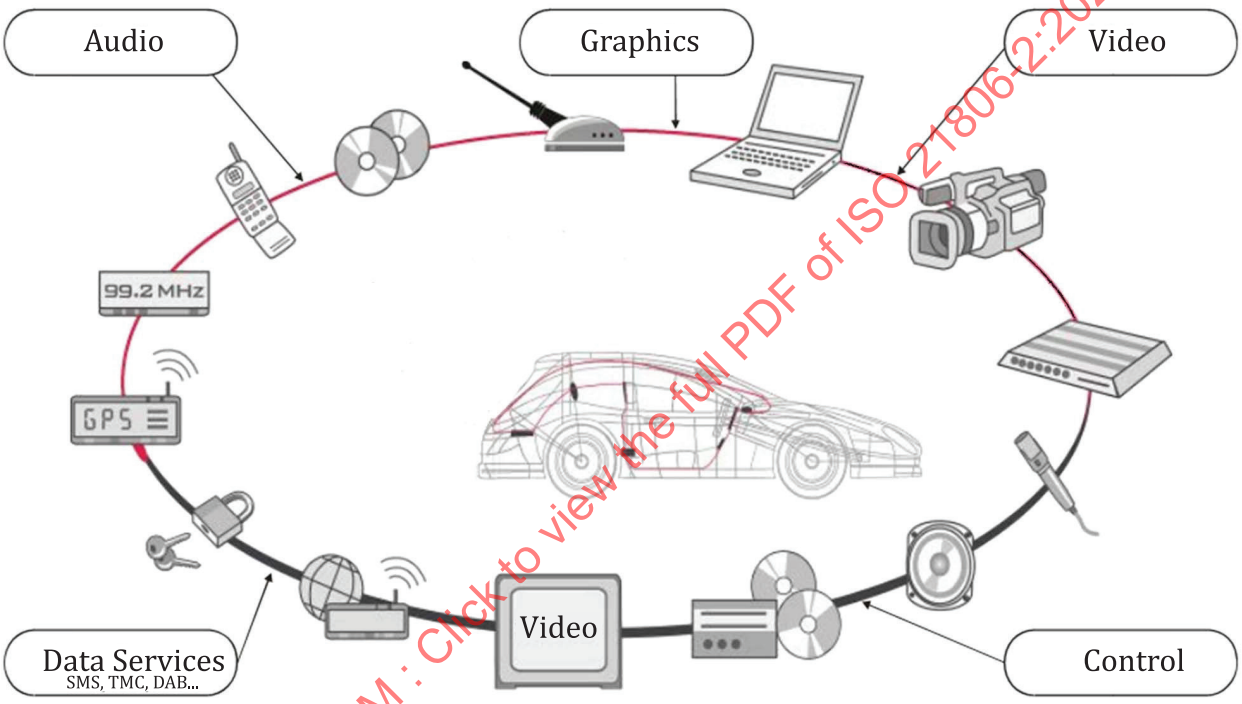


Figure 1 — MOST network example

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.

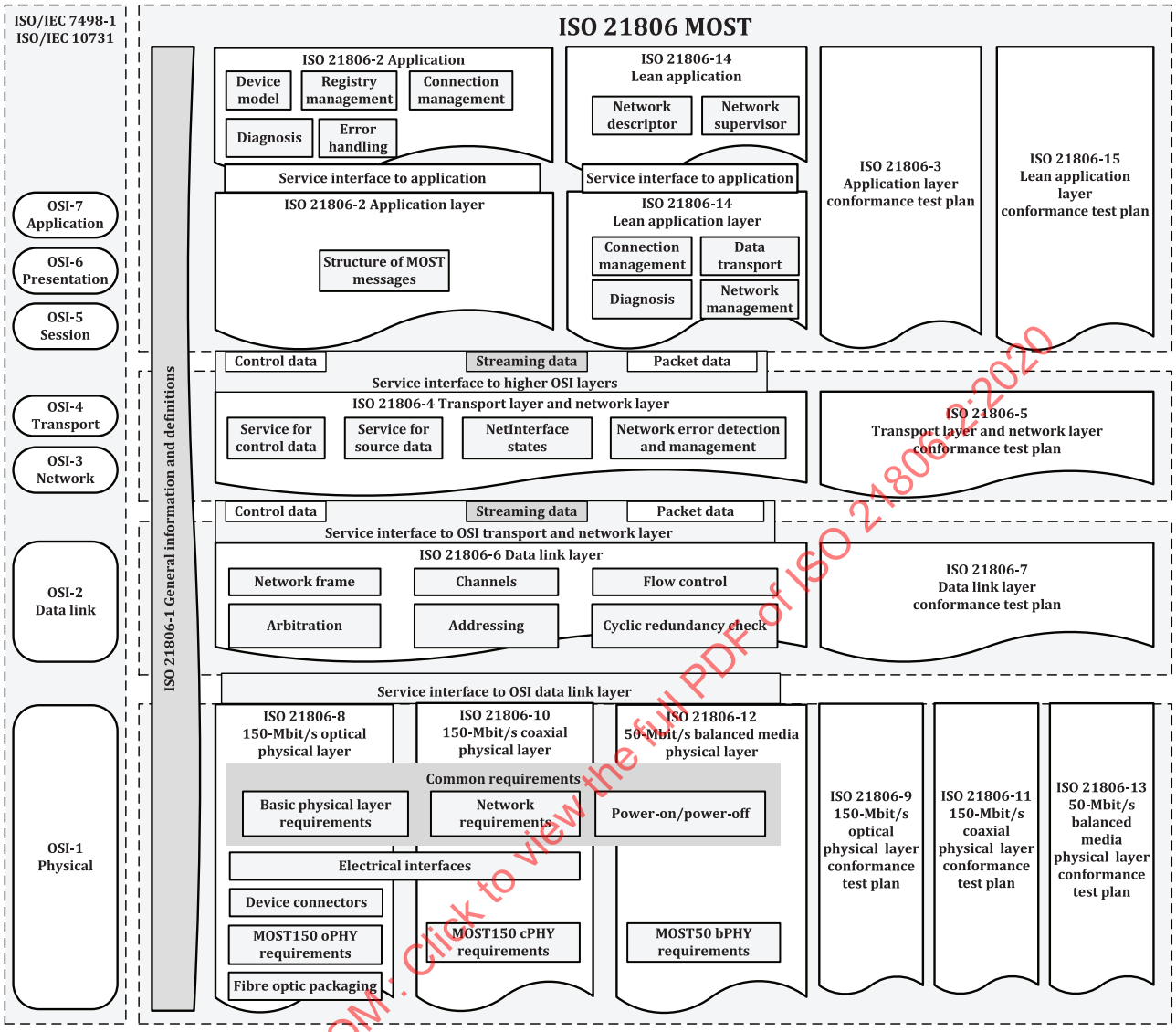


Figure 2 — The ISO 21806 series reference according to the OSI model

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

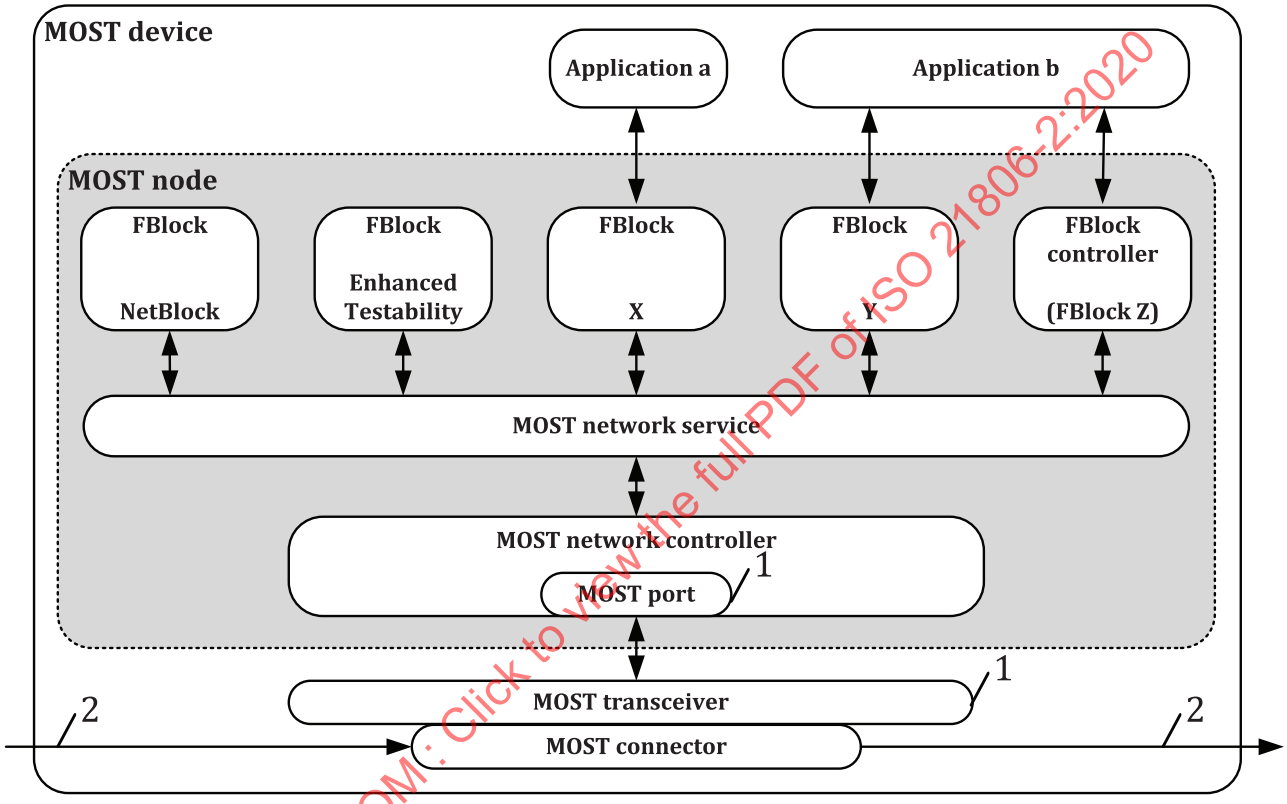
The following subclauses describe the logical model of a MOST device. A MOST device is a physical unit that can be connected to a MOST network via a MOST network controller (MNC).

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.



Key

1 multiple instances are possible

2 MOST network

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 00_{16} to $0F_{16}$), 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.

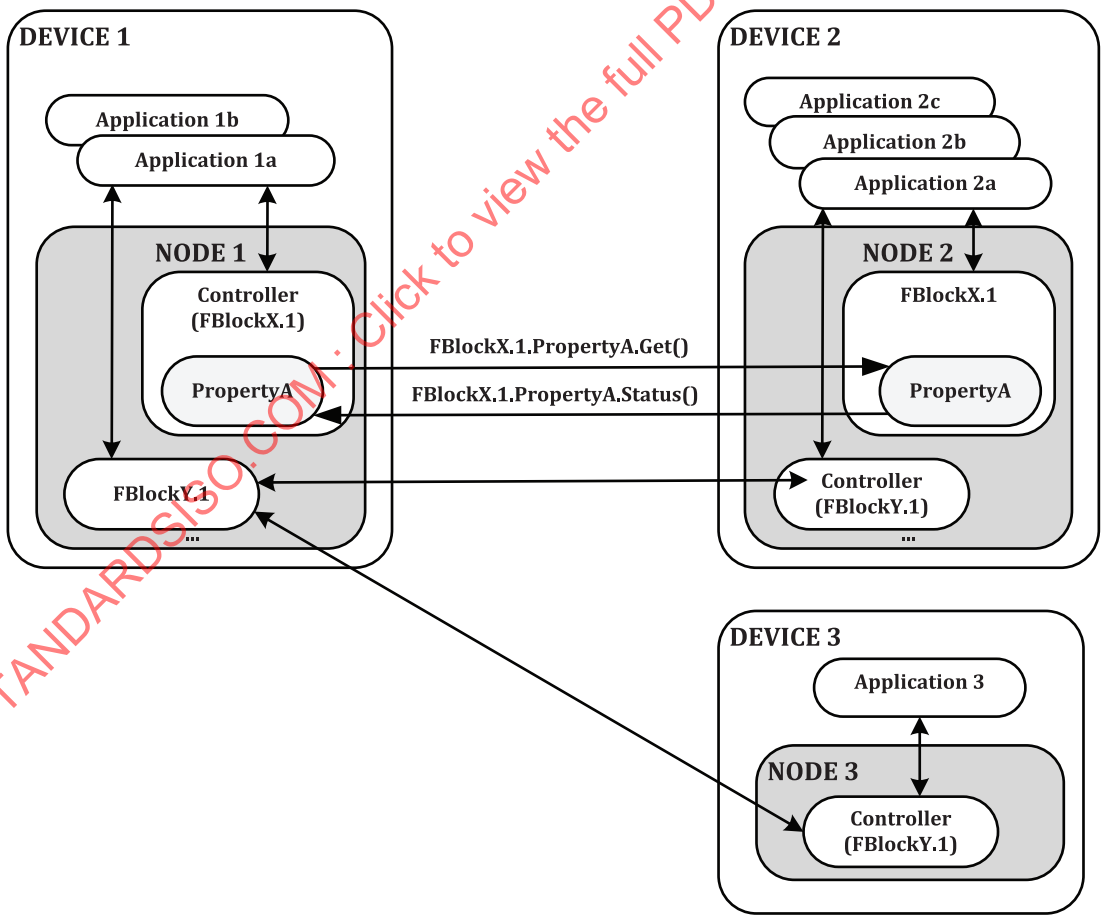


Figure 4 — Application, controller, and FBlock interaction example

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.

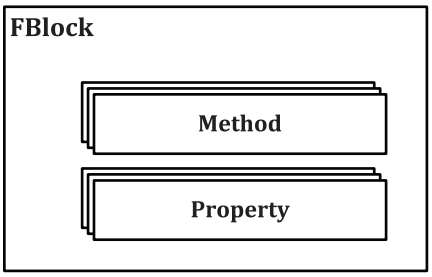


Figure 5 — Structure of an FBlock consisting of functions classifiable as methods and properties

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` 01_{16}), 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 — OPTypes for methods

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

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

REQ	8.15 APP – Methods – SenderHandle value
A controller shall choose a unique value for `SenderHandle`.	

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

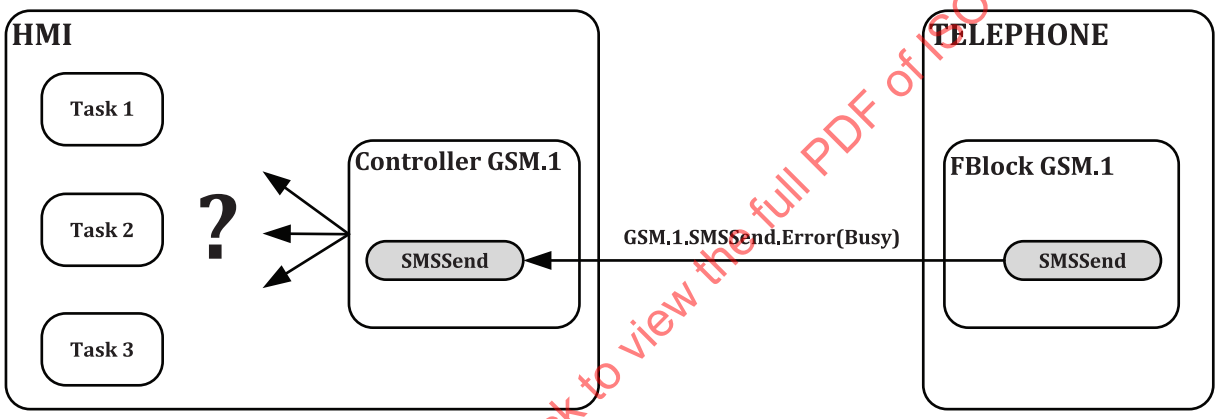


Figure 6 — Erroneous routing without the availability of a SenderHandle

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

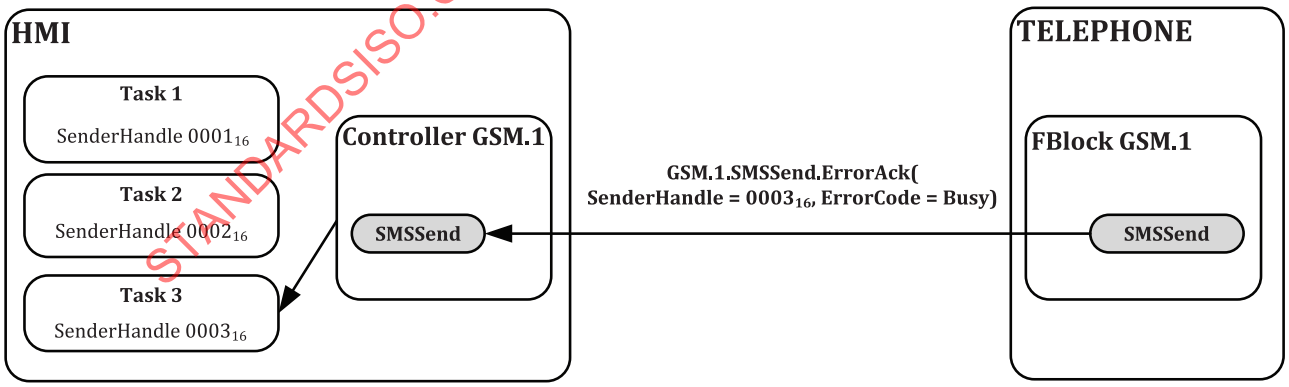


Figure 7 — Routing with the use of a SenderHandle

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 = 0003_16, SMSData)
```

It then receives this answer:

```
Telephone → HMI: GSM.1.SMSSend.ErrorAck(SenderHandle = 0003_16, 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.

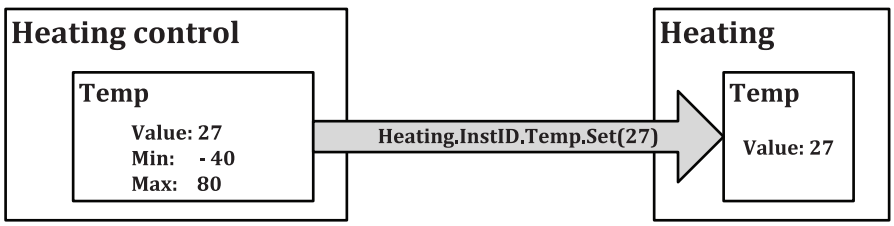


Figure 8 — Writing a property (temperature setting of a heating)

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`.

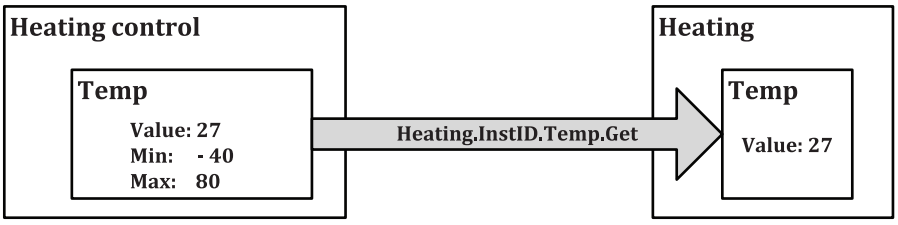


Figure 9 — Reading a property (temperature setting of a heating)

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

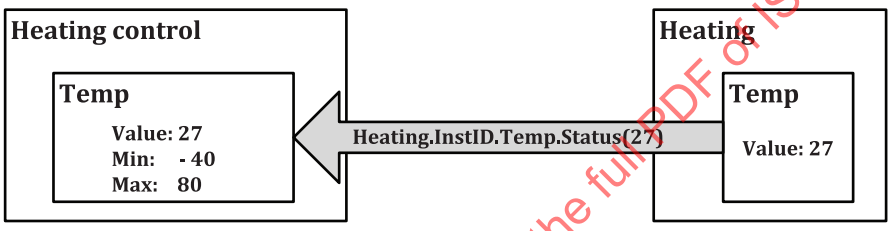


Figure 10 — Status report of a property (temperature setting of a heating)

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 `Status` 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 FktIDList is empty.
01_{16}	SetFunction	Entry is done for the following functions.
02_{16}	ClearAll	TargetAddress is deleted for all functions. The FktIDList is empty.
03_{16}	ClearFunction	TargetAddress 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 `Notification.Set`.	

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

REQ	8.23 APP – Notification mechanisms – Initial report for duplicate entry
If a node sends `Notification.Set` 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(`20_{16}`, `20_{16}`)`

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

`FBlock → Controller: FBlockID.InstID.Notification.Error(`20_{16}`, `21_{16}`)`

— `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(`20_{16}`, `10_{16}`, 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(`07_{16}`, `01_{16}`, 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` 41_{16} 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" (41_{16}). 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`.

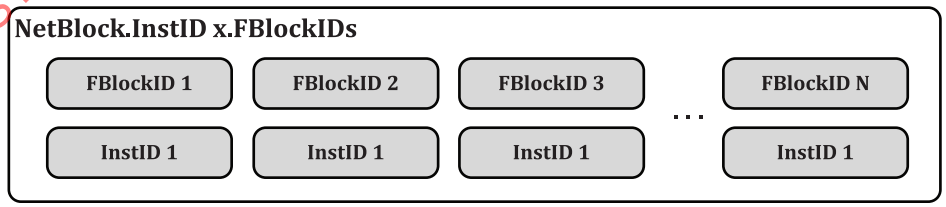


Figure 11 — Reading the FBlocks of a node from NetBlock FBlock

REQ	8.26 APP – Requesting FBlock information – NetBlock implements FBlockIDs
Every NetBlock FBlock shall implement the property `FBlockIDs` (000_{16}).	

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` (000_{16}).

It is read as follows:

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

Within an FBlock, `FktIDs` between 000_{16} and FFF_{16} (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` 000_{16} is available; the second bit is set to 1 if `FktID` 001_{16} 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 0_{16} represents a stuffing nibble.

Figure 13 shows how the information from `FBlockIDs` is used to request `FktIDs` from an FBlock.

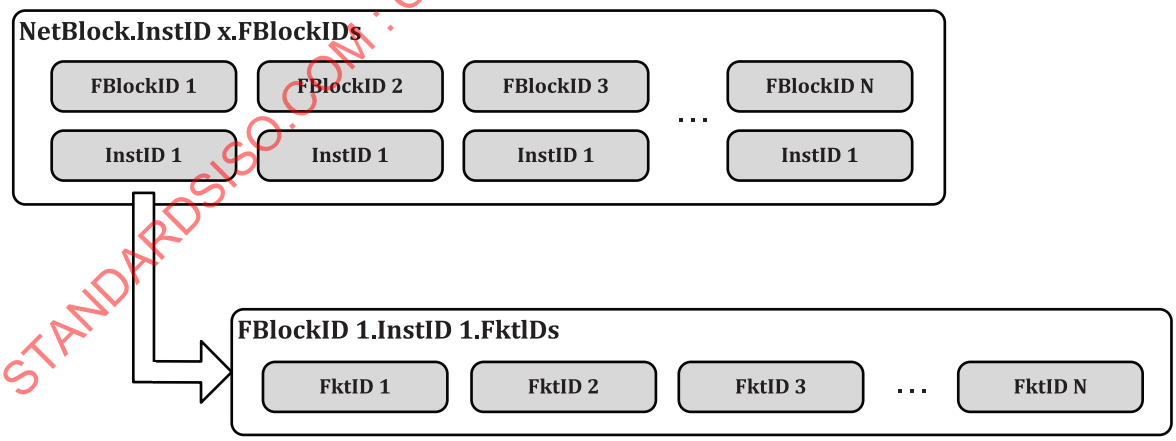


Figure 13 — Requesting the functions contained in an application FBlock

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 `FBlockInfo (FktID` 011_{16}`)`.	

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:

— 0_{16}: NotOK

— 1_{16}: OK

— 2_{16}: Invalid

— 3_{16}: Reserved

— 4_{16}: 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.

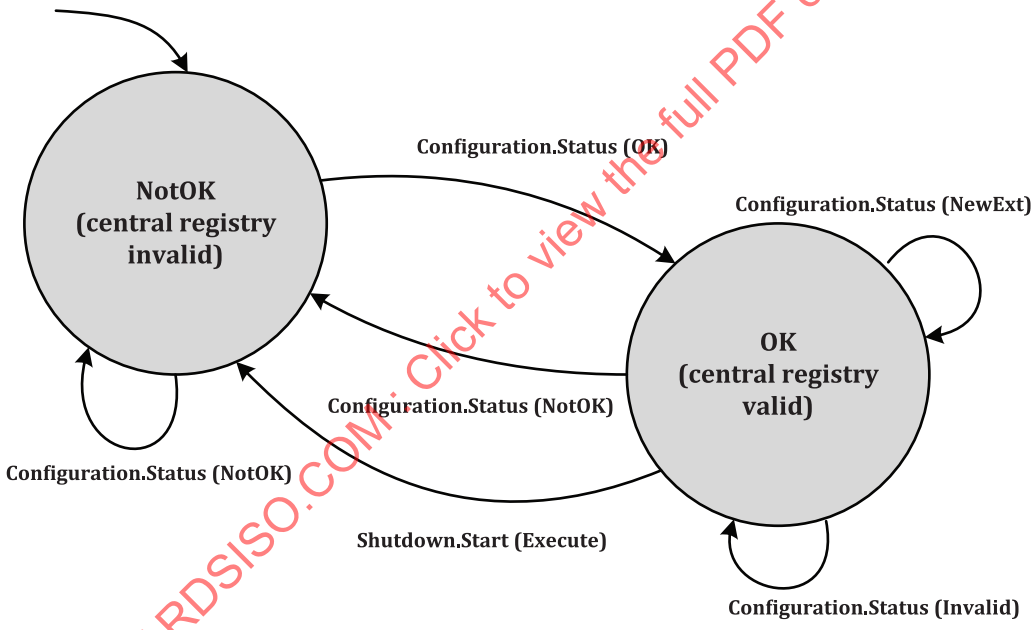


Figure 14 — States of the central registry in s_NetInterface_Normal_Operation

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 `Configuration.Status(NotOK)`, 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 `Shutdown.Start(Execute)` 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
`Configuration.Status(NotOK)`	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
`Configuration.Status(OK)`	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 `NetBlock.Shutdown.Start(Execute)`.	

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
`NetBlock.Shutdown.Start(Execute)`	Central registry state NotOK	— Shutdown by PowerMaster	— Network configuration reset (see 6.8.2) — Notify application
`Configuration.Status(NotOK)`	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
`Configuration.Status(NewExt)`	No transition	— New FBlocks are available	— Notify application
`Configuration.Status(Invalid)`	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 `Configuration.Status` 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 `NetworkMaster.Configuration.Status(NotOK)` while performing an FBlock scan, the NetworkMaster shall terminate the FBlock scan.	

REQ	8.36 APP – NetworkMaster – Start FBlock scan after NotOK
After sending `NetworkMaster.Configuration.Status(NotOK)` 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` 41_{16} 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` FF_{16} 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 `ev_Init_Ready` 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 `ev_Init_Ready` 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 `ev_Init_Ready` 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 `NetBlock.FBlockIDs.Get` 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 `FBlockIDs.Status` and no invalid announcements occurs (see 6.8.3.7 and 6.8.3.8), the NetworkMaster shall send `Configuration.Status(OK)` 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 `NetBlock.FBlockIDs.Status` messages in the order they are received.	

REQ	8.48 APP – NetworkMaster – InstID MSb evaluation
The NetworkMaster shall evaluate whether the MSb of the `InstID` of the NetBlock is set to 1 in the message `NetBlock.FBlockIDs.Status` (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 `NetBlock.FBlockIDs.Get`, the NetworkMaster shall set the most significant bit (MSb) of the `InstID` to 1.	

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

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 `InstID` (see 6.8.4.4.3) of all received `NetBlock.FBlockIDs.Status` 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 `NetBlock.FBlockIDs.Status` message where the MSb of the `InstID` 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 `Configuration.Status(NotOK)` 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.

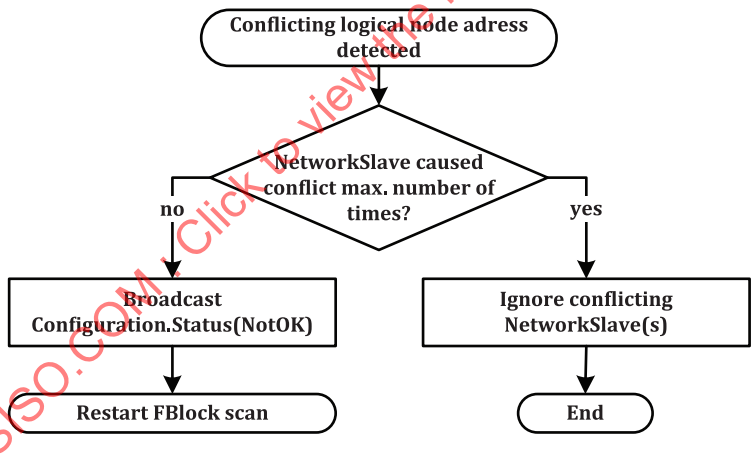


Figure 15 — NetworkMaster behaviour in the case of an address conflict

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.

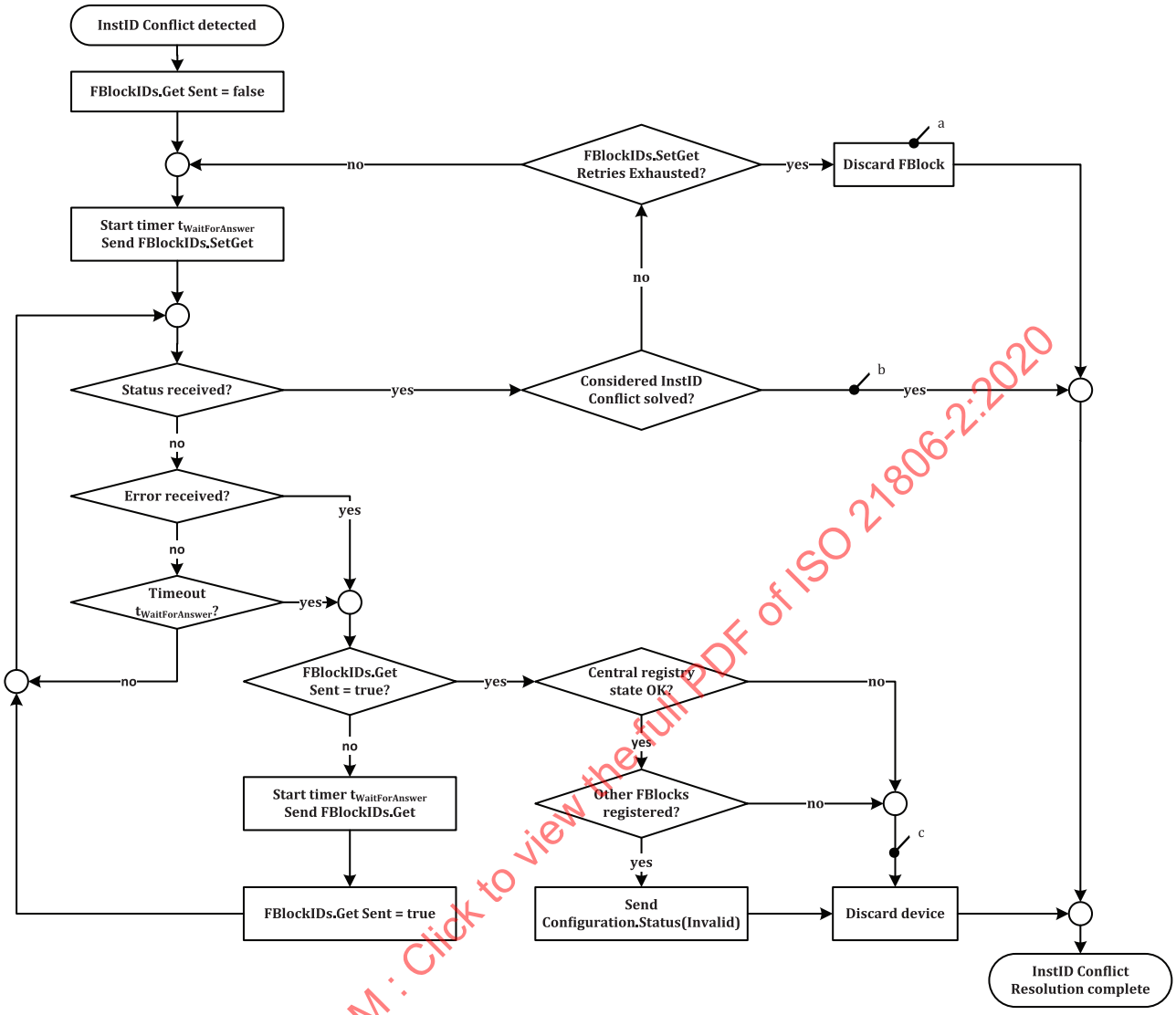
If a NetworkSlave does not accept a new InstID, it responds with an unchanged NetBlock.FBlockIDs.Status list.

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 `DeltaFBlockList` of `Configuration.Status` 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 `NetBlock.FBlockIDs.Status` 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: `Configuration.Status(Invalid, DeltaFBlockList).`	

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 03_{16} (ConnectionMaster) is removed from the central registry, the NetworkMaster shall send `Configuration.Status(NotOK)` 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 `OwnConfigInvalid` 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 `NetBlock.FBlockIDs.Status` 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: `Configuration.Status(NewExt, DeltaFBlockList).`	

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 `Configuration.Status(NewExt)` 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.

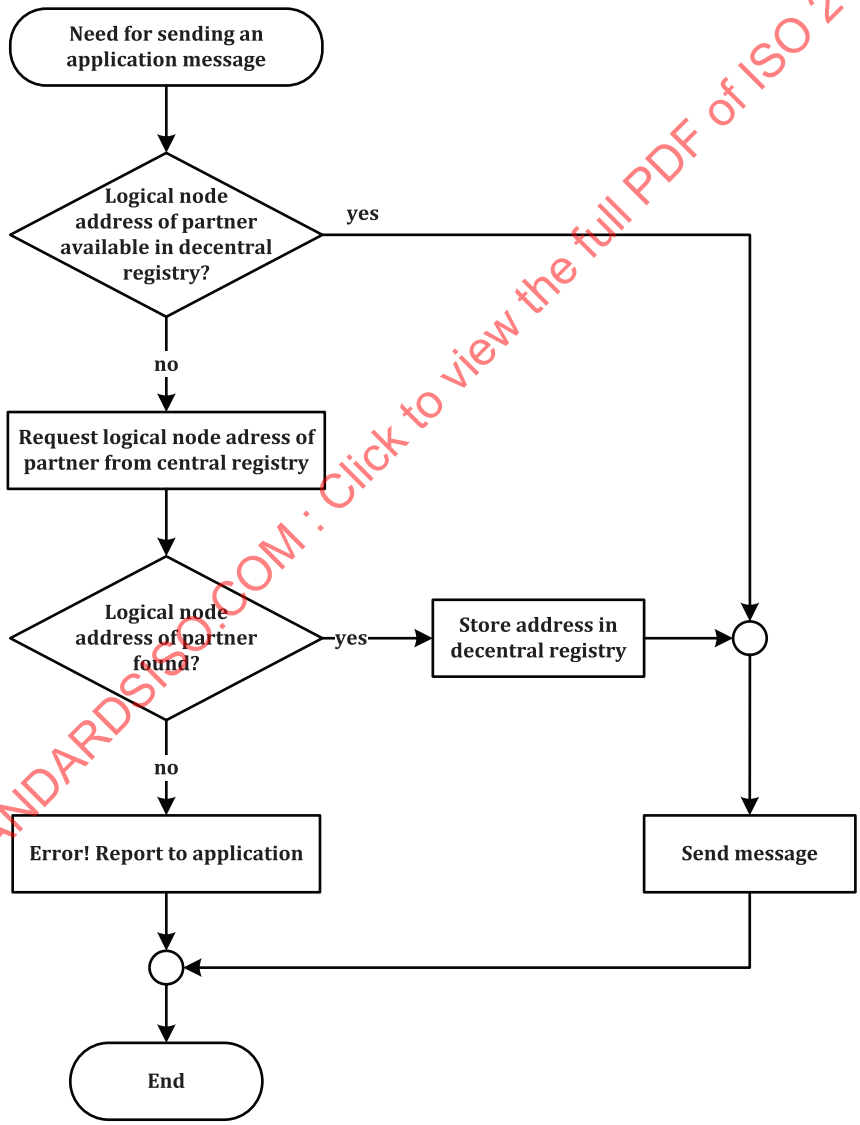


Figure 17 — Seeking the logical node address of a communication partner

6.8.4.2.2 APP — Building a decentral registry

The purpose of the entries in the decentral registry is to match the entries of the respective FBlock in the central registry.

The decentral registry can be built at any time after the central registry state is set to OK (e.g. when the application wants to control another node).

6.8.4.2.3 APP — Updating the decentral registry

REQ	8.73 APP – NetworkSlave – Updating the decentral registry
A NetworkSlave that maintains a decentral registry shall update it accordingly when the NetworkMaster distributes updates to the central registry.	

6.8.4.2.4 APP — Deleting the decentral registry

REQ	8.74 APP – NetworkSlave – Clearing the decentral registry
A NetworkSlave shall clear its decentral registry when the NetworkSlave enters or assumes central registry state NotOK.	

6.8.4.3 APP — Specific behaviour after ev_Init_Ready

6.8.4.3.1 APP — Initialisation

Following the `ev_Init_Ready` event, the NetworkSlave initialises its logical node address and services requests from the NetworkMaster.

A valid logical node address is any address within the dynamic or static address ranges as specified in 7.2.2.

6.8.4.3.2 APP — Valid logical node address not available

REQ	8.75 APP – NetworkSlave – No valid logical node address
If a NetworkSlave does not have a valid logical node address when the `ev_Init_Ready` event occurs, it shall set its logical node address to the calculated dynamic logical node address value (see 7.2.2).	

6.8.4.3.3 APP — Valid logical node address available

REQ	8.76 APP – NetworkSlave – Valid logical node address
If a NetworkSlave has a stored valid logical node address when the `ev_Init_Ready` event occurs, it shall use that logical node address.	

6.8.4.3.4 APP — Deriving the logical node address of the NetworkMaster

REQ	8.77 APP – NetworkSlave – Deriving NetworkMaster address
A NetworkSlave shall derive the logical node address of the NetworkMaster from the from the source address of the following messages: — `NetBlock.FBlockIDs.Get`, if the central registry state is NotOK; — `NetworkMaster.Configuration.Status(OK, NewExt, or Invalid)`.	

REQ	8.78 APP – NetworkSlave – Priorities for NetworkMaster address
If the address that a NetworkSlave derived from the `Configuration.Status` message differs from the address derived from `FBlockIDs.Get`, the NetworkSlave shall use the address that depends on `Configuration.Status`.	

REQ	8.79 APP – NetworkSlave – Triggers for setting NetworkMaster address
A NetworkSlave shall set the address of the NetworkMaster to $FFFF_{16}$ on the following events: — each transition to NetInterface state `s_NetInterface_Off`; — each transition to central registry state NotOK; — reception of `NetBlock.Shutdown.Start(Execute)`.	

6.8.4.4 APP — Normal operation of the NetworkSlave

6.8.4.4.1 APP — Behaviour in central registry state OK

A NetworkSlave may communicate freely while the central registry state is OK.

6.8.4.4.2 APP — Behaviour in central registry state NotOK

Functional addressing relies on a valid central registry, which is not present while the central registry state is NotOK. Communication that does not rely on a valid central registry uses, for example node position addressing or 48-bit addressing.

REQ	8.80 APP – NetworkSlave – No FBlock communication in NotOK
While the central registry state is NotOK, a NetworkSlave shall not initiate communication that is based on functional addressing.	

6.8.4.4.3 APP — Configuration requests and configuration changes

6.8.4.4.3.1 APP — General

To report its FBlock configuration, a NetworkSlave sends this message to the NetworkMaster (NWM):

`NetworkSlave → NWM: NetBlock.FBlockIDs.Status(FBlockIDList)`

FBlocks contained in `FBlockIDList` are available.

REQ	8.81 APP – NetworkSlave – Reported FBlocks
With the exception of `FBlockID` 00_{16}, in `NetBlock.FBlockIDs.Status`, a NetworkSlave shall report all FBlocks that are available for MOST communication except the following FBlocks, which shall not be reported: — `FBlockID` 01_{16} (NetBlock); — `FBlockID` 09_{16} (DebugMessages); — `FBlockID` $0A_{16}$ (ExtendedNetworkControl); — `FBlockID` $0F_{16}$ (EnhancedTestability); — FBlocks in the device/ECU supplier specific `FBlockID` range.	

REQ	8.82 APP – NetworkSlave – MNC FBlock not reported
A NetworkSlave that resides in a MOST node shall not report `FBlockID` 00_{16} (MNC) in `NetBlock.FBlockIDs.Status`.	

A NetworkSlave that resides in a remote-controlled node (see 6.2) may report `FBlockID` 00_{16} in `NetBlock.FBlockIDs.Status`.

REQ	8.83 APP – NetworkSlave – Value range for position dependent InstID
For FBlocks that are position dependent, a NetworkSlave shall not report an `InstID` value in the range of 40_{16} to FF_{16}.	

REQ	8.84 APP – NetworkSlave – Illegal values for position-independent InstID
For position-independent FBlocks, a NetworkSlave shall not report an `InstID` value of 00_{16} or FF_{16}.	

REQ	8.85 APP – NetworkSlave – Reporting empty FBlockIDList
If a NetworkSlave has no other FBlocks available for MOST communication than those that shall not be reported, when it sends `NetBlock.FBlockIDs.Status`, the `FBlockIDList` parameter shall be empty.	

REQ	8.86 APP – NetworkSlave – Withholding FBlock announcement
A NetworkSlave shall not announce its FBlocks again before having successfully transmitted its removal message to the node containing the NetworkMaster FBlock.	

REQ	8.87 APP – NetworkSlave – Conditions for setting InstID MSb
A NetworkSlave shall set the MSb of the `InstID` of the message `NetBlock.FBlockIDs.Status` to 1 if the MSb of the `InstID` in a previous `NetBlock.FBlockIDs.Get` is set since entering the NetInterface state `s_NetInterface_Normal_Operation` in the following cases: — in the first `FBlockIDs.Status` message after reaching NetInterface state `s_NetInterface_Normal_Operation`; — in a `NetBlock.FBlockIDs.Status` message with an empty `FBlockIDList`; — transmission of the preceding `NetBlock.FBlockIDs.Status` with MSb of the `InstID` set to 1 failed even after all low-level retries are exhausted.	

Setting the MSb of the `InstID` to 1 instructs the NetworkMaster to remove all of the NetworkSlave's FBlocks from the central registry.

REQ	8.88 APP – NetworkSlave – Ensuring FBlockIDs.Status reception
The NetworkSlave shall ensure that the `FBlockIDs.Status` message with the MSb of the `InstID` set to 1 is received by the NetworkMaster node.	

6.8.4.4.3.2 APP — Responding to Configuration Requests by the NetworkMaster

REQ	8.89 APP – NetworkSlave – Responding to NetworkMaster requests
A NetworkSlave shall respond to requests for FBlock configuration from the NetworkMaster at all times, regardless of the current central registry state.	

The NetworkMaster sends `NetBlock.FBlockIDs.SetGet(FBlockID, OldInstID, NewInstID)` to request a change from `FBlockID.OldInstID` to `FBlockID.NewInstID`.

REQ	8.90 APP – NetworkSlave – Failing to accept a new InstID
If the NetworkSlave fails to accept the new `InstID`, it shall send an unchanged list of FBlockIDs in `NetBlock.FBlockIDs.Status`.	

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 in the `OldInstID` parameter.

6.8.4.4.3.3 APP — Reporting Configuration Changes to the NetworkMaster

REQ	8.91 APP – NetworkSlave – Reporting changes
In central registry state OK, a NetworkSlave shall report changes of its FBlock configuration to the NetworkMaster.	

REQ	8.92 APP – NetworkSlave – No spontaneous reporting
In central registry state NotOK, if FBlockIDList is not empty, a NetworkSlave shall not send a `NetBlock.FBlockIDs.Status(FBlockIDList)` message without receiving a preceding `FBlockIDs.Get` request.	

6.8.4.4.4 APP — Unknown central registry state

REQ	8.93 APP – NetworkSlave – Wait for NetworkMaster after reset
After a reset of the MNC, a node shall not send control messages until it receives a message from the NetworkMaster.	

REQ	8.94 APP – NetworkSlave – Entering s_NetInterface_Normal_Operation
When a NetworkSlave enters NetInterface state `s_NetInterface_Normal_Operation`, it shall assume that the central registry state is NotOK.	

REQ	8.95 APP – NetworkSlave – Unknown central registry state means NotOK
If a NetworkSlave does not know the current central registry state, it shall assume that the central registry state is NotOK.	

6.8.4.4.5 APP — Determining the central registry state

REQ	8.96 APP – NetworkSlave – Determining the central registry state
NetworkSlave shall determine the current central registry state from the `NetworkMaster.Configuration.Status` message that is shown in Figure 18.	

This message is sent to the blocking broadcast address by the NetworkMaster FBlock.

The NetworkMaster sends `Configuration.Status(NotOK)` to announce that the central registry state is NotOK. All other `Configuration.Status` messages imply the central registry state is OK, for example:

— `Configuration.Status(OK);`

— `Configuration.Status(NewExt);`

— `Configuration.Status(Invalid);`

— `Configuration.Status(NewExt, <empty>).`

Figure 18 shows the NetworkSlave determining the central registry state in NetInterface state `s_NetInterface_Normal_Operation`.

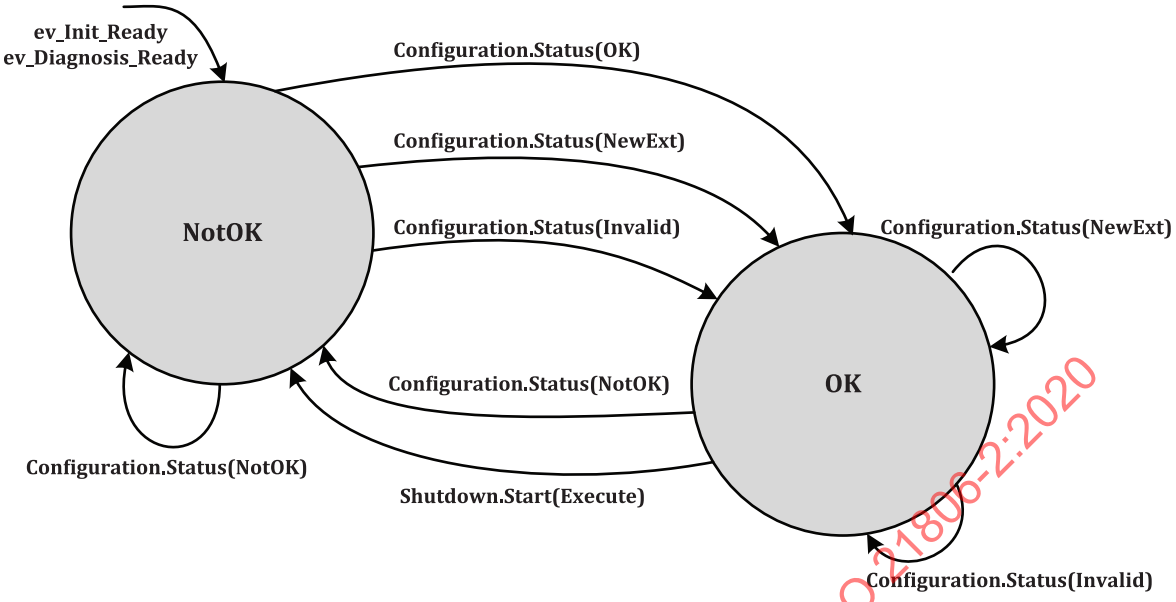


Figure 18 — NetworkSlave determines the central registry state in s_NetInterface_Normal_Operation

6.8.4.4.6 APP — Reaction to Configuration.Status(OK) when in central registry state NotOK

When the NetworkMaster sets the central registry state to OK the NetworkSlave

— builds a decentral registry when necessary, and

— initialises the application.

6.8.4.4.7 APP — Reaction to Configuration.Status(OK) when in central registry state OK

If a node receives `Configuration.Status(OK)` in central registry state OK, it ignores the message.

6.8.4.4.8 APP — Reaction to Configuration.Status(NotOK) when in central registry state NotOK

The NetworkMaster FBlock sends this message to reset all NetworkSlaves from a network point of view.

REQ	8.97 APP – NetworkSlave – Reaction on NotOK when in NotOK
When, in central registry state NotOK, a NetworkSlave receives `Configuration.Status(NotOK)`, it shall — clear any decentral registry, and — set the logical node address (see 7.2.2).	

The central registry state remains NotOK. For additional information, refer to 6.8.2.

6.8.4.4.9 APP — Reaction to Configuration.Status(NotOK) when in central registry state OK

REQ	8.98 APP – NetworkSlave – Reaction on NotOK when in OK
When, in central registry state OK, a NetworkSlave receives `Configuration.Status(NotOK)`, the NetworkSlave shall perform the actions that are described in 6.8.2.3.	

It services requests from the NetworkMaster while waiting for the central registry state to be set to OK. For additional information, refer to 6.8.2.

6.8.4.4.10 APP — Reaction to Configuration.Status(NewExt)

One or more FBlocks enter the MOST network and the central registry is updated with the FBlocks supplied in the message. These FBlocks should be added to the decentral registry, if they are used by the node.

This message is only sent in central registry state OK. The central registry state remains in OK. For additional information, refer to 6.8.2.

6.8.4.4.11 APP — Reaction to Configuration.Status(Invalid)

One or more FBlocks leave the MOST network and the FBlocks supplied in the message are removed from the central registry. If these FBlocks are contained in the decentral registry, they are removed.

REQ	8.99 APP – NetworkSlave – Own configuration invalid
If a node receives a `Configuration.Status(Invalid)` with (part of) its own `FBlockID`(s) and `InstID` ("own configuration invalid"), it shall reinitialise all applications and send an empty `FBlockID` list to the NetworkMaster.	

The following actions are taken before reinitialisation:

— A sink node secures its output and sets the `Status` parameter of the `Mute` property to `true`.

— A source node deallocates the bandwidth.

— All notifications are cleared.

If afterwards the MOST network remains in central registry state OK, the NetworkSlave announces its FBlocks, which prompts the NetworkMaster FBlock to send `Configuration.Status(NewExt)` to the blocking broadcast address.

6.8.4.4.12 APP — "Own configuration invalid" handling (optional)

During central registry state OK, when the NetworkMaster removes FBlocks of a NetworkSlave from the central registry, it sends `Configuration.Status(Invalid)` containing the list of FBlocks removed from the central registry to the NetworkSlave. The NetworkSlave identifies its own FBlocks and as a result sends `OwnConfigInvalid.StartResultAck(…, State=active, LogicalNodeAddress)` to the NetworkMaster FBlock; this starts the "own configuration invalid" handling. The NetworkSlave then sends an `FBlockIDs.Status` with an empty list.

The NetworkSlave cleans up streaming connections, clears the notification matrix, and reinitialises its FBlocks.

The NetworkSlave ignores all own FBlocks in `Configuration.Status(Invalid)` or `Configuration.Status(NewExt)` messages from the NetworkMaster FBlock until the "own configuration invalid" handling has completed. During the "own configuration invalid" handling, the NetworkSlave answers `FBlockIDs.Get` requests with an `FBlockIDs.Status` message that contains an empty list.

The end of the "own configuration invalid" state is signalled either by a transition to NetInterface state `s_NetInterface_Off`, `Configuration.Status(NotOK)`, or `OwnConfigInvalid.ResultAck(State = finished)`.

6.9 APP — Network wake-up, startup, and shutdown

6.9.1 APP — General

The wake-up, startup, and shutdown of the network or individual nodes is handled mainly by the PowerMaster. The PowerMaster uses NetBlock functions to communicate with PowerSlaves.

The PowerMaster should reside in the same node as the TimingMaster.

6.9.2 APP — Network wake-up and startup

In principle, the MOST network can be awakened by any node. The PowerMaster usually wakes the MOST network, for example, when there is communication on the car's bus or based on the status of the vehicle (clamp status).

It is up to the network owner to choose the preferred wake-up method. The process described here is independent of the wake-up method.

REQ	8.100 APP – Network wake-up, startup, and shutdown – Wake-up causes startup
A node that detects a qualified local wake-up event shall request `cmd_Network_Startup`.	

NOTE Requesting `cmd_Network_Startup` constitutes a network startup attempt.

REQ	8.101 APP – Network wake-up, startup, and shutdown – No spontaneous startup attempt
A node that does not detect a qualified local wake-up event shall not request `cmd_Network_Startup`.	

An application may use the electrical wake-up line to request that the PowerMaster node starts up the MOST network.

After a failed `cmd_Network_Startup` request, a node may perform another attempt. If the attempt violates timing restrictions, it might be blocked by the network layer.

The PowerMaster may continue attempting network startup as long as the condition to wake the MOST network is fulfilled.

REQ	8.102 APP – Network wake-up, startup, and shutdown – PowerSlave limited to 4 startup attempts
A node that does not contain the PowerMaster shall not perform more than 4 network startup attempts for a given qualified local wake-up event.	

The network owner may limit the startup attempts to less than 4.

6.9.3 APP — Network shutdown

6.9.3.1 APP — General

The MOST network is switched off either as the result of an error shutdown or by a normal shutdown performed by the PowerMaster. A detailed description regarding the shutdown event is included in ISO 21806-4[8].

6.9.3.2 APP — Error shutdown

A node switches off the MOST output when certain errors occur (e.g. critical unlock or low voltage).

6.9.3.3 APP — Emergency shutdown

To prevent an uncontrolled shutdown, an application may invoke an emergency shutdown by requesting `cmd_Emergency_Shutdown`.

6.9.3.4 APP — Normal shutdown

In a normal shutdown, the PowerMaster is responsible for switching off the MOST network.

The PowerMaster implements a shutdown procedure that has two stages. This procedure contains request and execution.

6.9.3.5 APP — Normal shutdown — Request stage

For the request stage, the PowerMaster sends `NetBlock.Shutdown.Start` with parameter `ControlCommand` set to `Query` to the blocking broadcast address.

A node without any further need for communication does not respond to `Shutdown.Start(Query)`.

A node that requires the availability of the MOST network responds with `Shutdown.Result(Suspend)`.

If the PowerMaster receives `Shutdown.Result(Suspend)` within time $t_{\text{WaitSuspend}}$, it postpones its attempt to switch off for time $t_{\text{RetryShutdown}}$, before retrying to shut down.

The PowerMaster may override suspend requests from its PowerSlaves and complete the shutdown, for example, to prevent the occurrence of critical conditions like low voltage.

If during $t_{\text{WaitSuspend}}$ or $t_{\text{RetryShutdown}}$ the shutdown reason ceases to exist, the PowerMaster does not send additional `Shutdown.Start(Query)` messages.

If no shutdown reason exists anymore, the PowerMaster cancels the shutdown procedure. For other nodes, the absence of `Shutdown.Start(Query)` messages indicates that the shutdown procedure is cancelled.

REQ	8.103 APP – Network wake-up, startup, and shutdown – Network activity disappears after query
If network activity disappears after `Shutdown.Start(Query)` is sent and the shutdown is not cancelled, the PowerMaster shall not wake the MOST network in order to finish its shutdown procedure.	

Network activity might disappear, for example, because of low voltage or critical unlock.

Figure 19 shows how a node suspends a shutdown request by the PowerMaster.

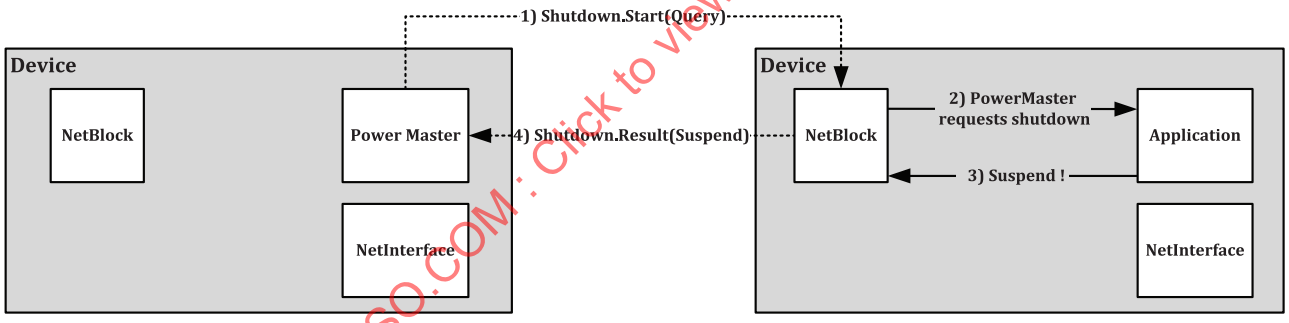


Figure 19 — Prevention of switching off MOST network via Shutdown.Result(Suspend)

6.9.3.6 APP — Normal shutdown — Execution stage

After sending `Shutdown.Start(Query)`, as shown in Figure 20, the PowerMaster waits for $t_{\text{WaitSuspend}}$ before it sends `Shutdown.Start(Execute)` to the blocking broadcast address to announce the shutdown of the MOST network. The shutdown process is started irrevocably.

The PowerSlaves do not reply to `Shutdown.Start(Execute)`. They prepare for shutting down (saving status) and then wait for network activity to end.

When $t_{\text{ShutdownWait}}$ expires, the PowerMaster hands `cmd_Off_Request` to the NetInterface.

Typically, the `cmd_Off_Request` is only generated by the application in the PowerMaster node.

If the network activity does not disappear within $t_{\text{SlaveShutdown}}$, a PowerSlave may switch off the MOST output by issuing `cmd_Off_Request`.

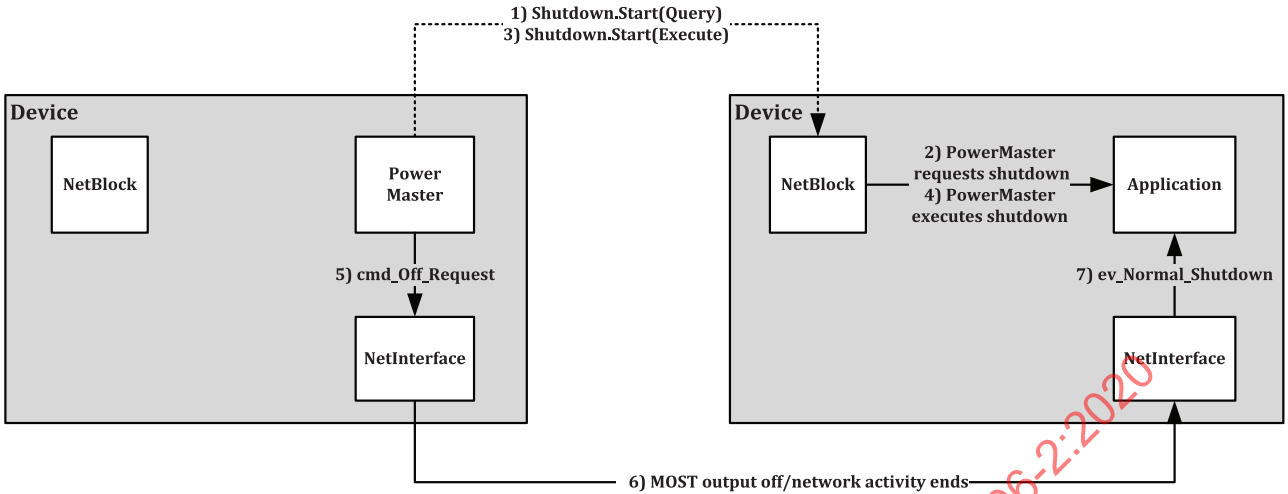


Figure 20 — Switching off MOST network by starting the Shutdown method in every NetBlock FBlock

6.9.4 APP — Device shutdown

6.9.4.1 APP — General

In order to minimise power consumption on system and device level, it can shut down specific applications in MOST nodes. This process is called device shutdown. Shutting down a node may affect the application on a system level. It is optional to support device shutdown.

When a node is shut down, all applications in the node may be shut down with the exception of the NetBlock FBlock, which is still active with limited functionality.

REQ	8.104 APP – Network wake-up, startup, and shutdown – Available NetBlock functions
In device shutdown state, a node shall provide the NetBlock properties `FBlockIDs`, `GroupAddress`, and `Shutdown`.	

Also, the node depends on determining the current central registry state when it exits device shutdown; therefore, it keeps track of the current central registry state that is distributed by the NetworkMaster even during device shutdown.

The `NetBlock.Shutdown.Start` message is used to bring a node into or out of device shutdown.

When managing device shutdown, this message may be sent to one node or a group of nodes, as opposed to network shutdown where this message is sent to the blocking broadcast address. The behaviour of a node during network shutdown is not affected by whether the node is in device shutdown or not. Refer to 6.9.3 for information about network shutdown.

In device shutdown state, the shutdown reason (see 6.11.2) cannot be stored by the application when the shutdown occurs. The shutdown reason, however, is retained in the NetInterface until the NetInterface is reset or `NetBlock.ShutdownReason.Set(`00_{16}`)` is called.

In the next transition to NetInterface state `s_NetInterface_Normal_Operation`, the application should use `cmd_Shutdown_Reason` to request the shutdown reason from the NetInterface and store it if the criteria in 6.11.2 are met.

6.9.4.2 APP — Performing device shutdown

6.9.4.2.1 APP — Device shutdown stages

The process of shutting down a node or a group of nodes can be divided into two stages, a request stage and an execution stage. The request stage is optional.

6.9.4.2.2 APP — Request stage (optional)

This stage guarantees that a node is not shut down while its FBlocks are communicating with FBlocks on other nodes. It is also useful if the PowerMaster wants to shut down a group of nodes but only if the whole group is ready.

a) The PowerMaster sends `NetBlock.Shutdown.Start(Query)` to a single node or a group of nodes.

b) The PowerMaster waits for $t_{\text{WaitSuspend}}$ to allow nodes to suspend the device shutdown.

c) A node that requires communication responds with `NetBlock.Shutdown.Result(Suspend)`.

d) If a node responds to the query, the PowerMaster waits for $t_{\text{RetryShutdown}}$ before trying again.

e) Steps 1 through 4 may be repeated. If no request to suspend the device shutdown process is received, $t_{\text{WaitSuspend}}$ expires and the execution stage is entered.

6.9.4.2.3 APP — Execution stage

To execute device shutdown, the PowerMaster starts the `NetBlock.Shutdown` method with parameter `ControlCommand` set to `DeviceShutdown` in a single node or in a group of nodes.

— The PowerMaster sends `NetBlock.Shutdown.Start(DeviceShutdown)`.

— A source node deallocates the bandwidth.

— A sink node secures any streaming outputs. The property `Mute`, which is specific to sinks, remains unchanged.

— The node removes itself from notification matrices in other nodes, if any.

— The node removes its FBlocks by sending an `NetBlock.FBlockIDs.Status` with an empty `FBlockIDList`.

— The NetworkMaster FBlock, using the blocking broadcast address, broadcasts the node's invalid FBlocks.

— The node may shut down its application but the NetBlock FBlock stays active.

6.9.4.3 APP — Exiting device shutdown

6.9.4.3.1 APP — General

The node can exit device shutdown due to the PowerMaster or due to a local condition.

6.9.4.3.2 APP — Exit due to PowerMaster

— The PowerMaster sends `NetBlock.Shutdown.Start(ExitDeviceShutdown)`.

— The node wakes its application.

— The node announces its own FBlocks using `NetBlock.FBlockIDs.Status(FBlockIDList)`.

— When the NetworkMaster reports the new FBlocks, they can be used.

6.9.4.3.3 APP — Exit due to local condition

— The part of the node deactivated due to device shutdown is activated.

— When the central registry state is OK or when explicitly asked by the NetworkMaster FBlock, the node announces its own FBlocks using `NetBlock.FBlockIDs.Status(FBlockIDList)`.

— When NetworkMaster reports the new FBlocks, they can be used.

A node that exits device shutdown performs the initialisation on application level (see 6.8.1.2.3).

6.9.4.4 APP — Persistence of device shutdown

The state of being in device shutdown is not memorized after a network restart.

REQ	8.105 APP – Network wake-up, startup, and shutdown – Persistence
If the PowerMaster application in a MOST network that supports device shutdown is reset in central registry state OK, the PowerMaster shall send `Shutdown.Start(ExitDeviceShutdown)` to the blocking broadcast address at the end of its reset procedure to establish a specified device shutdown state.	

6.9.4.5 APP — Response when device shutdown is unsupported

Since device shutdown is optional, the NetBlock FBlock of a node that does not have support for device shutdown responds to a request for device shutdown with `ErrorCode` 07_{16} (parameter not available).

6.10 APP — Connection management for streaming data

6.10.1 APP — Service for streaming data

6.10.1.1 APP — Streaming data

Figure 21 shows the streaming data categories.

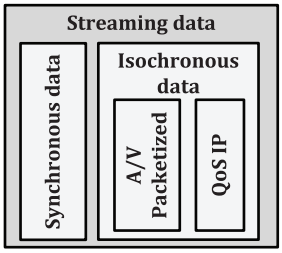


Figure 21 — Streaming data categories

Streaming data transfer is based on connections.

Streaming connections are administrated by a set of basic functions in sources and sinks.

For the application, a streaming connection is characterized by a 16-bit connection label and the required bandwidth, also a 16-bit value.

6.10.1.2 APP — Access to synchronous or isochronous data

Access to synchronous or isochronous data is provided by time division multiplexing (TDM).

Within a connection, streaming data is transmitted from a single source to one or more sinks.

Bandwidth is allocated when setting up streaming connections; therefore, it is guaranteed that in each MOST frame the source has access to the allocated number of bytes.

6.10.1.3 APP — Differentiating synchronous data and isochronous data

Synchronous data is streaming data that is transmitted at the rate of the MOST network clock.

Isochronous data is streaming data that is transmitted at a rate that is unrelated to the MOST network clock.

6.10.1.4 APP — Synchronous data

A synchronous data connection uses a fixed number of bytes in every MOST frame. This bandwidth is reserved during connection set-up.

6.10.1.5 APP — Isochronous data

An isochronous data connection allows the transmission of a variable number of bytes up to the reserved number of bytes in every MOST frame. The bandwidth to be reserved is based on the expected peak data rate.

The isochronous transmission class does not provide error detection (e.g. CRC). The receiving application should cope with frames corrupted during transmission.

Use cases for isochronous data include:

— A/V packetized isochronous (streaming): isochronous streams containing only data. Additional timebase information may be transported with the data. Typically, the data is arranged in packets. MPEG system layer streams (e.g. MPEG2-TS) fall into this category. No frame sync signal is included; nevertheless, the data contains multiple MPEG timestamps.

— QoS IP (streaming): this mode is used for packet transmission which allows for full quality of service requirements.

For more information on isochronous data, refer to the MOST stream transmission specification.

6.10.2 APP — Source and sink information

6.10.2.1 APP — StreamDataInfo function

An FBlock that contains sources or sinks implements the `StreamDataInfo` function. `StreamDataInfo` provides information on how many connections the FBlock may serve as source or as sink. To request the information, a controller sends this message:

```
Controller → FBlock: FBlockID.InstID.StreamDataInfo.Get
```

The answer contains a list of the source numbers and a list of the sink numbers in the FBlock:

```
FBlock → Controller: FBlockID.InstID.StreamDataInfo.Status(SourceNrList, SinkNrList)
```

6.10.2.2 APP — Source functions

6.10.2.2.1 APP — SourceInfo function

The property `SourceInfo` contains detailed information about the kind of data that the source provides. The source information is specific for each source number.

This message requests information about a source that is identified by `SourceNr`:

```
Controller → FBlock: FBlockID.InstID.SourceInfo.Get(SourceNr)
```

The source replies with this message:

```
FBlock → Controller: FBlockID.InstID.SourceInfo.Status(SourceNr, BlockWidth,
ConnectionLabel, TransmissionClass, ContentProtection,
ContentType, ContentDescription, TransmissionParameter)
```

`TransmissionClass` and the subsequent parameters determine the kind of data that can be processed by the source.

6.10.2.2.2 APP — SourceName function

The property `SourceName` provides the name of the streaming source.

This message requests the name of a source that is identified by `SourceNr`:

```
Controller → FBlock: FBlockID.InstID.SourceName.Get(SourceNr)
```

The answer is a string containing the name:

```
FBlock → Controller: FBlockID.InstID.SourceName.Status(SourceNr, SourceName)
```

6.10.2.2.3 APP — SourceActivity function

The optional `SourceActivity` controls the transmission of streaming data for each source. Based on the content of the parameter `Activity = [On/Off/Pause]`, streaming data transfer can be started, stopped, or paused:

```
Controller → FBlock: FBlockID.InstID.SourceActivity.StartResultAck
(SenderHandle, SourceNr, Activity)
```

When the requested action is complete, the following reply is sent:

```
FBlock → Controller: FBlockID.InstID.SourceActivity.ResultAck
(SenderHandle, SourceNr, Activity)
```

6.10.2.3 APP — Sink functions

6.10.2.3.1 APP — SinkInfo function

The property `SinkInfo` contains detailed information about the kind of data that the sink accepts. The sink information is specific for each sink number.

This message requests information about a sink that is identified by `SinkNr`:

```
Controller → FBlock: FBlockID.InstID.SinkInfo.Get(SinkNr)
```

The sink replies with this message:

```
FBlock → Controller: FBlockID.InstID.SinkInfo.Status(SinkNr,
BlockWidth,ConnectionLabel,
                                        TransmissionClass, ContentProtection,
                                            ContentType, ContentDescription,
                                                    TransmissionParameter)
```

`TransmissionClass` and the subsequent parameters determine the kind of data that can be processed by the sink.

6.10.2.3.2 APP — SinkName function

The property `SinkName` provides the name of the streaming sink. This message requests the name of a sink that is identified by `SinkNr`:

```
Controller → FBlock: FBlockID.InstID.SinkName.Get(SinkNr)
```

The answer is a string containing the name:

```
FBlock → Controller: FBlockID.InstID.SinkName.Status(SinkNr, SinkName)
```

6.10.3 APP — Streaming connections

6.10.3.1 APP — Source functions

6.10.3.1.1 APP — Allocate function

To request the allocation of bandwidth from a source, the `Allocate` method is used.

```
Controller → FBlock: FBlockID.InstID.Allocate.StartResultAck(SenderHandle, SourceNr)
```

If allocation is successful, the source reports the bandwidth and the corresponding connection label for the connection that it offers to sinks.

```
FBlock → Controller: FBlockID.InstID.Allocate.ResultAck(SenderHandle, SourceNr,
BlockWidth, ConnectionLabel)
```

REQ	8.106 APP – Streaming connections – Not enough bandwidth for allocation
If the unallocated bandwidth available to a source is less than the bandwidth requested in `Allocate.StartResultAck`, the source shall not allocate any bandwidth. The source shall reply with `ErrorCode` 20_{16} (function specific error); `ErrorInfo` shall contain the `SourceNr` and the required bandwidth: `FBlock → Controller: FBlockID.InstID.Allocate.ErrorAck(SenderHandle,` 20_{16}`, <SourceNr, BlockWidth>).`	

6.10.3.1.2 APP — DeAllocate function

`DeAllocate` is used to free previously allocated bandwidth.

```
Controller → FBlock: FBlockID.InstID.DeAllocate.StartResultAck(SenderHandle, SourceNr)
```

After successful execution, the connection is no longer offered to sinks; the source is disconnected.

```
FBlock → Controller: FBlockID.InstID.DeAllocate.ResultAck(SenderHandle, SourceNr)
```

6.10.3.2 APP — Sink functions

6.10.3.2.1 APP — Connect function

A controller uses `Connect` to link the sink to the connection that is identified by the connection label.

```
Controller → FBlock: FBlockID.InstID.Connect.StartResultAck (SenderHandle, SinkNr,
BlockWidth, ConnectionLabel)
```

If successful, the sink returns:

```
FBlock → Controller: FBlockID.InstID.Connect.ResultAck(SenderHandle, SinkNr)
```

6.10.3.2.2 APP — DisConnect function

The `DisConnect` method is used to remove the link to the existing connection.

```
Controller → FBlock: FBlockID.InstID.DisConnect.StartResultAck(SenderHandle, SinkNr)
```

When `DisConnect` finishes successfully, the following is reported:

```
FBlock → Controller: FBlockID.InstID.DisConnect.ResultAck(SenderHandle, SinkNr)
```

6.10.3.2.3 APP — Mute function

The output of streaming data from a sink can be stopped by using the property `Mute`.

```
Controller → FBlock: FBlockID.InstID.Mute.SetGet(SinkNr, Status)
```

`Status true` mutes. `Status false` turns mute off.

6.10.3.3 APP — Connection manager

6.10.3.3.1 APP — General

Streaming connections are managed by a connection manager.

REQ	8.107 APP – Streaming connections – Send requests to the connection manager
All requests for establishing connections shall be directed to the connection manager.	

REQ	8.108 APP – Streaming connections – Existence of the connection manager
The connection manager shall be implemented in every MOST network.	

The connection manager should reside in the same node as the TimingMaster.

The connection manager represents the functions specified in FBlock `ConnectionMaster` (03_{16}); it is not required that the connection manager implements that FBlock interface and is accessible by means of `FBlockID` 03_{16}.

REQ	8.109 APP – Streaming connections – ConnectionMaster FBlock
If the connection manager does provide the `FBlockID` 03_{16} interface, the NetworkMaster shall include the FBlock in the central registry.	

Table 10 contains the `ConnectionMaster` functions that are used in the administration of streaming connections.

Table 10 — Functions in FBlock ConnectionMaster used in the administration of streaming connection

Function	OPType	Sender	Receiver	Explanation
`BuildConnection`	`StartResultAck`	Controller	Connection manager	Request for building connection
	`ResultAck`	Connection manager	Controller	Answer with result
`ConnectionTable`	`Get`	Controller	Connection manager	Reading the property, where the connection manager stores all active point-to-point connections
	`Status`	Connection manager	Controller	Answer, containing the content of the property
`RemoveConnection`	`StartResultAck`	Controller	Connection manager	Request for removing connection
	`ResultAck`	Connection manager	Controller	Answer with result

For building a point-to-point connection, FBlock `ConnectionMaster` provides the `BuildConnection` method.

```
Controller → CManager: ConnectionMaster.1.BuildConnection.StartResultAck
(SenderHandle, Source, Sink)
```

`Source` in the method above refers to any source, identified by `FBlockID.InstID.SourceNr`.

`Sink` refers to any sink, identified by `FBlockID.InstID.SinkNr`.

CManager is the logical node address of the connection manager.

When a connection is built successfully, the FBlock `ConnectionMaster` returns:

```
CManager → Controller:  ConnectionMaster.1.BuildConnection.ResultAck
(SenderHandle, Source, Sink)
```

If setting up the connection fails, the `ConnectionMaster` answers with `OPType ErrorAck` and `ErrorCode` 42_{16} (processing error) and returns the parameters `Source` and `Sink`.

```
CManager → Controller:  ConnectionMaster.1.BuildConnection.ErrorAck
(SenderHandle, 42₁₆, <Source, Sink>)
```

For connection removal, the `RemoveConnection` method is used.

The FBlock `ConnectionMaster` generates an array of all existing connections including sources and sinks, where it adds more information. This array is accessible in the function `ConnectionTable`.

```
Controller → CManager:  ConnectionMaster.1.ConnectionTable.Get
CManager → Controller:  ConnectionMaster.1.ConnectionTable.Status(
<Source, Sink, BlockWidth, ConnectionLabel>,
<Source, Sink, BlockWidth, ConnectionLabel>,
<Source, Sink, BlockWidth, ConnectionLabel>,…)
```

The content of `ConnectionTable` cannot be set directly; building and removing connections is performed by the use of `BuildConnection` and `RemoveConnection` exclusively.

After switching off the MOST network, the content of `ConnectionTable` is deleted, leaving no streaming connections in the MOST network. Connections are rebuilt by new requests of the initiator(s).

REQ	8.110 APP – Streaming connections – Deleting ConnectionTable
The content of `ConnectionTable` shall be deleted when the connection manager receives `NetworkMaster.Configuration.Status(NotOK)`.	

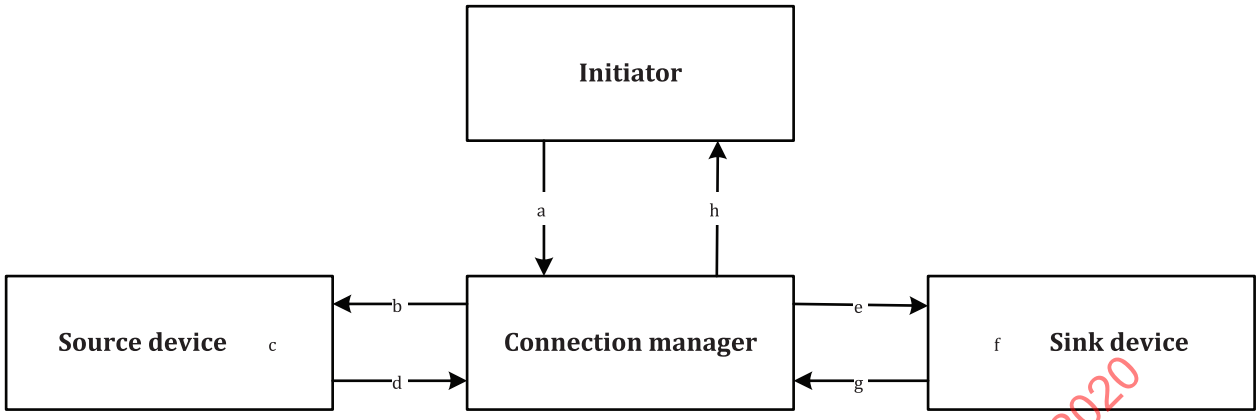
All connections are removed when `NetworkMaster.Configuration.Status(NotOK)` is received. See 6.8.2 for more information.

6.10.3.3.2 APP — Deadlock prevention

In order to prevent potential deadlocks in the connection building process, $t_{CM\_DeadlockPrev}$ is used. $t_{CM\_DeadlockPrev}$ is started as the connection manager sends the first request for building a connection to a source/sink FBlock. If the timer expires before the connection is created, the action is regarded as failed.

6.10.3.4 APP — Establishing streaming connections

When building a streaming connection, the connection manager uses the `Allocate` method in the source FBlock to connect it to the MOST network. After confirmation from the source, the connection manager uses the `Connect` method in the sink FBlock to link it to the connection used by the source. This mechanism is explained in Figure 22 and the text below.



a The `BuildConnection` method is started by an initiator, for building a connection between a source and a sink.

b Connection manager sends a command to the source node to connect its streaming output. The source supports the `Allocate` method.

c The source tries to allocate bandwidth. The following results are possible:

Enough free bandwidth. Reply to connection manager (Key d) as `Allocate.ResultAck` with parameters `SenderHandle`, `SourceNr`, `BlockWidth`, and `ConnectionLabel`.

Not enough bandwidth. Reply to connection manager (Key d) as `Allocate.ErrorAck` with parameters `SenderHandle`, `SourceNr`, and `BlockWidth`.

d The result is sent to the connection manager.

e If no error is reported, the connection manager starts the `Connect` method of the sink, communicating parameters `SenderHandle`, `ConnectionLabel`, and `BlockWidth`; then the sink has all information needed about the source.

f The sink connects.

g The result is sent to the connection manager as `Connect.ResultAck`.

h The connection manager reports the result of connection establishment to the initiator in `BuildConnection.ResultAck`. If successful, the connection manager internally stores the connection data. To connect another sink to this source, only the allocation data is sent to the new sink, as described in steps (Key e) to (Key g).

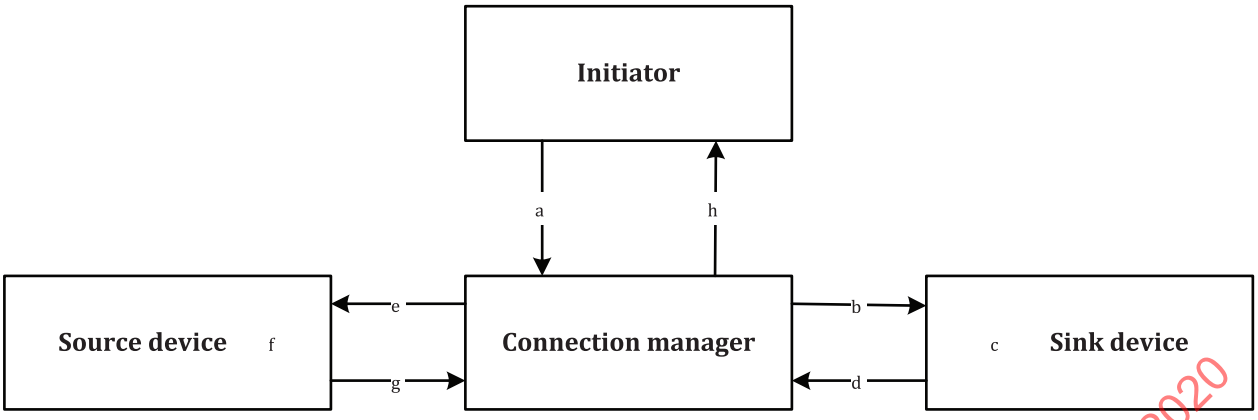
If the `Connect` process fails, the connection manager sends `BuildConnection.ErrorAck` and undoes all changes to the MOST network: The source is disconnected and if no other sink is already connected, the bandwidth is freed.

Figure 22 — Building a streaming connection step by step

6.10.3.5 APP — Removing streaming connections

In Figure 23, the initiator terminates a connection previously built by the connection manager. The connection manager commands the sink to disconnect from the connection that it is currently using.

After a positive answer, and if the connection is not in use by another sink, the connection manager disconnects the source, using `DeAllocate`. After that, the connection manager reports the result of the termination to the initiator.



a The `RemoveConnection` method is started by an initiator, for removing a connection between a source and a sink.

b The connection manager starts the `DisConnect` method in the sink FBlock, this makes the sink disconnect from the previously used connection.

c The sink disconnects.

d The sink reports the result to the connection manager by `DisConnect.ResultAck`.

e If no other sink uses the connection, the bandwidth in use is freed. Otherwise continue at step (Key h).

f The source deallocates and disconnects upon reception of `DeAllocate`.
The source attempts to deallocate the used bandwidth; the following result is expected:
Successful deallocation. Positive answer by `DeAllocate.ResultAck` (Key g).

g The result is sent to the connection manager as `DeAllocate.ResultAck`.

h If the connection manager has received a positive answer from the source, the connection is removed from its internal connection table. The connection manager sends an answer, `RemoveConnection.ResultAck`, to the initiator. The message contains the status of the requested termination of the connection.

Figure 23 — Step by step removal of a streaming connection

6.10.3.6 APP — Establishing DiscreteFrame isochronous streaming connections

DiscreteFrame isochronous streaming connections (containing data and additional timebase information in parallel) can be established by the use of `AllocateExt` and `ConnectExt`. Those are identical to Allocate and Connect apart from one additional parameter: `ClkSrcLabel`, which is the connection label of the clock source.

6.10.3.7 APP — Removing DiscreteFrame isochronous streaming connections

Termination of connections is performed by the use of `DeAllocate` and `DisConnect`.

REQ	8.111 APP – Streaming connections – Termination of a clock connection
The FBlock `ConnectionMaster` shall not terminate the clock connection before the last data connection is terminated.	

6.10.3.8 APP — Monitoring streaming connections

The availability of a connection is continuously verified by the data link layer. A source malfunction leads to automatic deallocation of the allocated bandwidth.

When a sink detects that one of its connections is no longer available, for example due to MOST network or source malfunction, this is referred to as source drop.

Every streaming sink is responsible for supervising the validity of its output (e.g. for an audio sink this is the output signal to the speakers).

REQ	8.112 APP – Streaming connections – Source drop reaction
A streaming data sink (e.g. amplifier) that detects a source drop shall secure its output signals.	

REQ	8.113 APP – Streaming connections – Muting after a source drop
A streaming data sink that detects a source drop shall set the `Status` parameter of the `Mute` property to `true`.	

REQ	8.114 APP – Streaming connections – Unlock reaction
A synchronous data sink (e.g. amplifier) that detects an unlock shall secure its output signals.	

REQ	8.115 APP – Streaming connections – Mute state after unlock
A synchronous data sink shall not change the `Mute` property due to an unlock.	

REQ	8.116 APP – Streaming connections – Invalid data reaction
If a sink application is able to verify the validity of the received streaming data and the data is rendered invalid, it shall secure its streaming output signals.	

When securing streaming outputs, sinks for synchronous streaming data, for example set the output volume to zero or display a test screen.

Additional error handling can be performed by the connection manager (e.g. cleaning up of spontaneously vanished connections).

6.10.3.9 APP — Handling of duplicate commands

6.10.3.9.1 APP — General

Duplicate commands to the streaming control command functions `Allocate`, `DeAllocate`, `Connect`, and `DisConnect` should not occur. This is in the responsibility of the connection manager. However, for the error case, the behaviour is specified in the following subclauses.

6.10.3.9.2 APP — Source behaviour

6.10.3.9.2.1 APP — Allocate

REQ	8.117 APP – Streaming connections – Duplicate Allocate request
If a source receives an `Allocate.StartResultAck` command with the source number of a currently allocated source, the source shall reply with `Allocate.ResultAck` and provide the connection label that is already assigned.	

6.10.3.9.2.2 APP — DeAllocate

REQ	8.118 APP – Streaming connections – DeAllocate for unallocated source
If a source receives a `DeAllocate.StartResultAck` command with a source number of a currently not allocated source, the source shall reply with `DeAllocate.ResultAck` and return the source number that was contained in the command.	

6.10.3.9.3 APP — Sink behaviour

6.10.3.9.3.1 APP — Connect

REQ	8.119 APP – Streaming connections – Duplicate Connect request
If a sink receives a `Connect.StartResultAck` command with the sink number of a currently connected sink and the same connection label that it is connected to, the sink shall reply with `Connect.ResultAck`.	

REQ	8.120 APP – Streaming connections – Switching a connection
To connect an already connected sink to a connection with a different label, the sink shall disconnect the existing connection and then establish the new connection. Then the sink shall send `Connect.ResultAck` to the requester.	

6.10.3.9.3.2 APP — DisConnect

REQ	8.121 APP – Streaming connections – DisConnect without connection
If a sink receives a `DisConnect.StartResultAck` command with the sink number of a currently not connected sink, the sink shall reply with `DisConnect.ResultAck` containing the requested sink number.	

6.10.4 APP — Compensating MOST network delay

The effective MOST network delay is negligible (delay is smaller than 1 µs per node). No MOST network delay compensation is necessary.

6.11 APP — Diagnosis

6.11.1 APP — FBlock configuration status information

Particularly when dealing with diagnostic mechanisms, it is important to have a reliable way of determining when the full FBlock configuration is reached. This information is, for example, necessary as the trigger for certain checks—such as the comparison of rated and actual configuration—and the precondition for diagnostic services.

This mechanism is optional.

Table 11 and Table 12 contain the structure of the `ImplFBlockIDs` function and the `ImplFBlockIDs` parameter, respectively.

Table 11 — ImplFBlockIDs function

FBlock	Function	OPType	Parameter
`NetBlock` (01_{16})	`ImplFBlockIDs` (012_{16})	`Get`	`---`
		`Status`	`ImplFBlockIDs`
		`Error`	`ErrorCode, ErrorInfo`

Table 12 — ImplFBlockIDs parameter

Basis datatype	Length	Condition	Description
Stream	---	---	Content: FBlockID, InstID {, FBlockID1, InstID1, FBlockID2, InstID2...}

All application FBlocks (i.e. not NetBlock, EnhancedTestability) implemented by the node are contained in this list with their `FBlockID` and `InstID`.

In the case of dynamic `InstIDs`, the `InstIDs` are numbered as reported by the node after first power on.

REQ	8.122 APP – Diagnosis – Error response
If a node that is not fully operable receives an `ImplFBlockIDs` command, the NetBlock FBlock shall return an error message with ErrorCode 41_{16} (function temporarily not available).	

REQ	8.123 APP – Diagnosis – Complete list
If an operable node receives an `ImplFBlockIDs.Get` command, it shall send `ImplFBlockIDs.Status`, containing all FBlock instances that are implemented.	

If the application is available and `ImplFBlockIDs` returns an empty list, the node does not implement application FBlocks.

REQ	8.124 APP – Diagnosis – Node position addressing
The NetworkMaster shall read `ImplFBlockIDs` using node position addressing.	

REQ	8.125 APP – Diagnosis – Trigger
The network owner shall specify the trigger to read the property `ImplFBlockIDs`.	

REQ	8.126 APP – Diagnosis – Not used in NotOK
The NetworkMaster shall not read the property `ImplFBlockIDs` in central registry state NotOK.	

REQ	8.127 APP – Diagnosis – Read once
After the trigger occurs, the NetworkMaster shall read the property `ImplFBlockIDs` of each node once.	

REQ	8.128 APP – Diagnosis – Read again
In addition, the NetworkMaster shall read the property `ImplFBlockIDs` after receiving an `FBlockIDs.Status` message from a node not replying to `ImplFBlockIDs.Get` at all or replying with `ImplFBlockIDs.Error(`41_{16}`)` before.	

A network owner may decide to delay that request according to the specific needs. For example, the request can be delayed until all NetworkSlaves have responded with a non-empty `FBlockIDs.Status`.

By comparing the list of implemented FBlocks with the list of FBlocks contained in the central registry, the NetworkMaster can determine the current state of the FBlock configuration.

REQ	8.129 APP – Diagnosis – SystemAvail
The NetworkMaster FBlock shall make the current state of the FBlock configuration available through the `SystemAvail` (`A10`$_{16}$) function, as shown in Table 13.	

Table 13 — NetworkMaster.SystemAvail function

FBlock	Function	OPType	Parameter
NetworkMaster (02_{16})	SystemAvail (`A10`$_{16}$)	`Get`	`---`
		`Status`	`DeviceAvail, FBlockAvail`
		`Error`	`ErrorCode, ErrorInfo`

Table 14 and Table 15 describe the `DeviceAvail` and `FBlockAvail` parameters, respectively.

Table 14 — SystemAvail.DeviceAvail parameter

Basis datatype	Range of values	Code	Description
Enum	00_{16} to 02_{16}	00_{16}	Incomplete: at least one node is not fully operable because of one of the following conditions: — All nodes have answered but at least one node has answered with the error 41_{16}. — The property `ImplFBlockIDs` is not read, for example, because the FBlock scan has detected an unsolvable address conflict or the FBlock scan is not finished.
		01_{16}	Complete: all nodes are fully operable: All available nodes have reported their implemented FBlocks through the property `ImplFBlockIDs`.
		02_{16}	Error: at least one node has not answered to `ImplFBlockIDs` or has returned an error not equal to 41_{16}.

Table 15 — SystemAvail.FBlockAvail parameter

Basis datatype	Range of values	Code	Description
Enum	00_{16} to 01_{16}	00_{16}	Incomplete: all other cases (not all FBlocks reported by the property `ImplFBlockIDs` of the available nodes are currently contained in the central registry or the state of `DeviceAvail` is still `Incomplete`).
		01_{16}	Complete: `DeviceAvail` is `Complete` and all FBlocks reported by `ImplFBlockIDs` are contained in the central registry.

6.11.2 APP — Shutdown reason

The application should store the shutdown reason to ensure that the information is still available after a reset.

When NetInterface state `s_NetInterface_Normal_Operation` is entered, the application should use `cmd_Shutdown_Reason` to request the shutdown reason from the NetInterface.

The application stores the shutdown reason obtained from the NetInterface if it differs from the currently stored shutdown reason and one of the following conditions applies:

— The application has `No_Result_Available` stored.

— The application has `No_Fault_Saved` stored.

— The application has `Critical_Unlock` stored and the NetInterface provides `Sudden_Signal_Off`.

In some cases, for example, in device shutdown state, the application may not be able to store the shutdown reason. To retain the shutdown reason even when the application was not available when the shutdown took place, in the next transition to NetInterface state `s_NetInterface_Normal_Operation`, the application uses `cmd_Shutdown_Reason` to request the shutdown reason from the NetInterface.

6.11.3 APP — Shutdown reason analysis

After the restart of the MOST network, a central component may query all nodes for stored faults. This query takes place when the central registry state is OK, i.e. after the NetworkMaster FBlock sends the `Configuration.Status(OK)` message. This unique controller sends the following single cast message to all nodes:

```
NetBlock.ShutdownReason.Get()
```

As it may happen that a node is not contained in `NetworkMaster.CentralRegistry.Status` (e.g. it only contains the NetBlock FBlock), `NetBlock.ShutdownReason.Get` is sent using node position addressing.

The nodes answer with the corresponding `Status` message or with error "FktID not available" if the application is not available yet. When it has gathered the information from all nodes, it sends `NetBlock.ShutdownReason.Set(00`$_{16}$`)` to all nodes. Only now all nodes delete the stored shutdown reason. This ensures that in the case of a reappearing fault the result of the analysis is not corrupted.

REQ	8.130 APP – Diagnosis – Error priority
If the TimingMaster detects an error, the unique controller shall ignore the shutdown reason in the TimingMaster if another node also reports the fault.	

REQ	8.131 APP – Diagnosis – ShutdownReason
Every NetBlock FBlock shall implement the `ShutdownReason` function.	

6.11.4 APP — Ring break diagnosis

6.11.4.1 APP — Starting ring break diagnosis

An external trigger is applied to the node to start the ring break diagnosis. The trigger event is system-specific.

The application uses `cmd_Start_Diagnosis` to request the start of ring break diagnosis.

REQ	8.132 APP – Diagnosis – Do not start RBD in s_NetInterface_Normal_Operation
`cmd_Start_Diagnosis` shall not be requested in `s_NetInterface_Normal_Operation`.	

6.11.4.2 APP — Ring break diagnosis result

The ring break diagnosis process sends `NetBlock.RBDResult.Status` to the blocking broadcast address to report the ring break diagnosis result. The message contains the status of the test (`ActivityAndLock/ActivityButNoLock/NoActivity`) and the diagnostic identifier of the node (`DiagID`).

The content of the `DiagID` is specified by the network owner.

The ring break diagnosis process reports the DiagResult to the application. The application may request the DiagResult from the NetInterface by `cmd_DiagResult`.

6.11.5 APP — Network diagnosis

Network diagnosis as described in this subclause is an optional feature.

Network diagnosis provides information about the status of the MOST network.

For network diagnosis, all MOST devices in the MOST network are connected by a diagnosis line.

REQ	8.133 APP – Diagnosis – Diagnosis master
The network owner shall designate one node in the MOST network as the diagnosis master.	

The diagnosis master collects and evaluates network diagnosis reports.

In network diagnosis, the following checks are performed:

— network activity detection;

— lock detection.

In Figure 24, the event “latch on” has two different meanings, depending on the use case:

— For network activity detection, it indicates network activity.

— For lock detection, it indicates locking onto the TimingMaster’s transmitted bit stream.

“Latch off” means the opposite:

— For network activity detection, it indicates the end of network activity.

— For lock detection, it indicates losing lock on the TimingMaster’s transmitted bit stream.

Figure 24 illustrates the network diagnosis states, which are provided by the NetInterface and which are valid for network activity detection as well as Lock detection. “Off” is the initial state.

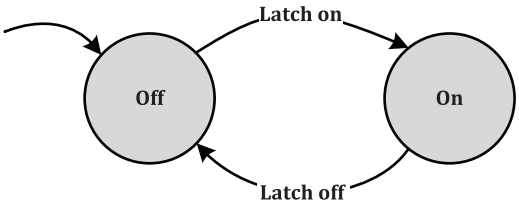


Figure 24 — Generic network diagnosis states

Network diagnosis is triggered by an external event (e.g. over electrical wake-up, signal on diagnosis line, or switch to power).

REQ	8.134 APP – Diagnosis – Network startup
If the MOST network is not running already when network diagnosis is triggered, the PowerMaster shall request `cmd_Network_Startup`.	

The network owner is responsible for specifying the timing definitions of the test so that the test result is obtained after the PowerMaster has initiated network startup by requesting `cmd_Network_Startup`.

REQ	8.135 APP – Diagnosis – Shutdown after completion
If the MOST network was started up to perform network diagnosis and no qualified local wake-up event or network wake-up event occurs, the PowerMaster shall shut down the MOST network when reporting the results on the diagnosis line has started.	

REQ	8.136 APP – Diagnosis – Querying the network diagnosis state
In the time frame that is specified by the network owner, each node shall query the current network diagnosis state and report “Off” or “On” over the diagnosis line.	

If network activity or lock — depending on the test — is identified, “On” is reported. If a ring break is present, “Off” is reported.

6.12 APP — Error handling

6.12.1 APP — General

The NetworkMaster and the PowerMaster handle errors pertaining to the entire MOST network, that is, errors that cannot be resolved locally.

6.12.2 APP — Handling overload in a message receiver

The MNC informs the MOST network service that the receiving node rejects a telegram after performing low-level retries (NAK error). This is an indicator for a momentary overload or a defect. The MOST

network service passes the NAK error through to the application, which decides what needs to be done (retry, reject, postpone).

6.12.3 APP — Over-temperature management

6.12.3.1 APP — General over-temperature behaviour

Some devices might experience malfunctions or permanent damage when exposed to temperature conditions above their operating limits. Even though it should be the design goal of every system that such condition is never reached during normal operation, it is still necessary to specify the system's behaviour for this worst case.

This subclause specifies the mandatory behaviour of every node that is able to monitor its own temperature.

In addition, optional behaviour of the PowerMaster and the PowerSlaves is described.

No central component in the MOST network supervises the temperature of a node and decides for the node when it should shut down; this is completely at the node's discretion.

Three different temperature levels are specified. From least severe to most severe, the levels are:

— restart temperature level ($\vartheta_{\text{Restart}}$);

— shutdown temperature level ($\vartheta_{\text{Shutdown}}$);

— critical temperature level ($\vartheta_{\text{Critical}}$).

All temperature levels are node-specific.

REQ	**8.137 APP – Error handling – Temperature levels**
$\vartheta_{\text{Shutdown}} \leq \vartheta_{\text{Critical}}$	

6.12.3.2 APP — Mandatory over-temperature behaviour

REQ	**8.138 APP – Error handling – Switching off the MOST output**
Above $\vartheta_{\text{Critical}}$, the overheated node shall send `cmd_Off_Request`.	

Additional node behaviour above $\vartheta_{\text{Critical}}$ is specified by the network owner.

Figure 25 shows the general mandatory over-temperature behaviour.

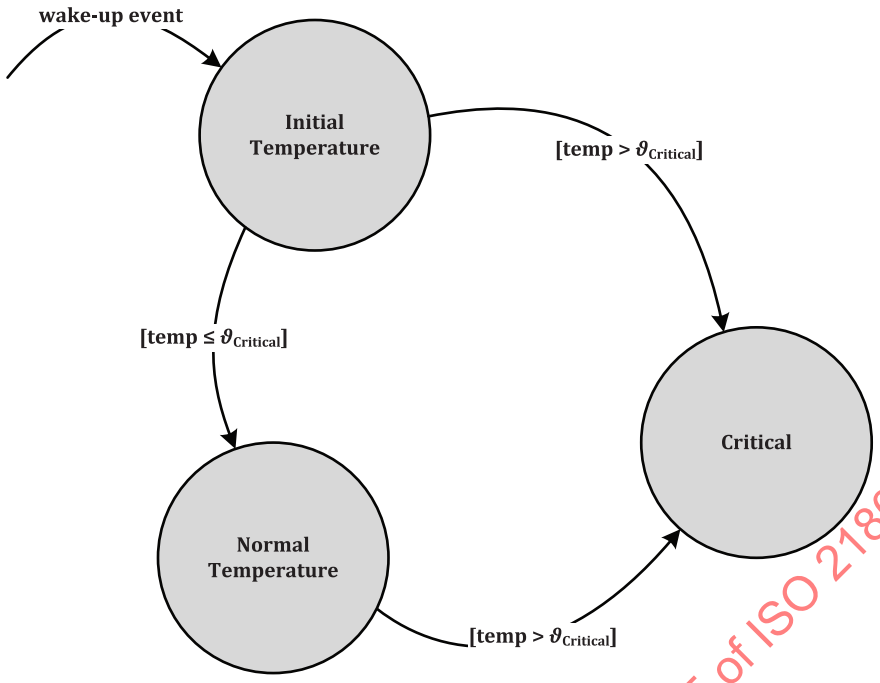


Figure 25 — Mandatory over-temperature behaviour

6.12.3.3 APP — Optional over-temperature behaviour

6.12.3.3.1 APP — Common over-temperature behaviour

The optional over-temperature behaviour may only be applied if the following conditions are met:

$\vartheta_{Critical} \neq \vartheta_{Shutdown}$ and $\vartheta_{Shutdown} \geq \vartheta_{Restart}$.

In `NetBlock.Shutdown.Result(Temperature)`, the symbolic name `Temperature` (value 03_{16} of parameter `ControlResult`) refers to a temperature shutdown.

Figure 26 shows the general optional over-temperature behaviour.

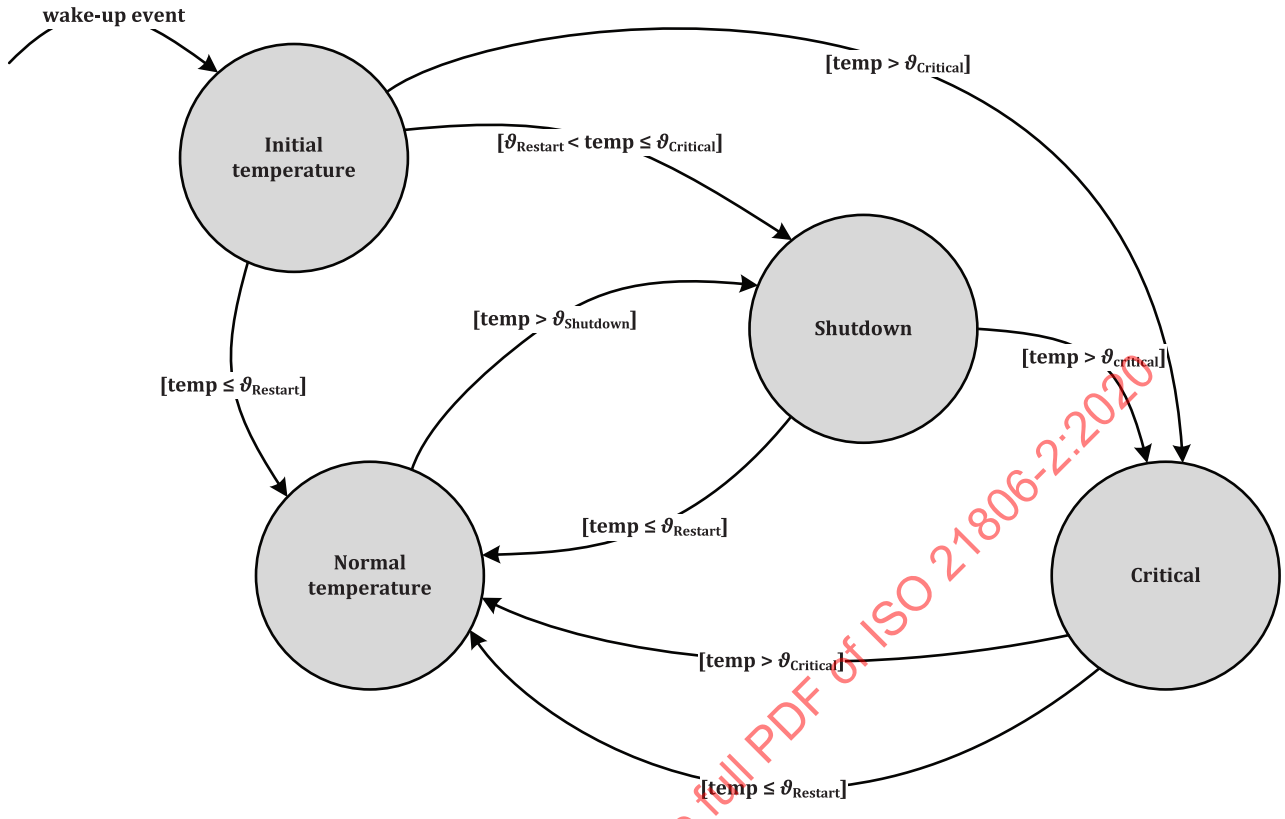


Figure 26 — Optional over-temperature behaviour

6.12.3.3.2 APP — Optional PowerMaster behaviour

When the PowerMaster receives `NetBlock.Shutdown.Result(Temperature)` from a PowerSlave, the PowerMaster recognizes that the PowerSlave is experiencing an over-temperature situation. In this case, a critical condition exists and therefore, the request stage of the shutdown is skipped.

REQ	8.139 APP – Error handling – MOST network shutdown
If the PowerMaster receives `NetBlock.Shutdown.Result(Temperature)`, it shall shut down the MOST network by performing the execution stage of the normal shutdown (see 6.9.3).	

6.12.3.3.3 APP — Optional PowerSlave behaviour

6.12.3.3.3.1 APP — Shutdown temperature level

REQ	8.140 APP – Error handling – Reporting over-temperature condition
If in NetInterface state `s_NetInterface_Normal_Operation` the temperature rises above $\vartheta_{Shutdown}$ for the first time, the overheated node shall send `NetBlock.Shutdown.Result(Temperature)` to the blocking broadcast address.	

6.12.3.3.3.2 APP — Restart temperature level

REQ	8.141 APP – Error handling – Over-temperature condition during restart
If during a restart attempt the temperature is above $\vartheta_{Restart}$, the overheated node shall send `NetBlock.Shutdown.Result(Temperature)` to the blocking broadcast address when entering NetInterface state `s_NetInterface_Normal_Operation`.	

6.12.3.3.3.3 APP — Restart

If the node is able to supervise its own cooling phase, it may start up the MOST network when it has cooled down.

6.13 APP — Timing definitions

6.13.1 APP — Definitions

6.13.1.1 APP — Exceptions to timing restrictions

Timers can be interrupted and as a potential result the corresponding timing restrictions are not met when a node executes a `cmd_Emergency_Shutdown` request, or transitions through reset or `s_NetInterface_Sleep`.

6.13.1.2 APP — Timers

Timers are initialised by setting the current timer value to 0. After a timer is started, it runs until it is paused, stopped, or it expires. When a timer is paused, the current timer value is retained. When a timer is stopped or expires, the current timer value is set to 0. When a timer is started, it runs from the current timer value.

When a timer expires, the specified action (e.g. error handling) is performed.

Whenever a timer definition contains the character "-" instead of a value, that particular value is not specified.

6.13.1.3 APP — Timing constraints

All timing constraints related to MOST messages are delimited by the presence of the corresponding telegrams on the MOST network.

Timing constraints represent an upper limit for a particular action.

6.13.2 APP — NetworkMaster communication

6.13.2.1 APP — Overview

REQ	8.142 APP – Timing definitions – Timing constraints
The values specified in Table 16 shall apply to the corresponding timing constraints.	

Table 16 — NetworkMaster communication timing constraints

Name	Value	Unit	Purpose
$t_{\text{WaitBeforeScan}}$	Network owner specific	ms	Limit for the NetworkMaster to start an FBlock scan
$t_{\text{ConfigurationAnnounce}}$	Network owner specific	ms	Limit for the NetworkMaster to set the central registry state

REQ	8.143 APP – Timing definitions – Timers
The values specified in Table 17 shall apply to the corresponding timers.	

Table 17 — NetworkMaster communication timers

Name	Minimum value	Typical value	Maximum value	Unit	Purpose
$t_{WaitForAnswer}$	Network owner specific	---	Network owner specific	ms	Time the NetworkMaster waits for all NetworkSlaves to respond
$t_{DelayCfgRequest1}$	500	500	700	ms	Time after which the NetworkMaster starts to query NetworkSlaves again
$t_{DelayCfgRequest2}$	10	10	11	s	Same as $t_{DelayCfgRequest1}$, but from the 21st attempt on

6.13.2.2 APP — Constraint $t_{WaitBeforeScan}$

6.13.2.2.1 APP — Purpose

The NetworkMaster constraint $t_{WaitBeforeScan}$ is the permissible duration between an `ev_Init_Ready` event, or broadcast of `NetworkMaster.Configuration.Status(NotOK)`, or an NCE and start of an FBlock scan by the NetworkMaster.

6.13.2.2.2 APP — Validity conditions

REQ	8.144 APP – Timing definitions – $t_{WaitBeforeScan}$ validity
$t_{WaitBeforeScan}$ shall become valid when the NetworkMaster detects an `ev_Init_Ready` event, broadcasts `NetworkMaster.Configuration.Status(NotOK)`, or detects an NCE.	

REQ	8.145 APP – Timing definitions – $t_{WaitBeforeScan}$ reset
If during the period that $t_{WaitBeforeScan}$ is valid any of the validity conditions appears or reappears, the permissible duration shall be reset.	

REQ	8.146 APP – Timing definitions – $t_{WaitBeforeScan}$ stops applying
$t_{WaitBeforeScan}$ shall stop applying when the NetworkMaster starts an FBlock scan.	

6.13.2.2.3 APP — Violation consequences

If the NetworkMaster does not start an FBlock scan within $t_{WaitBeforeScan}$, the central registry might not be up to date.

6.13.2.3 APP — Constraint $t_{ConfigurationAnnounce}$

6.13.2.3.1 APP — Purpose

The NetworkMaster constraint $t_{ConfigurationAnnounce}$ is the permissible duration between the start of the FBlock scan and setting the central registry state.

REQ	8.147 APP – Timing definitions – $t_{ConfigurationAnnounce}$ value
The value of $t_{ConfigurationAnnounce}$ shall be greater than the maximum value of $t_{WaitForAnswer}$.	

6.13.2.3.2 APP — Validity conditions

REQ	8.148 APP – Timing definitions – $t_{ConfigurationAnnounce}$ validity
$t_{ConfigurationAnnounce}$ shall become valid when the NetworkMaster begins its FBlock scan.	

REQ	8.149 APP – Timing definitions – $t_{ConfigurationAnnounce}$ stops applying due to FBlock scan termination
$t_{ConfigurationAnnounce}$ shall stop applying when the NetworkMaster terminates the FBlock scan.	

REQ	8.150 APP – Timing definitions – $t_{ConfigurationAnnounce}$ stops applying due to Configuration.Status
$t_{ConfigurationAnnounce}$ shall stop applying when the NetworkMaster sends `NetworkMaster.Configuration.Status`.	

6.13.2.3.3 APP — Violation consequences

If no error condition exists (see 6.8.3.7) and the NetworkMaster does not set the central registry state to OK within $t_{ConfigurationAnnounce}$, the central registry might remain in NotOK longer than necessary.

6.13.2.4 APP — Timer $t_{WaitForAnswer}$

6.13.2.4.1 APP — Purpose

The NetworkMaster timer $t_{WaitForAnswer}$ controls how long the NetworkMaster waits for the answer to `NetBlock.FBlockIDs` requests. $t_{WaitForAnswer}$ ensures that the central registry state is changed to OK even if not all nodes respond to the `FBlockIDs.Get` request by the NetworkMaster.

6.13.2.4.2 APP — Start and Stop conditions

REQ	8.151 APP – Timing definitions – $t_{WaitForAnswer}$ start
The NetworkMaster shall set the current timer value to 0 and start $t_{WaitForAnswer}$ when it sends `NetBlock.FBlockIDs.Get` or `NetBlock.FBlockIDs.SetGet`.	

REQ	8.152 APP – Timing definitions – $t_{WaitForAnswer}$ stop due to FBlock scan completion
The NetworkMaster shall stop $t_{WaitForAnswer}$ when it receives all expected responses to `NetBlock.FBlockIDs` requests.	

REQ	8.153 APP – Timing definitions – $t_{WaitForAnswer}$ stop due to FBlock scan termination
The NetworkMaster shall stop $t_{WaitForAnswer}$ when it terminates the FBlock scan.	

6.13.2.4.3 APP — Timer expiration

REQ	8.154 APP – Timing definitions – $t_{WaitForAnswer}$ expiration causes central registry update
When $t_{WaitForAnswer}$ expires, the NetworkMaster shall update the central registry.	

Subclause 6.8.3.9 describes the update procedure.

REQ	8.155 APP – Timing definitions – $t_{WaitForAnswer}$ expiration causes central registry state OK
If $t_{WaitForAnswer}$ expires in central registry state NotOK and no error condition exists (see 6.8.3.7), the NetworkMaster shall send `NetworkMaster.Configuration.Status(OK)` to the blocking broadcast address.	

When $t_{WaitForAnswer}$ expires, the NetworkMaster repeats the request after either $t_{DelayCfgRequest1}$ or $t_{DelayCfgRequest2}$ expires.

6.13.2.5 APP — Timer $t_{DelayCfgRequest1}$

6.13.2.5.1 APP — Purpose

$t_{DelayCfgRequest1}$ controls how long the NetworkMaster waits before it repeats a `NetBlock.FBlockIDs` request to NetworkSlaves answering with an error or not answering within $t_{WaitForAnswer}$.

This time is used after each of the first 20 queries since entering the NetInterface state `s_NetInterface_Normal_Operation`.

6.13.2.5.2 APP — Start and Stop conditions

REQ	8.156 APP – Timing definitions – $t_{DelayCfgRequest1}$ start due to $t_{WaitForAnswer}$
After $t_{WaitForAnswer}$ expires, if the $t_{DelayCfgRequest1}$ expiration counter value is less than 20 since entering the NetInterface state `s_NetInterface_Normal_Operation` and $t_{DelayCfgRequest1}$ is not already running, the NetworkMaster shall start $t_{DelayCfgRequest1}$ when it sends a `NetBlock.FBlockIDs` request to a NetworkSlave not answering a previous request.	

REQ	8.157 APP – Timing definitions – $t_{DelayCfgRequest1}$ start due to error
When the NetworkMaster receives `NetBlock.FBlockIDs.Error`, if the $t_{DelayCfgRequest1}$ expiration counter value is less than 20 since entering the NetInterface state `s_NetInterface_Normal_Operation` and $t_{DelayCfgRequest1}$ is not already running, the NetworkMaster shall start $t_{DelayCfgRequest1}$ when it sends a `NetBlock.FBlockIDs` request to a NetworkSlave not answering a previous request.	

REQ	8.158 APP – Timing definitions – $t_{DelayCfgRequest1}$ stop due to FBlock scan completion
The NetworkMaster shall stop $t_{DelayCfgRequest1}$ when it receives all responses from NetworkSlaves previously not responding.	

REQ	8.159 APP – Timing definitions – $t_{DelayCfgRequest1}$ stop due to FBlock scan termination
The NetworkMaster shall stop $t_{DelayCfgRequest1}$ when it terminates the FBlock scan.	

REQ	8.160 APP – Timing definitions – $t_{DelayCfgRequest1}$ counter reset
If the TimingMaster stops $t_{DelayCfgRequest1}$, it shall set the $t_{DelayCfgRequest1}$ expiration counter to 0.	

6.13.2.5.3 APP — Timer expiration

REQ	8.161 APP – Timing definitions – $t_{DelayCfgRequest1}$ expiration
When $t_{DelayCfgRequest1}$ expires, the NetworkMaster shall repeat its `NetBlock.FBlockIDs` requests to NetworkSlaves not answering a previous request or answering with `NetBlock.FBlockIDs.Error`.	

6.13.2.6 APP — Timer $t_{DelayCfgRequest2}$

6.13.2.6.1 APP — Purpose

$t_{DelayCfgRequest2}$ has the same purpose as $t_{DelayCfgRequest1}$, but from the 21st query on until all NetworkSlaves respond.

6.13.2.6.2 APP — Start and Stop conditions

REQ	8.162 APP – Timing definitions – $t_{DelayCfgRequest2}$ start due to $t_{WaitForAnswer}$
After $t_{WaitForAnswer}$ expires, if the $t_{DelayCfgRequest1}$ expiration counter value is 20 since entering the NetInterface state `s_NetInterface_Normal_Operation` and $t_{DelayCfgRequest2}$ is not already running, the NetworkMaster shall start $t_{DelayCfgRequest2}$ when it sends a `NetBlock.FBlockIDs` request to a NetworkSlave not answering a previous request.	

REQ	8.163 APP – Timing definitions – $t_{DelayCfgRequest2}$ due to error
When the NetworkMaster receives `NetBlock.FBlockIDs.Error`, if the $t_{DelayCfgRequest1}$ expiration counter value is 20 since entering the NetInterface state `s_NetInterface_Normal_Operation` and $t_{DelayCfgRequest2}$ is not already running, the NetworkMaster shall start $t_{DelayCfgRequest2}$ when it sends a `NetBlock.FBlockIDs` request to a NetworkSlave not answering a previous request.	

REQ	8.164 APP – Timing definitions – $t_{DelayCfgRequest2}$ stop due to FBlock scan completion
The NetworkMaster shall stop $t_{DelayCfgRequest2}$ when it receives all responses from NetworkSlaves previously not responding.	

REQ	8.165 APP – Timing definitions – $t_{DelayCfgRequest2}$ due to FBlock scan termination
The NetworkMaster shall stop $t_{DelayCfgRequest2}$ when it terminates the FBlock scan.	

REQ	8.166 APP – Timing definitions – $t_{DelayCfgRequest2}$ counter reset
If the TimingMaster stops $t_{DelayCfgRequest2}$, it shall set the $t_{DelayCfgRequest1}$ expiration counter value to 0.	

6.13.2.6.3 APP — Timer expiration

REQ	8.167 APP – Timing definitions – $t_{DelayCfgRequest2}$ expiration
When $t_{DelayCfgRequest2}$ expires, the NetworkMaster shall repeat its `NetBlock.FBlockIDs` requests to NetworkSlaves not answering a previous request or answering with `NetBlock.FBlockIDs.Error`.	

6.13.3 APP — PowerMaster communication

6.13.3.1 APP — Overview

REQ	8.168 APP – Timing definitions – Timers
The values specified in Table 18 shall apply to the corresponding timers.	

Table 18 — PowerMaster communication timers

Name	Minimum value	Typical value	Maximum value	Unit	Purpose
$t_{WaitSuspend}$	2 000	---	Network owner specific	ms	Time for receiving `Shutdown.Result(Suspend)` after sending `Shutdown.Start(Query)`.
$t_{ShutdownWait}$	1	2	15	s	Time between sending `Shutdown.Start(Execute)` and the `cmd_Off_Request`.
$t_{RetryShutdown}$	9,5	10	10,5	s	Time before sending the next `Shutdown.Start(Query)` message.
$t_{SlaveShutdown}$	16	---	Network owner specific	s	Time after which a PowerSlave may ignore network activity and send `cmd_Off_Request`.

6.13.3.2 APP — Timer $t_{WaitSuspend}$

6.13.3.2.1 APP — Purpose

$t_{WaitSuspend}$ controls how long the PowerMaster waits for a `NetBlock.Shutdown.Result(Suspend)` message after sending `NetBlock.Shutdown.Start(Query)`.

6.13.3.2.2 APP — Start and Stop conditions

REQ	8.169 APP – Timing definitions – $t_{WaitSuspend}$ start
The PowerMaster shall start $t_{WaitSuspend}$ when it sends `NetBlock.Shutdown.Start(Query)`.	

REQ	8.170 APP – Timing definitions – $t_{WaitSuspend}$ stop due to Suspend
The PowerMaster shall stop $t_{WaitSuspend}$ when it receives `NetBlock.Shutdown.Result(Suspend)`.	

REQ	8.171 APP – Timing definitions – $t_{WaitSuspend}$ stop due to shutdown reason inexistence
If the shutdown reason ceases to exist, the PowerMaster shall stop $t_{WaitSuspend}$.	

6.13.3.2.3 APP — Timer expiration

When $t_{WaitSuspend}$ expires, the PowerMaster sends `NetBlock.Shutdown.Start(Execute)`.

6.13.3.3 APP — Timer $t_{ShutdownWait}$

6.13.3.3.1 Purpose

$t_{ShutdownWait}$ controls how long the PowerMaster waits between broadcasting `NetBlock.Shutdown.Start(Execute)` and setting the shutdown flag.

6.13.3.3.2 APP — Start and Stop conditions

REQ	8.172 APP – Timing definitions – $t_{ShutdownWait}$ start
The PowerMaster shall start $t_{ShutdownWait}$ when it sends `NetBlock.Shutdown.Start(Execute)` to the blocking broadcast address.	

REQ	8.173 APP – Timing definitions – $t_{ShutdownWait}$ not stopped
The PowerMaster shall not stop $t_{ShutdownWait}$.	

6.13.3.3.3 APP — Timer expiration

REQ	8.174 APP – Timing definitions – $t_{ShutdownWait}$ expiration
When $t_{ShutdownWait}$ expires, the PowerMaster shall hand `cmd_Off_Request` to the NetInterface.	

6.13.3.4 APP — Timer $t_{RetryShutdown}$

6.13.3.4.1 APP — Purpose

$t_{RetryShutdown}$ controls how long the PowerMaster waits before sending `NetBlock.Shutdown.Start(Query)` again after receiving `NetBlock.Shutdown.Result(Suspend)`.

6.13.3.4.2 APP — Start and Stop conditions

REQ	8.175 APP – Timing definitions – $t_{RetryShutdown}$ start
If the PowerMaster receives `NetBlock.Shutdown.Result(Suspend)` while $t_{WaitSuspend}$ is running, it shall start $t_{RetryShutdown}$.	

REQ	8.176 APP – Timing definitions – $t_{RetryShutdown}$ stop due to shutdown reason inexistence
If the shutdown reason ceases to exist, the PowerMaster shall stop $t_{RetryShutdown}$.	

REQ	8.177 APP – Timing definitions – $t_{RetryShutdown}$ due to Execute
The PowerMaster shall stop $t_{RetryShutdown}$ when it sends `NetBlock.Shutdown.Start(Execute)`.	

6.13.3.4.3 APP — Timer expiration

REQ	8.178 APP – Timing definitions – $t_{RetryShutdown}$ expiration
When $t_{RetryShutdown}$ expires, the PowerMaster shall send `NetBlock.Shutdown.Start(Query)`.	

6.13.3.5 APP — Timer $t_{SlaveShutdown}$

6.13.3.5.1 APP — Purpose

$t_{SlaveShutdown}$ controls when a PowerSlave may switch off the MOST output after `NetBlock.Shutdown.Start(Execute)`, even if there still is network activity.

6.13.3.5.2 APP — Start and Stop conditions

REQ	8.179 APP – Timing definitions – $t_{SlaveShutdown}$ start
A PowerSlave shall start $t_{SlaveShutdown}$ when it receives `NetBlock.Shutdown.Start(Execute)`.	

REQ	8.180 APP – Timing definitions – $t_{SlaveShutdown}$ stop
A PowerSlave shall stop $t_{SlaveShutdown}$ when it switches off the MOST output.	

6.13.3.5.3 APP — Timer expiration

After $t_{SlaveShutdown}$ expires, a PowerSlave may switch off the MOST output regardless of network activity.

7 AL — Application layer

7.1 AL — Structure of MOST messages

The principal structure of messages on the application layer is:

```
Source address → Target address:
FBlockID.InstID.FktID.OPType(Parameter1, Parameter2,…)
```

On the network layer, the combined fields `FBlockID` (8-bit wide), `InstID` (8-bit wide), `FktID` (12-bit wide), and `OPType` (4-bit wide) are referred to as `MsgID` (message identifier).

The 16-bit source address field contains the logical node address (see 7.2.2) of the sender.

The 16-bit target address field contains a node address or a group address.

MOST messages rely on functional addressing. Therefore, the source address and the target address are not part of a MOST message.

7.2 AL — Addressing

7.2.1 AL — Overview

MOST addresses are divided into two categories:

— 16-bit addresses, and

— 48-bit addresses.

7.2.2 AL — 16-bit addressing

7.2.2.1 AL — Node position address

A node position address is used only for administrative tasks, for example, by the NetworkMaster during initialisation.

A node position address consists of an offset plus the node position:

— node position address = 0400_{16} + Pos;

— Pos = 00_{16} for the TimingMaster;

— Pos = 01_{16} for the first MOST node or remote-controlled node downstream of the TimingMaster.

7.2.2.2 AL — Logical node address

Logical node addressing is used to address a single node. A logical node address may be either dynamic or static. A dynamic address is calculated based on the node position address. A static address is predefined and is not changed when the NetworkMaster performs a network reset by setting the central registry state to NotOK.

REQ	7.1 AL – Addressing – Logical node address uniqueness
A logical node address shall be unique in the MOST network.	

7.2.2.3 AL — Dynamic logical node address

A dynamic logical node address is calculated as follows:

— dynamic logical node address = 0100_{16} + Pos.

The TimingMaster is located at node position 0; the resulting logical node address is 0100_{16}. The resulting address for a MOST node or remote-controlled node at position 5 in the MOST network is 0105_{16}.

7.2.2.4 AL — Static logical node address

A static logical node address is preconfigured, not calculated based on the node position. The distribution of static logical node addresses within a MOST network is determined by the network owner.

7.2.2.5 AL — Group address

The function `GroupAddress` in the NetBlock FBlock provides the group address. The default procedure for deriving a group address is to take the `FBlockID` of the FBlock that is most characteristic for the node. The high byte of the group address is always fixed to 03_{16} (see 7.2.2.8):

$$\texttt{GroupAddress} = 0300_{16} + \texttt{FBlockID}.$$

Groups can be built dynamically by modifying the group addresses.

Group addressing is typically used for controlling several nodes of the same type (e.g. active speakers). The grouping of nodes is determined when designing the MOST network.

7.2.2.6 AL — Blocking broadcast address

The blocking mechanism supports that all nodes receive important broadcast messages. Broadcast addressing requires system resources and therefore, should be used for administrative tasks only.

A single transfer that is sent to the blocking broadcast address ($03C8_{16}$), addresses all nodes in the MOST network.

REQ	7.2 AL – Addressing – No segmented broadcast
A node shall not send segmented transfers to the blocking broadcast address.	

7.2.2.7 AL — Non-blocking broadcast address

A non-blocking broadcast message (either single transfer or segmented transfer to address $03FF_{16}$) does not block the transmission of other control messages. This kind of transmission can be used for uncritical data transmission, which might not be received by some nodes.

7.2.2.8 AL — 16-bit address mapping

REQ	7.3 AL – Addressing – Address area structure
For 16-bit addressing, the address area of an MNC shall be structured as specified in Table 19.	

Table 19 — 16-bit address ranges

Address range	Description
0000_{16}	Internal communication. This address can also be used for administrative communication between MNCs on data link layer. Messages transmitted over the MOST network using 0000_{16} as target address are not handed over to the application layer.
0001_{16} to $000F_{16}$	Internal communication
0010_{16} to $00FF_{16}$	Static address range
0100_{16} to $013F_{16}$	Dynamic calculated (0100_{16} + Pos) address range
0140_{16} to $02FF_{16}$	Static address range
0300_{16} to $03FF_{16}$	Groupcast and broadcast
0400_{16} to $043F_{16}$	Node position (0400_{16} + Pos) address range
0440_{16} to $04FF_{16}$	Reserved
0500_{16} to $0EFF_{16}$	Static address range
$0F00_{16}$ to $0FEF_{16}$	Internal communication
$0FF0_{16}$	Debug address
$0FF1_{16}$ to $0FFF_{16}$	Internal communication
1000_{16} to $FFFE_{16}$	Reserved
$FFFF_{16}$	Un-initialised logical node address

7.2.3 AL — 48-bit addressing (Ethernet MAC address)

The permanent unique address is an EUI-48[10], conforming to the IEEE standard. The EUI-48[10] can be used as an Ethernet compatible MAC address. Each MAC address is unique and has a length of 48 bit.

7.3 AL — Function block identifier (FBlockID)

The `FBlockID` is the identifier of an FBlock.

Depending on the `FBlockID`, an FBlock contains a mandatory set of functions. It may contain additional functions.

The administrative FBlocks and the corresponding FBlockIDs are MOST network controller (00_{16}), NetBlock (01_{16}), NetworkMaster (02_{16}), ConnectionMaster (03_{16}), PowerMaster (04_{16}), Router

(08_{16}), DebugMessages (09_{16}), Bridge ($0B_{16}$), ExtendedNetworkControl ($0A_{16}$), Tool ($0E_{16}$), and EnhancedTestability ($0F_{16}$).

NOTE 1 System-specific proprietary FBlockIDs are used by network owners (e.g. vehicle manufacturers). They are specific for a system and are coordinated by the network owner between the device/ECU suppliers. A second kind of proprietary FBlockIDs is called device/ECU supplier specific. Those FBlockIDs are used by device/ECU suppliers for proprietary purpose.

REQ	7.4 AL – FBlockID – ISO 21806 range
FBlockIDs in the range of 00_{16} and $9F_{16}$ shall be reserved by this document. NOTE 2 FBlockID 00_{16} and FBlockID $0A_{16}$ are used for communication with the MNC.	

REQ	7.5 AL – FBlockID – MOST network controller FBlock count
Within a node, no instance or one instance of FBlockID 00_{16} shall exist.	

REQ	7.6 AL – FBlockID – MOST network controller FBlock access
Within a node, only the MOST network service shall address the local instance of FBlockID 00_{16}.	

REQ	7.7 AL – FBlockID – Identifer and name assignment
The FBlock identifiers shall be assigned according to Table 20.	

Table 20 — FBlockIDs

Kind	FBlockID	FBlock name	Description
Administration	00_{16}	MNC	MOST network controller FBlock
	01_{16}	NetBlock	NetBlock FBlock
	02_{16}	NetworkMaster	NetworkMaster FBlock
	03_{16}	ConnectionMaster	ConnectionMaster FBlock
	04_{16}	PowerMaster	PowerMaster FBlock
	05_{16}	Vehicle	Vehicle FBlock
	06_{16}	Diagnosis	Diagnosis FBlock
	08_{16}	Router	Router FBlock
	09_{16}	DebugMessages	Debug messages FBlock
	$0A_{16}$	ExtendedNetworkControl	Extended network control FBlock
	$0B_{16}$	Bridge	Bridge FBlock
	$0C_{16}$	RemoteControlCommunication	Remote control communication (RCC) FBlock
	$0E_{16}$	Tool	Tool FBlock
	$0F_{16}$	EnhancedTestability	Enhanced testability FBlock
Operation	10_{16}	HumanMachineInterface	Human machine FBlock
	11_{16}	SpeechRecognition	Speech recognition FBlock
	12_{16}	SpeechOutput	Speech output FBlock
	13_{16}	SpeechDatabase	Speech database FBlock

Table 20 *(continued)*

Kind	FBlockID	FBlock name	Description
Audio	20_{16}	AudioMaster	Audio master FBlock
	21_{16}	AudioDSP	Audio digital signal processor FBlock
	22_{16}	AudioAmplifier	Audio amplifier FBlock
	23_{16}	HeadphoneAmplifier	Headphone amplifier FBlock
	24_{16}	AuxiliaryInput	Auxiliary input FBlock
	25_{16}	AuxiliaryOutput	Auxiliary output FBlock
	26_{16}	MicrophoneInput	Microphone input FBlock
	28_{16}	Handsfree Processor	Handsfree processor FBlock
	29_{16}	AuxiliaryInputOutput	Auxiliary input and output FBlock
Drives	30_{16}	AudioTapeRecorder	Audio tape recorder FBlock
	31_{16}	AudioDiskPlayer	Audio disc player FBlock
	32_{16}	ROMDiskPlayer	ROM disc player FBlock
	33_{16}	MultimediaDiskPlayer	Multimedia disk player FBlock
	34_{16}	DVDVideoPlayer	DVD video player FBlock
Receiver	40_{16}	AMFMTuner	AM/FM tuner FBlock
	41_{16}	TMCTuner	Traffic message channel tuner FBlock
	42_{16}	TVTuner	Television tuner FBlock
	43_{16}	DABTuner	DAB tuner FBlock
	44_{16}	SatelliteRadio	Satellite radio FBlock
	45_{16}	TPEGTuner	TPEG tuner FBlock
	46_{16}	ETSISatelliteDigitalRadio	ETSI Satellite digital radio (ESDR) FBlock
Communication	50_{16}	Telephone	Telephone FBlock
	51_{16}	Phonebook	Phonebook FBlock
	52_{16}	NavigationSystem	Navigation system FBlock
	53_{16}	TMCDecoder	TMC decoder FBlock
	54_{16}	Bluetooth	Bluetooth FBlock
Video	60_{16}	Display	Display FBlock
	61_{16}	Camera	Camera FBlock
	62_{16}	VideoTapeRecorder	Video tape recorder FBlock
Content protection	70_{16}	DTCP	Digital transmission content protection (DTCP) FBlock
	71_{16}	HDCP	High-bandwidth digital content protection (HDCP) FBlock
Reserved	80_{16} to $9F_{16}$	Reserved	Reserved by this document (future use)
Proprietary	$A0_{16}$ to $C7_{16}$	System-specific	Reserved for network owner
	$C8_{16}$	Reserved	Reserved for conformance testing
	$C9_{16}$ to EF_{16}	System-specific	Reserved for network owner
	$F0_{16}$ to FE_{16}	Device/ECU supplier specific	Reserved for device/ECU supplier
	FF_{16}	Reserved	Reserved by this document (do not use)

7.4 AL — Instance identifier (InstID)

7.4.1 AL — Distinction of FBlock instances

There may be several equal[2)] FBlocks (instances) with the same FBlockID in the MOST network (multiple cameras, several diagnosis FBlocks, etc.). In order to address these FBlocks unambiguously, the FBlockID is complemented by an eight-bit instance identification number (InstID). The combination of FBlockID and InstID is referred to as the functional address.

7.4.2 AL — Uniqueness of functional addresses

Each node is responsible for the uniqueness of its functional addresses within the node. The NetworkMaster is responsible for the uniqueness of functional addresses within the entire MOST network (see 6.8.3).

7.4.3 AL — Assigning InstID

By default, every FBlock has InstID 01_{16}. In the case that there are several FBlocks of the same kind within one MOST node, the default numbering within the node starts at 01_{16} and is then incremented. In principle, as long as the InstID provides the possibility to differentiate between equal FBlocks, the InstID can be chosen freely. For example, the network owner may choose to use hard coded InstIDs or set the InstIDs depending on certain ranges with respect to the supported functions of the FBlock.

REQ	7.8 AL – InstID – Assignment
For FBlocks that are not position dependent, the values 00_{16} and FF_{16} shall not be used for InstID assignment.	

InstIDs of position-dependent FBlocks (e.g. NetBlock) contain the value of the node position. Positions are counted in the direction of the MOST output. Position counting starts with the TimingMaster, where any present position dependent FBlock is assigned InstID 00_{16}. With each following node, the position, and accordingly the InstID, is increased by one.

7.4.4 AL — InstID of NetBlock FBlock

REQ	7.9 AL – InstID – NetBlock InstID
The InstID of FBlock NetBlock shall be identical to the node position.	

7.4.5 AL — InstID of NetworkMaster FBlock

The InstID of the NetworkMaster FBlock may be zero; the default value is 01_{16}.

REQ	7.10 AL – InstID – Requests to the NetworkMaster
NetworkSlaves shall send requests to the NetworkMaster FBlock with the InstID set to 00_{16} (wildcard see 7.4.7).	

7.4.6 AL — InstID of FBlock EnhancedTestability

REQ	7.11 AL – InstID – EnhancedTestability
The InstID of FBlock EnhancedTestability shall be identical to the node position.	

2) The expression “equal” means that those FBlocks have the same functionality (e.g. two CD drives). This means that the basic functions are equal, but there is the possibility that they differ with respect to the total functionality (e.g. CD drive with or without random play).

7.4.7 AL — Wildcard values for InstID

`InstID` wildcards exist that are used when addressing FBlocks.

REQ	7.12 AL – InstID – Interpretation of InstID wildcards
The `InstID` wildcards shall be interpreted according to Table 21.	

Table 21 — InstID of FBlock Wildcards

InstID value	Description
00_{16}	Don't care (within a node). The node dispatches the message to one specific FBlock in the node.
FF_{16}	Broadcast (within a node). The message is dispatched to all instances of the matching FBlock.

For position-dependent FBlocks, the `InstID` value 00_{16} is interpreted differently in commands (wildcard) and in reports (position).

REQ	7.13 AL – InstID – Wildcard 00_{16} with logical node addresses
When using the logical node address to address an FBlock with a position dependent `InstID`, the wildcard 00_{16} shall be used. NOTE 1 The reason is that the actual `InstID` of the addressed FBlock cannot be foreseen.	

REQ	7.14 AL – Value with position dependence
When using the node position address to address an FBlock with a position dependent `InstID`, either the wildcard 00_{16} or the node position-dependent InstID value shall be used.	

REQ	7.15 AL – InstID – No FF_{16} with position dependence
An FBlock with a position dependent InstID shall not be addressed with `InstID` FF_{16}.	

REQ	7.16 AL – InstID – No InstID check with position dependence
An FBlock with a position dependent InstID shall not check commands for a valid `InstID` value.	

REQ	7.17 AL – InstID – "InstID not available" with position dependence
An FBlock with a position dependent InstID shall not send error messages with error code 02_{16} (`InstID` not available). NOTE 2 When sending a report, an FBlock only uses `InstID` wildcards in certain error situations that are in conjunction with an `InstID` value.	

REQ	7.18 AL – InstID – Report with correct InstID
If an FBlock is not in an error situation in conjunction with an `InstID` value, it shall use its correct `InstID` when sending reports.	

REQ	7.19 AL – InstID – InstID for "FBlock not available"
When any FBlock is addressed using `InstID` 00_{16}, but the FBlock is not implemented, the returned error message with `ErrorCode` 01_{16} (`FBlockID` not available) shall contain `InstID` 00_{16}.	

REQ	7.20 AL – InstID – InstID for "segmentation error"
The error message corresponding to a segmentation error (`ErrorCode` $0C_{16}$) shall have the same `InstID` as the original message. NOTE 3 This error is handled on the transport layer but not on the application layer, therefore the `InstID` is not evaluated.	

7.5 AL — Function identifier (FktID)

The FktID identifies a function. When transmitted on the MOST network, the FktID is 12-bit encoded, so that 4 096 different functions (methods and properties) can be addressed in an FBlock.

The address range of FktIDs is subdivided into the following ranges:

— Coordination (000_{16} to $1FF_{16}$): functions for administrative purposes in an FBlock.

— Application (200_{16} to BFF_{16}): functions in the application range represent the main functionality of an FBlock depending on the FBlock range, which are specified by this document, the network owner, or the device/ECU supplier.

— Network owner specific ($C00_{16}$ to EFF_{16}): functions that may be used by any network owner (car maker). These functions are coordinated by the network owner between the device/ECU suppliers.

— Device/ECU supplier specific ($F00_{16}$ to FFF_{16}): functions that can be used by device/ECU suppliers for any proprietary purpose:

  — Device/ECU supplier specific functions are not reported in <FBlockID>.FktIDs.Status.

  — If a device/ECU supplier specific property supports notification, notification of this property is not affected by the <FBlockID>.Notification.Set(SetAll) command.

  — If a device/ECU supplier-specific property supports notification, notification of this property is cleared by the <FBlockID>.Notification.Set(ClearAll) command.

Table 22 lists examples of predefined FktIDs for FBlocks that are not administrative FBlocks.

Table 22 — Example of predefined FktIDs in application FBlocks

FktID	Function name	Description
000_{16}	FktIDs	Reports the FktIDs of all functions contained in the FBlock (see 6.7.2).
001_{16}	Notification	Used to set or clear entries in the notification matrix (see 6.6).
002_{16}	NotificationCheck	Used to read the content of the notification matrix.

When developing FBlocks, FktIDs may be used freely in accordance with Table 23.

REQ	7.21 AL – FktID – Use GeneralFBlock for coordination range
If application FBlocks contain functions within the coordination range, those functions shall be based on the GeneralFBlock template.	

For each function, it can be determined if the use of the function is mandatory, optional, or depends on a condition specific to the application domain.

REQ	7.22 AL – FktID – FBlockID and FktID responsibilities
The responsibilities for FBlockID and FktID ranges shall apply as specified in Table 23.	

Table 23 regulates who is authorised to specify certain FBlockID/FktID combinations.

Table 23 — Responsibilities for FBlockID and FktID ranges

FBlockID	FktID			
	000_{16} to $1FF_{16}$ (Coordination)	200_{16} to BFF_{16} (Application)	$C00_{16}$ to EFF_{16} (Network owner specific)	$F00_{16}$ to FFF_{16} (Device/ECU supplier specific)
00_{16} to $0F_{16}$ (ISO)	ISO, in this document	ISO, in this document	Network owner	Network owner
10_{16} to $9F_{16}$ (ISO)	ISO, in this document	Network owner	Network owner	Device/ECU supplier
$A0_{16}$ to $C7_{16}$, $C9_{16}$ to EF_{16} (System-specific)	ISO, in this document	Network owner	Network owner	Device/ECU supplier
$F0_{16}$ to FE_{16} (Device/ECU supplier specific)	ISO, in this document	Device/ECU supplier	Device/ECU supplier	Device/ECU supplier
FF_{16}	Reserved			
$C8_{16}$	Reserved for conformance testing			

7.6 AL — Operation type (OPType)

7.6.1 AL — Overview

The OPType, as listed in Table 24, indicates which operation is performed on the property or method specified in `FktID`:

Table 24 — OPTypes for properties and methods

Category	OPType	For properties	For methods
Commands	0_{16}	`Set`	`Start`
	1_{16}	`Get`	`Abort`
	2_{16}	`SetGet`	`StartResult`
	3_{16}	`Increment`	Reserved
	4_{16}	`Decrement`	Reserved
	5_{16}	Not allowed	Not allowed
	6_{16}	Not allowed	`StartResultAck`
	7_{16}	Not allowed	`AbortAck`
	8_{16}	Not allowed	`StartAck`
Reports	9_{16}	`ErrorAck`[a]	`ErrorAck`
	A_{16}	Not allowed	`ProcessingAck`
	B_{16}	Reserved	`Processing`
	C_{16}	`Status`	`Result`
	D_{16}	Not allowed	`ResultAck`
	E_{16}	Not allowed	Not allowed
	F_{16}	`Error`	`Error`[b]

[a] `ErrorAck` is used for properties to report syntax errors in conjunction with illegal `OPTypes` (`ErrorCode` 01_{16} to 04_{16}).

[b] `OPType Error` may be used in conjunction with methods in cases where the `SenderHandle` parameter is not known or not applicable.

Commands and reports can be transmitted on the control channel or the packet channel.

REQ	7.23 AL – OPType – Channel selection
For every message, the network owner shall determine which channel is used.	

The `OPTypes Start`, `Abort`, `StartResult`, `Processing`, and `Result` are used in administrative FBlocks. For application FBlocks, the corresponding `OPTypes` with the `SenderHandle` parameter should be used instead.

An error message is a report with `OPType Error` or `ErrorAck`.

7.6.2 AL — Set, Get and SetGet

REQ	7.24 AL – Set, Get and SetGet – Writing a property
A controller shall use `OPType Set` or `SetGet` to write the content of a property.	

REQ	7.25 AL – Set, Get and SetGet – Setting a property
If an FBlock receives a message with `OPType Set` or `SetGet` to a property and no error is detected, it shall update the property according to the content of the message. NOTE 1 When using `Set`, the controller does not expect any reply (except error reports). If the syntax check is ok, the value of the property is updated.	

REQ	7.26 AL – Set, Get and SetGet – No reply to Set
An FBlock shall not reply to a `Set` command if no error is detected. NOTE 2 `OPType SetGet` is a combination of `Set` and `Get`, which means that successfully changing the value of a property results in a status message.	

For controllers that have no corresponding entries in the notification matrix for a particular property, `OPType SetGet` should be used to change the value of that property.

REQ	7.27 AL – Set, Get and SetGet – Reading a property
A controller shall use `OPType Get` to read a property. NOTE 3 If the controller does not receive a reply within $t_{\text{WaitForProperty}}$ after sending `Get` or `SetGet`, this is treated as if the controller receives any error message. NOTE 4 The maximum value for $t_{\text{WaitForProperty}}$ is specified by the network owner.	

7.6.3 AL — Increment and Decrement

`Increment` and `Decrement` are used to increase or decrease the value of a property. When using `Increment` or `Decrement`, the new status is reported to the triggering controller, as well as to the controllers that have corresponding entries in the notification matrix.

REQ	7.28 AL – Increment and Decrement – Incrementing and decrementing
If an FBlock receives a message with `OPType Increment` and `Decrement` to a property and no error is detected, it shall update the property according to the content of the message.	

REQ	7.29 AL – Increment and Decrement – Leaving valid range – No change
If `Increment` or `Decrement` result in leaving the range of valid values, the value shall remain unchanged.	

REQ	7.30 AL – Increment and Decrement – Leaving valid range – Reporting
If `Increment` or `Decrement` result in leaving the range of valid values, the unchanged value shall be reported. NOTE This case is not treated as an error and therefore, no error message with `ErrorCode` 06_{16} (parameter out of range) is sent. The report is directed to the triggering controller only. Because the value remains unchanged, no message due to notification is sent to other controllers.	

The `OPTypes Increment` and `Decrement` should not be used in conjunction with multicast messages.

7.6.4 AL — Status

REQ	7.31 AL – Status – Status reply
An FBlock shall reply with an `OPType Status` to a `Get`, `SetGet`, `Increment`, or `Decrement` command if no error is detected.	

REQ	7.32 AL – Status – Reply within $t_{Property}$
If a property receives a `Get` or `SetGet` command, the transmission of the corresponding `Status` or error message shall be started within $t_{Property}$ after completely receiving the command. NOTE If a controller has a corresponding entry in the notification matrix, using `SetGet`, `Increment` or `Decrement` might result in additional `Status` messages.	

7.6.5 AL — Start and StartAck

REQ	7.33 AL – Start and StartAck – Sending Start or StartAck
A controller shall use the `OPType Start` or `StartAck` to start execution of a method that is not expected to return a result.	

REQ	7.34 AL – Start and StartAck – Receiving Start or StartAck
If an FBlock receives a message with `OPType Start` or `StartAck` to a method and no error is detected, it shall start execution of the method according to the content of the message. NOTE Figure 27 illustrates the possible reactions of the FBlock. If an FBlock receives a message with the `OPType Start` or `StartAck` that contains syntax or parameter errors, it replies with an error message. The FBlock does not start the corresponding method.	

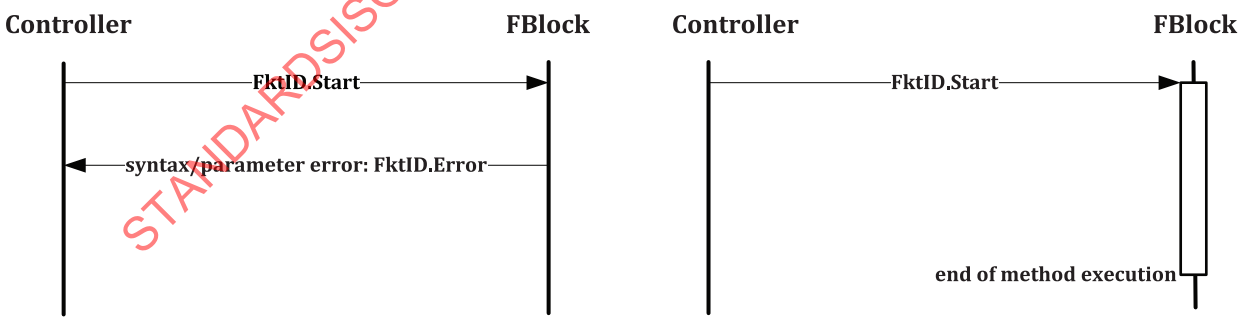


Figure 27 — Sequences when using FktID.Start with and without error

7.6.6 AL — StartResult and StartResultAck

REQ	7.35 AL – StartResult and StartResultAck – Sending StartResult or StartResultAck
A controller shall use the `OPType StartResult` or `StartResultAck` to start execution of a method that is expected to return a result.	

REQ	7.36 AL – StartResult and StartResultAck – Receiving StartResult or StartResultAck
If an FBlock receives a message with `OPType StartResult` or `StartResultAck` to a method and no error is detected, it shall start execution of the method according to the content of the message.	

7.6.7 AL — Result and ResultAck

REQ	7.37 AL – Result and ResultAck – Reporting with Result or ResultAck
If an FBlock receives a message with `OPType StartResult` or `StartResultAck` to a method, the FBlock shall provide the result with the `OPType Result` or `ResultAck`, respectively, or report failure with an error message.	

REQ	7.38 AL – Result and ResultAck – Waiting for StartResult or StartResultAck
If a controller starts a method with the `OPType StartResult` or `StartResultAck` and does not receive a report with the `OPType Result` or `ResultAck`, respectively, or an error message within the time specified by the network owner, it shall behave as if receiving an error message.	

Figure 28 shows an example how the reception of the `OPType StartResultAck` is handled.

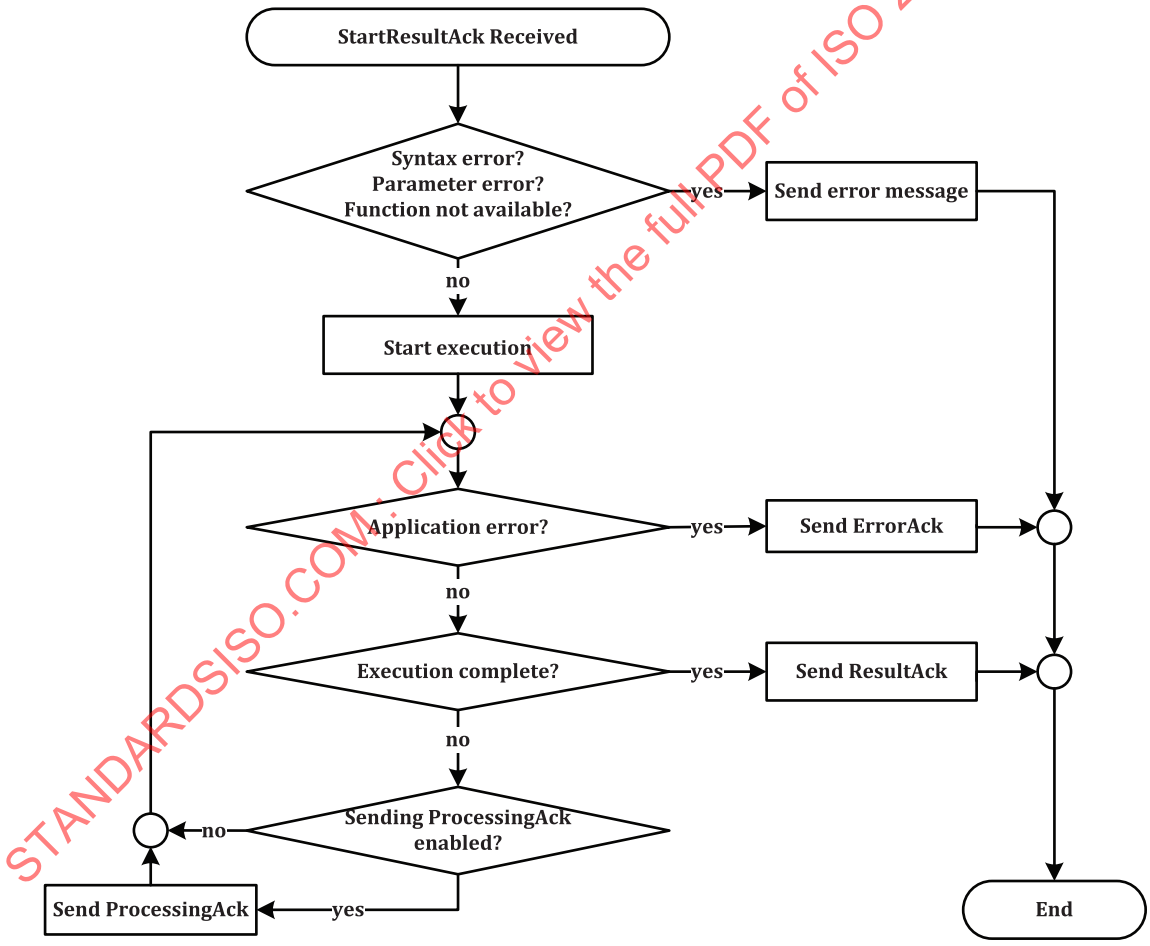


Figure 28 — Example flow for handling communication of methods (FBlock side)

7.6.8 AL — Processing and ProcessingAck

While an FBlock executes a method started with the `OPType StartResult` or `StartResultAck`, it may send reports with the `OPType Processing` or `ProcessingAck`, respectively.

7.6.9 AL — Abort and AbortAck

REQ	7.39 AL – Abort and AbortAck – Abort terminates Start or StartResult
A controller shall use the `OPType Abort` to terminate execution of a method started with `Start` or `StartResult`.	

REQ	7.40 AL – Abort and AbortAck – AbortAck terminates StartAck or StartResultAck
A controller shall use the `OPType AbortAck` to terminate execution of a method started with `StartAck` or `StartResultAck`.	

REQ	7.41 AL – Abort and AbortAck – No parameters for Abort
The `OPType Abort` shall not have any parameters.	

REQ	7.42 AL – Abort and AbortAck – One parameter for AbortAck
The `OPType AbortAck` shall not have any parameters except `SenderHandle`.	

A method should be aborted only by the controller starting it.

7.6.10 AL — Error and ErrorAck

REQ	7.43 AL – Error and ErrorAck – Send error message to report failure
An FBlock shall send an error message if the action that corresponds to a command fails.	

REQ	7.44 AL – Error and ErrorAck – Reply with error to erroneous command
If an FBlock receives a command that contains an error, it shall reply with an error message.	

REQ	7.45 AL – Error and ErrorAck – No method execution for erroneous commands
If an FBlock receives a message with the `OPType Start, StartAck, StartResult` or `StartResultAck` that contains an error, the FBlock shall not execute the corresponding method.	

REQ	7.46 AL – Error and ErrorAck – Sending error stops method execution
If an FBlock sends an error message for a method that is being executed, it shall stop execution of the method.	

REQ	7.47 AL – Error and ErrorAck – Error message structure
The error message shall contain an error code (`ErrorCode`) in the data field (`Data[0]` for `Error` or `Data[2]` for `ErrorAck`) as specified in Table 25.	

The error message may contain additional error information (`ErrorInfo`) in the data field (`Data[1]` for `Error` or `Data[3]` for `ErrorAck`).

`Data[0] to Data[n]` represent the payload of an application message; see ISO 21806-4:2020, 7.2.2[8].

Table 25 — ErrorCodes and additional information

Error category	ErrorCode[a]	ErrorCode description	ErrorInfo[b]	ErrorInfo description
Syntax errors	01_{16}	FBlockID not available	---	No information.
	02_{16}	InstID not available	---	No information.
	03_{16}	FktID not available	---	No information.
	04_{16}	OPType not available	<OPType> (Unsigned Byte)	Return the invalid OPType.
Segmentation error	$0C_{16}$	Segmentation error ErrorInfo contains a value in the range of 01_{16} to 07_{16}.	01_{16} (Unsigned Byte)	First segment missing, that is, the first telegram of a segmented transfer is not received.
			02_{16} (Unsigned Byte)	Target node does not provide enough buffers to handle a message of this size.
			03_{16} (Unsigned Byte)	Unexpected segment number.
			04_{16} (Unsigned Byte)	Too many unfinished segmentation messages pending.
			05_{16} (Unsigned Byte)	Timeout while waiting for next segment.
			06_{16} (Unsigned Byte)	Node not capable of handling segmented transfers.
			07_{16} (Unsigned Byte)	Segmented transfer is not finished before the arrival of another message with identical FBlockID, InstID, FktID, and OPType sent by the same node.
Application errors	05_{16}	Invalid length	---	No information.
	06_{16}	Parameter wrong/out of range One or more of the parameters are not within the boundaries specified for the function.	<ParameterPosition> (Unsigned Byte) {,<ParameterValue>} (data type of first incorrect parameter)	ParameterPosition contains the position of the parameter, where the value 1 corresponds to the first parameter. If a SenderHandle is present, 1 corresponds to the first parameter following the SenderHandle. ParameterValue is the value of the first incorrect parameter (optional). Parameters with higher position numbers are ignored.

[a] Data[0] (Error) Data[2] (ErrorAck).

[b] Data[1] to Data[n] (Error) Data[3] to Data[n] (ErrorAck).

Table 25 *(continued)*

Error category	ErrorCode[a]	ErrorCode description	ErrorInfo[b]	ErrorInfo description
Application errors	07_{16}	Parameter not available One or more of the parameters are within the boundaries specified for the function, but are not available at that time.	<ParameterPosition> (Unsigned Byte) {,<ParameterValue>} (data type of first incorrect parameter)	ParameterPosition contains the position of the parameter, where the value 1 corresponds to the first parameter. If a SenderHandle is present, 1 corresponds to the first parameter following the SenderHandle. ParameterValue is the value of the first unavailable parameter (optional). Parameters with higher position numbers are ignored.
	08_{16}	Reserved. Usage deprecated	---	No information.
	09_{16}	Reserved. Usage deprecated	---	No information.
	$0A_{16}$	Reserved	---	No information.
	$0B_{16}$	Device malfunction	---	No information.
	20_{16}	Function specific. After this ErrorCode, any function specific ErrorInfo can be sent.	<ErrorInfoID> (Unsigned Byte) {,<ErrorInfoDetail>} (data type depends on function specific requirements)	ErrorInfoDetail is optional and described in the function catalogue.
	40_{16}	Busy Function is available, but is busy.	---	No information.
	41_{16}	Not available Function is implemented in principle, but is not available at the moment.	---	No information.
	42_{16}	Processing error	---	No information.
	43_{16}	Method aborted This ErrorCode is used to indicate termination of a method by the Abort/AbortAckOPTypes.	---	No information.

[a] Data[0] (Error) Data[2] (ErrorAck).

[b] Data[1] to Data[n] (Error) Data[3] to Data[n] (ErrorAck).

Table 25 *(continued)*

Error category	ErrorCode[a]	ErrorCode description	ErrorInfo[b]	ErrorInfo description
Network owner-specific errors	$C0_{16}$ to EF_{16}	Network owner- (e.g. vehicle manufacturer) specific	Optional	---
Device/ECU supplier-specific errors	$F0_{16}$ to FE_{16}	Device-/ECU-supplier specific After this ErrorCode, any supplier specific ErrorInfo can be sent.	Optional	Device/ECU supplier specific ErrorInfo.
Reserved	FF_{16}	Reserved	---	---
[a] Data[0] (Error) Data[2] (ErrorAck). [b] Data[1] to Data[n] (Error) Data[3] to Data[n] (ErrorAck).				

The function-specific ErrorCode (20_{16}) should not be used, instead ErrorCodes in the ranges 01_{16} to $0B_{16}$ and 40_{16} to 43_{16} should be used.

Error messages, except segmentation errors, are not limited to single transfers.

In general, no error messages are sent from a controller to the FBlock. Segmentation errors are the only exception.

REQ	7.48 AL – Error and ErrorAck – No Error reply for reports
Except for segmentation errors, if a node receives a report (OPType > 8), it shall not react by sending a message with OPType Error.	

A controller never sends OPType ErrorAck to the FBlock.

REQ	7.49 AL – Error and ErrorAck – No ErrorAck reply for reports
If a node receives a report (OPType > 8), it shall not react by sending a message with OPType ErrorAck.	

REQ	7.50 AL – Error and ErrorAck – No error report for multicast messages
Except for segmentation errors, a node shall not send an error message when the corresponding command is received as multicast message (blocking broadcast, non-blocking broadcast, groupcast, as well as wildcards for all instances, see 7.4.7).	

Error messages are sent to report different kinds of errors.

a) Syntax error (ErrorCode 01_{16} to 04_{16})

A syntax error occurs if, for example, a function is accessed that does not exist or if an OPType that is not implemented is called. A syntax error is reported after reception of a faulty command. This also applies to methods, which are not started in that case.

Example for requesting a non-existing FBlock:

— SrcAdr → TrgAdr: FBlockID.InstID.FktID.OPType(…)

If the FBlock is not available:

— TrgAdr → SrcAdr: FBlockID.InstID.FktID.Error(ErrorCode=01_{16})

— Because ErrorCode 04_{16} (OPType not available) is only reported from FBlock to controller, the value for ErrorInfo—containing the OPType—should be in the command range (00_{16} to 08_{16}).

b) Segmentation error (ErrorCode $0C_{16}$)

Errors during reassembly of the original message in the receiver can be caused, for example, by missing segments, wrong order of arrival, or exceeding the timeout between two segments. The segmentation error informs the sender about the failure of the transfer. The MOST network service reports the error to the application within the sender of the original message.

The application may react in an appropriate way, for example, by trying to send the same message again. The reaction depends on the problem that caused the error.

Segmentation error is not limited to the context of segmented transfers. The error may also be reported as a result of a single transfer reception failure. The most likely reason in that case is an input buffer overflow.

REQ	7.51 AL – Error and ErrorAck – Reporting segmentation error
If a node detects a segmentation error, it shall send an error message with the corresponding ErrorCode/ ErrorInfo parameters.	

REQ	7.52 AL – Error and ErrorAck – Use Error OPType for segmentation error
When a node reports a segmentation error, it shall use the Error OPType.	

REQ	7.53 AL – Error and ErrorAck – Single transfer for segmentation error
When a node reports a segmentation error, it shall send the error message as single transfer.	

In the case of methods using the SenderHandle parameter (Ack-methods), as a result of the segmentation error, the segment containing the sender handle might be missing. Therefore, the ErrorAck OPType, which requires the SenderHandle parameter, is not used.

c) Application error

1) Parameter error (ErrorCode 05_{16} to 06_{16})

REQ	7.54 AL – Error and ErrorAck – Invalid length error
If the specified length of a received message does not match the actual length of the data field, an FBlock shall send an error message with ErrorCode 05_{16} (invalid length). NOTE This means that, compared to the message definition, there are too few parameters or too many parameters.	

REQ	7.55 AL – Error and ErrorAck – Parameter out of range error
If a parameter is out of range, an FBlock shall send an error message with ErrorCode 06_{16} (parameter wrong/ out of range).	

2) Messages are only accepted when the expected number of parameters is present and all parameters are within the specified range. Arrays are an exception because they may be shorter than specified (see 8.2.3.6). If an FBlock receives a command in which a parameter contains a value which is specified as “reserved” or “not allowed”, the FBlock should reply with an error message, containing ErrorCode 06_{16} (parameter wrong).

3) Temporarily not available (ErrorCode 07_{16}, 40_{16}, 41_{16})

In some cases, the message is correct, but execution is not possible at the moment. The following cases can be distinguished:

— Methods cannot be executed or properties cannot be accessed due to operation status. Sending a text message (SMSSend) using the mobile phone is an example of a method, which cannot be executed if the communication network is not available. If execution of the method is requested, it sends an error message with ErrorCode 41_{16} (not available). In such

a case, the controller can supervise the status of the telephone and repeat the sending of the SMS as soon as the network is available again.

— A method may be busy at the moment. For example, `SMSSend` of the telephone is busy while sending another SMS. In that case, `ErrorCode` 40_{16} (busy) is reported. The controller may perform retries. `ErrorCode` 40_{16} is only reported for methods.

If a method that does not support multiple instances is still running and any controller tries to start the same method again, the method should reply with an error, using `ErrorCode` 40_{16} (busy).

A property cannot be busy by definition but a value within the valid range might not be selectable at the moment. An example is property `DeckStatus` of the CD drive, which cannot be set to "play" if there is no CD loaded. Depending on the system design, this can generate an `ErrorCode` 07_{16} (parameter not available).

4) Device malfunction (`ErrorCode` $0B_{16}$)

This error indicates that the requested function is temporarily not available due to a device malfunction.

5) Specific execution error (`ErrorCode` 20_{16})

`ErrorCode` 20_{16} (function specific) is used to report specific errors during function execution.

6) General execution error (`ErrorCode` 42_{16})

Especially when using methods, execution errors may occur. Such an error (unspecific; command is correct, but execution failed) may be reported by `ErrorCode` 42_{16} (processing error).

7) Method aborted (`ErrorCode` 43_{16})

REQ	7.56 AL – Error and ErrorAck – Method aborted
If an FBlock receives a command with `OPType Abort` or `AbortAck`, the FBlock shall send an error message with `ErrorCode` 43_{16} (method aborted).	

The examination and processing of errors are done in the logical and temporary sequence as described above and in Figure 29.

8) Reserved (`ErrorCode` FF_{16})

`ErrorCode` FF_{16} should not be reported.

Figure 29 shows how application messages are checked for errors.

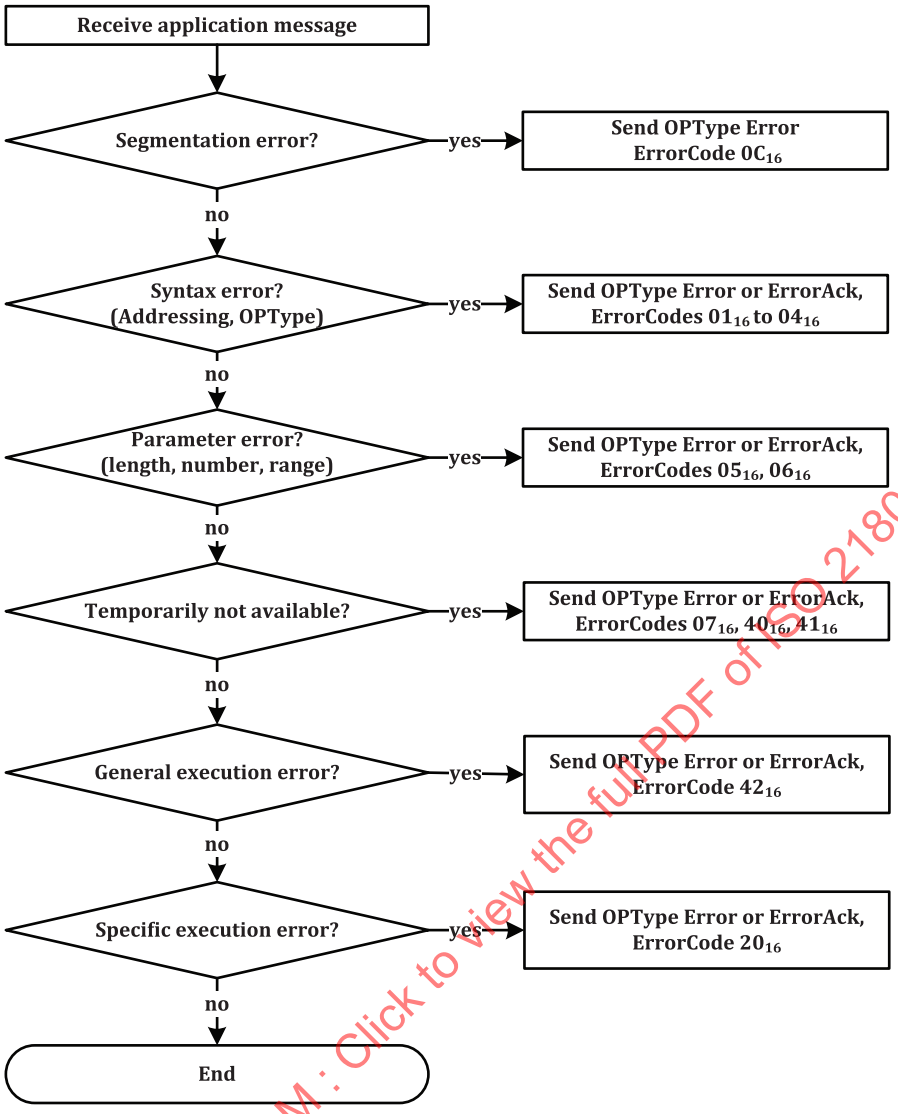


Figure 29 — Error checks for application messages

The network owner may specify additional application error management and the corresponding error messages.

7.6.11 AL — Parameters

The parameters of the operation follow the OPType, resulting in this structure:

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

REQ 7.57, 7.58, and 7.59 specify naming conventions for parameters.

REQ	7.57 AL – Parameters – Identical names
Within one function, parameters with identical names shall be of the same data type and shall have an identical structure.	

REQ	7.58 AL – Parameters – Permissible name characters
Parameter names shall contain no characters other than letters, numbers, and underscores.	

REQ	7.59 AL – Parameters – Name without leading number
Parameter names shall not start with a number.	

The structure and the restrictions for the parameters are specified in Clause 8.

7.7 AL — Timing definitions

7.7.1 AL — Definitions

7.7.1.1 AL — Exceptions to timing restrictions

Timers can be interrupted and as a potential result the corresponding timing restrictions are not met when a node executes a `cmd_Emergency_Shutdown` request, or transitions through reset or `s_NetInterface_Sleep`.

7.7.1.2 AL — Timers

Timers are initialised by setting the current timer value to 0. After a timer is started, it runs until it is paused, stopped, or it expires. When a timer is paused, the current timer value is retained. When a timer is stopped or expires, the current timer value is set to 0. When a timer is started, it runs from the current timer value.

When a timer expires, the specified action (e.g. error handling) is performed.

Whenever a timer definition contains the character "-" instead of a value, that particular value is "not specified".

7.7.1.3 AL — Timing constraints

All timing constraints related to MOST messages are delimited by the presence of the corresponding telegrams on the MOST network.

Timing constraints represent an upper limit for a particular action.

7.7.2 AL — General communication

7.7.2.1 AL — Overview

REQ	7.60 AL – Timing definitions – General communication timing constraints
The values specified in Table 26 shall apply to the corresponding timing constraints.	

Table 26 — General communication timing constraints

Name	Value	Unit	Purpose
t_{Property}	Network owner specific	ms	Limit for responding to a command that reads a property
$t_{\text{NotificationProperty}}$	Network owner specific	ms	Limit for reacting to a `Notification.Set` message

REQ	7.61 AL – Timing definitions – General communication timers
The values specified in Table 27 shall apply to the corresponding timers.	

Table 27 — General communication timers

Name	Minimum value	Typical value	Maximum value	Unit	Purpose
$t_{WaitForProperty}$ [a]	250	---	Network owner specific	ms	Time to wait for the response after sending a command to read a property
$t_{CM\_DeadlockPrev}$	Network owner specific	---	1 000	ms	Time that prevents deadlocks during connection building
[a] The maximum value is dependent on the retry settings.					

7.7.2.2 AL — Constraint $t_{Property}$

7.7.2.2.1 AL — Purpose

$t_{Property}$ is the permissible duration between complete reception of a command to read a property and sending the initial telegram of the response.

7.7.2.2.2 AL — Validity conditions

REQ	7.62 AL – Timing definitions – $t_{Property}$ validity
If a node is not currently sending another message, $t_{Property}$ shall become valid when the node completely receives a command to read a property.	

REQ	7.63 AL – Timing definitions – $t_{Property}$ stops applying
$t_{Property}$ shall stop applying when a node receiving a command to read a property sends the initial telegram of the response.	

7.7.2.2.3 AL — Violation consequences

If the initial telegram of the response is not sent within $t_{Property}$, $t_{WaitForProperty}$ might expire and the controller might repeat the command for reading the property.

7.7.2.3 Constraint $t_{NotificationProperty}$

7.7.2.3.1 Purpose

$t_{NotificationProperty}$ is the permissible duration between receiving a `Notification.Set` message and the initial telegram of any of the affected functions.

The network owner may set $t_{NotificationProperty}$ to the same value as $t_{Property}$.

7.7.2.3.2 AL — Validity conditions

REQ	7.64 AL – Timing definitions – $t_{NotificationProperty}$ validity
If a node is not currently sending another message, $t_{NotificationProperty}$ shall become valid when the node completely receives a `Notification.Set` message.	

REQ	7.65 AL – Timing definitions – $t_{NotificationProperty}$ stops applying
$t_{NotificationProperty}$ shall stop applying when a node that received a `Notification.Set` message sends the initial telegram of the response.	

7.7.2.3.3 AL — Violation consequences

If the initial telegram of any of the affected functions is not sent within $t_{NotificationProperty}$, the requesting controller might retry setting the notification.

7.7.2.4 AL — Timer $t_{WaitForProperty}$

7.7.2.4.1 AL — Purpose

$t_{WaitForProperty}$ controls how long a controller waits for the response after sending a command to read a property.

7.7.2.4.2 AL — Start and Stop conditions

REQ	7.66 AL – Timing definitions – $t_{WaitForProperty}$ start
A controller shall start $t_{WaitForProperty}$ when it sends a message with `OPType Get` or `SetGet`.	

REQ	7.67 AL – Timing definitions – $t_{WaitForProperty}$ stop
A controller shall stop $t_{WaitForProperty}$ when it receives a reply to a message with `OPType Get` or `SetGet`.	

7.7.2.4.3 AL — Timer expiration

REQ	7.68 AL – Timing definitions – $t_{WaitForProperty}$ expiration
When $t_{WaitForProperty}$ expires, the controller shall behave as if any error message is received.	

7.7.2.5 AL — Timer $t_{CM\_DeadlockPrev}$

7.7.2.5.1 AL — Purpose

$t_{CM\_DeadlockPrev}$ controls how long the connection building process may take before the connection manager regards it as failed.

7.7.2.5.2 AL — Start and Stop conditions

REQ	7.69 AL – Timing definitions – $t_{CM\_DeadlockPrev}$ start
The connection manager shall start $t_{CM\_DeadlockPrev}$ when it sends the first request for building a connection to a source/sink FBlock.	

REQ	7.70 AL – Timing definitions – $t_{CM\_DeadlockPrev}$ stop
The connection manager shall stop $t_{CM\_DeadlockPrev}$ when the corresponding connection building process is completed or fails.	

7.7.2.5.3 AL — Timer expiration

REQ	7.71 AL – Timing definitions – $t_{CM\_DeadlockPrev}$ expiration
When $t_{CM\_DeadlockPrev}$ expires, the connection manager shall behave as if the corresponding connection building process failed.	

8 PL — Presentation layer

8.1 PL — Data and basic data types

8.1.1 PL — General

Within the data field, none, one, or multiple parameters of the data types can be transported in any allowed combination.

REQ	6.1 PL – Data and basic data types – Big Endian alignment
Instances of data types shall be transported with the most significant byte first (Big Endian alignment) and most significant bit first.	

REQ	6.2 PL – Data and basic data types – Sign encoded in the most significant bit
The sign of the signed integer data types shall be encoded in the most significant bit.	

REQ	6.3 PL – Data and basic data types – Signed integer represented in two's complement
Signed integer data types shall be represented in two's complement.	

Angle brackets "<...>" are used to group the fields of structured data types like Array Type or Record Type. The content between the opening bracket and the closing bracket corresponds to one MOST parameter.

Table 28 specifies the default values for basic data types, which are used to initialise variables.

Table 28 — Default values for basic data types

Data type	Value
`Boolean`	$0000\ 0000_2$
`Enum`	Minimal specified value
`Unsigned Byte`	00_{16}
`Signed Byte`	00_{16}
`Unsigned Word`	0000_{16}
`Signed Word`	0000_{16}
`Unsigned Long`	$0000\ 0000_{16}$
`Signed Long`	$0000\ 0000_{16}$
`Unsigned Long Long`	$0000\ 0000\ 0000\ 0000_{16}$
`Signed Long Long`	$0000\ 0000\ 0000\ 0000_{16}$
`Float`	0,0
`Double`	0,0
`Length-coded String`	Empty Length-coded String (<01_{16}, $0000\ 0000_{16}$>)
`Stream`	If stream parameters are specified, the default value of every stream parameter corresponds to the default value of the underlying data type.
`Array Type`	`ElementCount` size type `Unsigned Byte`: 00_{16} `ElementCount` size type `Unsigned Word`: 0000_{16} `ElementCount` size type `Unsigned Long`: $0000\ 0000_{16}$
`Record Type`	Default value for each member of the Record
`BitField`	Size 1: Mask, Data = (0_{16}, 0_{16}) Size 2: Mask, Data = (00_{16}, 00_{16}) Size 4: Mask, Data = (0000_{16}, 0000_{16}) Size 8: Mask, Data = ($0000\ 0000_{16}$, $0000\ 0000_{16}$)

Table 28 *(continued)*

Data type	Value
`String`	Empty String (<01_{16}, "\0">)
`Stream`	The default value of a `Stream` is the empty `Stream`, that is, no data is stored or transmitted. If stream parameters are repeated, zero is the default for repetitions. The default value for every stream signal is 0.
`Short Stream`	The default value of a `Short Stream` is the empty `Short Stream`, where only the `Length` field exists. The `Length` field contains the value 00_{16}. If stream parameters are specified, the default value of every stream parameter corresponds to the default value of the underlying data type. If stream parameters are repeated, zero is the default for repetitions. The default value for every stream signal is 0.
`Classified Stream`	The default value of the `Classified Stream` is the empty `Classified Stream`, containing only the `Length` field and the empty `String` for the MediaType: <0001_{16}, "\0">

Some data types have attributes, for example, maximum size or number of elements. These attributes are specified in the MOST FBlock library. In the following subclauses, these are labelled as "MOST FBlock library definitions".

8.1.2 PL — Boolean

REQ	6.4 PL – Data and basic data types – Boolean attributes and values
The attributes and values for data type `Boolean` shall be in accordance with Table 29.	

Table 29 — Boolean

Attribute	Value
Name	`Boolean`
Summary	Logical data type that has one of these values: true or false.
Size	1 byte
Encoding	false = `0000 0000`$_2$, true = `0000 0001`$_2$
Description	Only the least significant bit of the byte can be used.

REQ	6.5 PL – Data and basic data types – Single bit values
If a single bit value is interpreted as `Boolean`, the value 0 shall correspond to false and 1 shall correspond to true.	

8.1.3 PL — Enum

REQ	6.6 PL – Data and basic data types – Enum attributes and values
The attributes and values for data type `Enum` shall be in accordance with Table 30.	

REQ	6.7 PL – Data and basic data types – Unique Enum values
The value of each symbol of an `Enum` shall be unique in the enumeration.	

REQ	6.8 PL – Data and basic data types – Range of Enum values
`Enum` values shall be in the range of 0 and 2^8 - 1, 2^{16} - 1, 2^{32} - 1, or 2^{64} - 1 depending on the size for 1, 2, 4, or 8 data bytes, respectively.	

Table 30 — Enum

Attribute	Value
Name	`Enum`
Summary	Enumeration of constant symbols and values
Size	1, 2, 4, or 8 bytes
Encoding	Enum: <Symbol$_1$> = <Value$_1$>, <Symbol$_2$> = <Value$_2$>, ..., <Symbol$_n$> = <Value$_n$> The values are encoded as unsigned integers.
Description	The maximum number of symbols of an Enum depends on the size and is 2^8, 2^{16}, 2^{32}, or 2^{64} for 1, 2, 4, or 8 data bytes respectively.

8.1.4 PL — Numeric data types

8.1.4.1 PL — FBlock library attributes for numeric data types

The attributes in Table 31 are relevant when describing numeric data types in the MOST FBlock library.

Table 31 — FBlock library attributes for numeric data types

MOST FBlock library definition	Data type	Description
Min	Same as value	Smallest valid value of a value range
Max	Same as value	Largest valid value of a value range
Unit	---	Unit, according to the International System of Units (SI). The default value for Unit is “none”.

The valid range of data elements that are based on numeric data types can be specified by the use of value ranges. A value range is a number pair that consists of minimum value and maximum value.

For any data element that is based on a number data type, it should be possible to specify no range, one range, or multiple ranges.

Multiple ranges should not overlap.

REQ	6.9 PL – Data and basic data types – Valid value range for number data types
If no value range is specified, the valid range shall include the smallest and largest possible value that the data type can represent and every value in between.	

8.1.4.2 PL — Integer data types

8.1.4.2.1 PL — Attributes for integer data types

The attributes in Table 32 are relevant when describing integer data types in the MOST FBlock library.

Table 32 — Attributes for integer data types

MOST FBlock library definition	Data type	Description
`Exponent`	`Signed Byte`	Position of decimal comma; Value = Number × $10^{\texttt{Exponent}}$ The default value for `Exponent` is 0.
`Step`	`Unsigned Byte`	Step width for adjusting. The following applies: — Max: Min + (n × `Step`) — Default value for `Step`: 1.

8.1.4.2.2 PL — Fixed-point representation of integers

The information about the location of the decimal comma is contained in the exponent constant. Therefore, the location of the decimal comma is fixed and is not transmitted with the actual value.

The conversion between the fixed-point representation of a number and its transmitted value is specified by the following:

— $Value_{Transmitted}$: $Number_{Fixed\text{-}point} \times 10^{Exponent}$

— $Number_{Fixed\text{-}point}$: $Value_{Transmitted} \times 10^{Exponent}$

EXAMPLE 1

$Value_{Transmitted}$: 1 073 (`Unsigned Word`)

`Exponent`: –1

Unit: MHz

`Step`: 1

$Number_{Fixed\text{-}point}$: 107,3 MHz

In the case of an `Increment` operation with `NSteps` = 5, the current frequency is incremented from 107,3 MHz to 107,8 MHz.

EXAMPLE 2

$Value_{Transmitted}$: 1 073 (`Unsigned Word`)

`Exponent`: 5

Unit: Hz

`Step`: 1

$Number_{Fixed\text{-}point}$: 107 300 000 Hz

In the case of an `Increment` operation with `NSteps` = 5, the current frequency is incremented from 107 300 000 Hz to 107 800 000 Hz.

EXAMPLE 3

$Value_{Transmitted}$: 1 000 (`Unsigned Word`)

`Exponent`: –3

Unit: m

`Step`: 10

$Number_{Fixed\text{-}point}$: 1,000 m

In the case of an `Increment` operation with `NSteps` = 5, the current length is incremented from 1,000 m to 1,050 m.

8.1.4.2.3 PL — Bit-oriented representation of unsigned integers

The data types `Unsigned Byte`, `Unsigned Word`, `Unsigned Long`, and `Unsigned Long Long` may be used to represent a series of single-bit or multi-bit values.

One value that consists of a single bit or multiple bits is called a flag.

REQ	6.10 PL – Data and basic data types – Order of flags for bit-oriented integer
If multiple flags exist, they shall be ordered right to left, that is, the first signal includes the least significant bit (LSB).	

REQ	6.11 PL – Data and basic data types – No gaps for bit-oriented integers
If multiple flags exist, there shall be no gaps between the flags.	

REQ	6.12 PL – Data and basic data types – No Increment or Decrement for bit-oriented integers
The `OPTypes Increment` and `Decrement` shall not be used on an unsigned integer data element when it is used in the bit-oriented representation.	

Figure 30 shows a bit-oriented Unsigned Word example, a series of Unsigned Word-based flags. 11 bits are used for representing 11 single bit values or flags.

Byte 1								Byte 0							
D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
					F.10	F.9	F.8	F.7	F.6	F.5	F.4	F.3	F.2	F.1	F.0

Figure 30 — Bit-oriented Unsigned Word example

8.1.4.2.4 PL — Unsigned Byte

REQ	6.13 PL – Data and basic data types – Unsigned Byte attributes and values
The attributes and values for data type `Unsigned Byte` shall be in accordance with Table 33.	

Table 33 — Unsigned Byte

Attribute	Value
Name	`Unsigned Byte`
Summary	Unsigned, 8-bit integer
Size	1 byte
Encoding	Binary numeral system encoding
Description	---

8.1.4.2.5 PL — Signed Byte

REQ	6.14 PL – Data and basic data types – Signed Byte attributes and values
The attributes and values for data type `Signed Byte` shall be in accordance with Table 34.	

Table 34 — Signed Byte

Attribute	Value
Name	`Signed Byte`
Summary	Signed, 8-bit integer
Size	1 byte
Encoding	Two's complement encoding
Description	---

8.1.4.2.6 PL — Unsigned Word

REQ	6.15 PL – Data and basic data types – Unsigned Word attributes and values
The attributes and values for data type `Unsigned Word` shall be in accordance with Table 35.	

Table 35 — Unsigned Word

Attribute	Value
Name	`Unsigned Word`
Summary	Unsigned, 16-bit integer
Size	2 bytes
Encoding	Binary numeral system encoding
Description	---

8.1.4.2.7 PL — Signed Word

REQ	6.16 PL – Data and basic data types – Signed Word attributes and values
The attributes and values for data type `Signed Word` shall be in accordance with Table 36.	

Table 36 — Signed Word

Attribute	Value
Name	`Signed Word`
Summary	Signed, 16-bit integer
Size	2 bytes
Encoding	Two's complement encoding
Description	---

8.1.4.2.8 PL — Unsigned Long

REQ	6.17 PL – Data and basic data types – Unsigned Long attributes and values
The attributes and values for data type `Unsigned Long` shall be in accordance with Table 37.	

Table 37 — Unsigned Long

Attribute	Value
Name	`Unsigned Long`
Summary	Unsigned, 32-bit integer
Size	4 bytes
Encoding	Binary numeral system encoding
Description	---

8.1.4.2.9 PL — Signed Long

REQ	6.18 PL – Data and basic data types – Signed Long attributes and values
The attributes and values for data type `Signed Long` shall be in accordance with Table 38.	

Table 38 — Signed Long

Attribute	Value
Name	`Signed Long`
Summary	Signed, 32-bit integer
Size	4 bytes
Encoding	Two's complement encoding
Description	---

8.1.4.2.10 PL — Unsigned Long Long

REQ	6.19 PL – Data and basic data types – Unsigned Long Long attributes and values
The attributes and values for data type `Unsigned Long Long` shall be in accordance with Table 39.	

Table 39 — Unsigned Long Long

Attribute	Value
Name	`Unsigned Long Long`
Summary	Unsigned, 64-bit integer
Size	8 bytes
Encoding	Binary numeral system encoding
Description	---

8.1.4.2.11 PL — Signed Long Long

REQ	6.20 PL – Data and basic data types – Signed Long Long attributes and values
The attributes and values for data type `Signed Long Long` shall be in accordance with Table 40.	

Table 40 — Signed Long Long

Attribute	Value
Name	`Signed Long Long`
Summary	Signed, 64-bit integer
Size	8 bytes
Encoding	Two's complement encoding
Description	---

8.1.4.3 PL — Floating-point data types

8.1.4.3.1 PL — Float

REQ	6.21 PL – Data and basic data types – Float attributes and values
The attributes and values for data type `Float` shall be in accordance with Table 41.	

Table 41 — Float

Attribute	Value
Name	`Float`
Summary	Signed, 32-bit floating-point number
Size	4 bytes
Encoding	Encoding according to IEEE 754-2008.
Description	---

8.1.4.3.2 PL — Double

REQ	6.22 PL – Data and basic data types – Double attributes and values
The attributes and values for data type `Double` shall be in accordance with Table 42.	

Table 42 — Double

Attribute	Value
Name	`Double`
Summary	Signed, 64-bit floating-point number
Size	8 bytes
Encoding	Encoding according to IEEE 754-2008.
Description	---

8.1.5 PL — Length-coded String

REQ	6.23 PL – Data and basic data types – Length-coded String attributes and values
The attributes and values for data type `Length-coded String` shall be in accordance with Table 43.	

Table 43 — Length-coded String

Attribute	Value
Name	`Length-coded String`
Summary	Length-coded sequence of characters with a particular encoding
Size	Arbitrary size up to 2^{32}-1 data bytes
Encoding	`Length-coded String`: (`Identifier`, `Length`, `Data`) `Identifier`: `Unsigned Byte;` specifies the subsequent encoding of the string. `Length`: `Unsigned Long;` specifies the length of the Data field in bytes. `Data`: arbitrary; contains the sequence of characters of the string.
Description	Every `Length-coded String` starts with an `Identifier`, which specifies the encoding of the rest of the string. All instances of data type `Length-coded String` are length-coded, i.e. they contain a `Length` field, which specifies the length of the `Data` field in bytes. A `Length-coded String` does not require a terminating null character. The length of a `Length-coded String` is limited to 2^{32}-1 bytes. See Table 53 under data type `String` for the list of `Identifiers` and their corresponding encodings. See Table 44 for a list of representations of empty `Length-coded String`s. `Length-coded String`s are always transmitted with the most significant byte of a multi-byte character first.

Table 44 illustrates the representation of empty `Length-coded String`s in various encodings.

Table 44 — Empty Length-coded Strings

Character encoding	Empty Length-coded String
Unicode, UTF-16	00_{16} 0000 0000_{16}
ISO/IEC 8859-15[4], 8 bit	01_{16} 0000 0000_{16}
Unicode, UTF-8	02_{16} 0000 0000_{16}
RDS	03_{16} 0000 0000_{16}
DAB Charset 0001	04_{16} 0000 0000_{16}
DAB Charset 0010	05_{16} 0000 0000_{16}
DAB Charset 0011	06_{16} 0000 0000_{16}
SHIFT_JIS	07_{16} 0000 0000_{16}

The attribute in Table 45 is relevant when describing Length-coded String data elements in the MOST FBlock library.

Table 45 — Attributes for Length-coded Strings

MOST FBlock library definition	Data Type	Description
`MaxSize`	`Unsigned Long`	Maximum size of the Length-coded String in bytes

8.1.6 PL — Array Type

REQ	6.24 PL – Data and basic data types – Array Type attributes and values
The attributes and values for data type `Array Type` shall be in accordance with Table 46.	

Table 46 — Array Type

Attribute	Value
Name	`Array Type`
Summary	Ordered collection of instances of the same data type
Size	Maximum size depends on `ElementCount` data type.
Encoding	`Array Type`: (`ElementCount`, `Element1`, `Element2`, ..., `Elementn`) `ElementCount`: `Unsigned Byte`, `Unsigned Word`, or `Unsigned Long` Contains the number of elements in the array. `Elementi`: Arbitrary type Contains the i^{th} element of the array.

Figure 31 shows an `Array Type` example, using an array of three `Unsigned Long` values: 8, 13, 21.

`ElementCount`: Data type `Unsigned Long`

`Element1` represents the value 8, `Element2` represents 13, and `Element3` represents 21.

	ElementCount				Element1				Element2				Element3			
Byte Nr.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Data	00_{16}	00_{16}	00_{16}	03_{16}	00_{16}	00_{16}	00_{16}	08_{16}	00_{16}	00_{16}	00_{16}	$0D_{16}$	00_{16}	00_{16}	00_{16}	15_{16}

Figure 31 — Array Type example 1

Figure 32 shows an `Array Type` example for `Length-coded String`s, containing an Array of two `Length-coded String`s: “MOSTCO”, “MOST150”.

`ElementCount`: Data type `Unsigned Long`

The two elements “MOSTCO” and “MOST150” are encoded according to ISO/IEC 8859-15[4].

	ElementCount				Element1										
Byte Nr.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Data	00_{16}	00_{16}	00_{16}	02_{16}	01_{16}	00_{16}	00_{16}	00_{16}	06_{16}	M	O	S	T	C	O

	Element2											
Byte Nr.	15	16	17	18	19	20	21	22	23	24	25	26
Data	01_{16}	00_{16}	00_{16}	00_{16}	07_{16}	M	O	S	T	1	5	0

Figure 32 — Array Type example 2

8.1.7 PL — Record Type

REQ	**6.25 PL – Data and basic data types – Record Type attributes and values**
The attributes and values for data type `Record Type` shall be in accordance with Table 47.	

Table 47 — Record Type

Attribute	Value
Name	`Record Type`
Summary	Fixed sequence of instances of different data types
Size	Arbitrary size
Encoding	`Record Type`: (`Member1`, `Member2`, ..., `Membern`) `Memberi`: Arbitrary type; contains the i^{th} member of the record.
Description	---

Figure 33 shows a `Record Type` example, containing an address Record with the following members:

— Identifier: 12345678 `Unsigned Long`

— Name: “John Smith” `Length-coded String` in ISO/IEC 8859-15[4] encoding

— ZIP: 12345 `Unsigned Long`

— Phone: “123-456-7890” `Length-coded String` in ISO/IEC 8859-15[4] encoding

	Identifier				Name														
Byte Nr.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Data	00_{16}	BC_{16}	61_{16}	$4E_{16}$	01_{16}	00_{16}	00_{16}	00_{16}	$0A_{16}$	J	o	h	n	20_{16}	S	m	i	t	h

	ZIP				Phone															
Byte Nr.	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
Data	00_{16}	00_{16}	30_{16}	39_{16}	01_{16}	00_{16}	00_{16}	00_{16}	$0C_{16}$	1	2	3	-	4	5	6	-	7	8	9	0

Figure 33 — Record Type example

8.1.8 PL — Stream

8.1.8.1 PL — Attributes and values

Table 48 provides the attributes and values for data type `Stream`.

Simple streams, unstructured streams, cyclically recurring stream parameters, and stream signals should not be used.

Table 48 — Attributes and values for data type Stream

Attribute	Value
Name	`Stream`
Summary	Sequence of unformatted data bytes without length information
Size	Arbitrary size
Encoding	---
Description	In the definition of streams, the following cases are distinguished: — Unstructured stream: A stream without structural information. — Simple stream: A stream that contains only stream parameters, possibly repeated. — Composed stream: A stream with one or more stream cases. — A composed stream is closely related to the notion of a Variant Record (Disjoint Union) in common programming languages. — The individual stream cases contain a series of stream parameters. — A stream parameter is a data element that is assigned a MOST data type. — Stream parameters may recur cyclically. Repetition is in-order and may be selective. — The length of a stream is undefined if it cannot be determined based on information provided in the MOST FBlock Library or MOST function catalogue. For example, this is the case for unstructured streams or streams with parameter repetition. — See REQ 6.28. — See REQ 6.29. — When the `Stream` with undefined length appears at the end of the message, the `Stream` length is based on the message length. When a `Stream` is embedded in a `Short Stream`, the length of the `Stream` can be determined by using the length information of the `Short Stream`.

REQ	6.26 PL – Data and basic data types – Stream position in a message
If a message contains a data type `Stream` with undefined length and the `Stream` is not embedded in a `Short Stream`, this `Stream` shall be at the end of the message.	

REQ	6.27 PL – Data and basic data types – Stream within Short Stream
If a `Short Stream` contains a data type `Stream` with undefined length, this `Stream` shall be at the end of the `Short Stream`.	

8.1.8.2 PL — Stream parameter repetition

Figure 34 shows a stream with three parameters, third parameter repeating cyclically.

Figure 34 — Transmission sequence for one repeated stream parameter

Figure 35 shows a stream with two parameters, both repeating cyclically.

Figure 35 — Transmission sequence for two repeated stream parameters

REQ	6.28 PL – Data and basic data types – Stream parameter repetition
Stream parameters or groups of stream parameters, which are not at the end of the parameter list, shall not be repeated.	

Therefore, the stream layout in Figure 36 with four parameters, second and fourth repeating cyclically, is not a permitted design.

Figure 36 — Illegal transmission sequence for two non-consecutive repeated stream parameters

8.1.8.3 PL — Stream signals

A stream signal represents a value that consists of an arbitrary number of bits.

EXAMPLE

In Figure 37, the state of one 4-bit port and two 2-bit ports is encoded for transmission. The `PortState` stream is divided into three stream signals.

Basis data type	Max. length	Condition	Description
Stream	---	---	Stream Signals: `Port0`, `Port1`, `Port2`

Signal Name	Bit position	Number of bits	Description
`Port0`	0	4	State of 4-bit port.
`Port1`	4	2	State of 2-bit port 1.
`Port2`	6	2	State of 2-bit port 2.

Figure 37 — PortState stream example

8.1.8.4 PL — Stream cases

REQ	6.29 PL – Data and basic data types – No stream signals in stream cases
Stream cases shall not contain stream signals.	

A selector value determines which specified type is used, that is, which stream case applies. The following requirements for stream cases ensure that the selector value can be identified unambiguously.

REQ	6.30 PL – Data and basic data types – Stream selector outside Stream
The selector value itself shall not be part of the stream it controls.	

REQ	6.31 Stream selector in front of Stream
The selector value shall be in front of the stream it controls.	

The selector of a composed stream may be embedded in another stream.

The stream selector and the corresponding stream should be transported in the same message. For example, a partial update containing a stream case without the stream selector should be avoided.

REQ	6.32 PL – Data and basic data types – Stream selector data types
The stream selector value shall be based on one of these data types: — `Enum` — `Unsigned Byte` — `Signed Byte` — `Unsigned Word` — `Signed Word` — `Unsigned Long` — `Signed Long` — `Unsigned Long Long` — `Signed Long Long` — `Short Stream`	

REQ	6.33 PL – Data and basic data types – No repeated stream parameters in stream cases
Repeated stream parameters shall not be used in stream cases.	

REQ	6.34 PL – Data and basic data types – No repeated data element as stream selector
The selector of a stream shall not be a repeated data element. NOTE Repeated data elements, for example, are repeated stream parameters and array elements.	

EXAMPLE

Information about a contact is encoded for transmission. The contact can be a real person, a company, or an institution. Depending on the kind of contact (`ContactType`) in Table 49, the number of transmitted data items (`ContactData`) in Table 50 varies.

`ContactType`: The `ContactType Enum` is the selector. If, for instance, the value of parameter `ContactType` is 00_{16}, the contact is a person.

Table 49 — Stream case example—Enum

Basis data type	Range of values	Code	Description
`Enum`	00_{16} to 02_{16}	00_{16}	Person
		01_{16}	Company
		02_{16}	Institution

ContactData: this is the actual contact information. Depending on the value of the selector (`ContactType`), `ContactData` contains different items. If, for example, `ContactType` is 00_{16} (the contact is a person), `ContactData` contains `LastName`, `MiddleName`, and `FirstName`.

Table 50 — Stream case example—Stream

Basis data type	Length	Condition	Description
`Stream`	---	`ContactType` = 00_{16}	Content: LastName, MiddleName, FirstName
		`ContactType` = 01_{16}	Content: CompanyName, CompanyType
		`ContactType` = 02_{16}	Content: InstitutionName

8.1.9 PL — BitField

REQ	6.35 PL – Data and basic data types – BitField attributes and values
The attributes and values for data type `BitField` shall be in accordance with Table 51.	

Table 51 — BitField

Attribute	Value
Name	`BitField`
Summary	Fixed-size array of Boolean values that can be individually selected and modified by a mask.
Size	1, 2, 4, or 8 bytes
Encoding	`BitField(Size)` = (`Mask`, `Data`) with `Mask` = `Data` = Size/2
Description	`Mask` is the masking bit field of the `Data` area. `Mask` and `Data` have the same size: each bit of `Mask` indicates whether the corresponding bit of the `Data` area is selected. This allows the individual selection of each bit as well as the processing of an arbitrary combination of bits of the `Data` area. The least significant bit of `Mask` corresponds to the least significant bit of the `Data` area and the kth bit of `Mask` corresponds to the kth bit of the `Data` area: Bit k of `Mask` = 1 → Bit k of `Data` is selected. Bit k of `Mask` = 0 → Bit k of `Data` is not selected.

EXAMPLE

State: `MyBitField.Status` (XXXX $XXXX_2$, 1010 1001_2)

Operation: `MyBitField.Set` (0000 1000_2, 1010 0111_2)

New state: `MyBitField.Status` (XXXX $XXXX_2$, 1010 0001_2)

"X" means "don't care" in this example. The FBlock should set these bits to zero in the `Status` message.

REQ	6.36 PL – Data and basic data types – Ignore BitField Mask in Status messages
When a controller receives a `Status` message containing a `BitField` parameter, it shall ignore the content of `Mask`.	

BitField in the Array and Record function classes

A BitField in an Array or Record function class represents one variable. That means it is addressable as an entity via one dedicated value of Pos.

```
CManager → Controller:  ConnectionMaster.1.ConnectionTable.Status
(<Source, Sink, BlockWidth, ConnectionLabel>,
<Source, Sink, BlockWidth, ConnectionLabel>,
<Source, Sink, BlockWidth, ConnectionLabel>,…)
```

Requesting Status report (1)

```
MyArray.Get(Position = <x = 00₁₆, y = 00₁₆>)
```

Answer

```
CManager → Controller:  ConnectionMaster.1.ConnectionTable.Status
(<Source, Sink, BlockWidth, ConnectionLabel>,
<Source, Sink, BlockWidth, ConnectionLabel>,
                           <Source, Sink, BlockWidth, ConnectionLabel>,…)
```

Requesting Status report (2)

```
MyArray.Get(Position = <x = 02₁₆, y = 00₁₆>)
```

Answer

```
MyArray.Status(Position = <x = 02₁₆, y = 00₁₆>,   XXXX XXXX₂, 1110 0011₂)
```

Performing a Set operation (1)

```
MyArray.Set(Position =   <x = 00₁₆, y = 00₁₆>,   1000 0001₂, 1000 0001₂,
                                                 1000 0001₂, 1000 0001₂,
                                                 1000 0001₂, 0111 1110₂,
                                                 1000 0001₂, 0111 1110₂)
```

Result

```
MyArray =   XXXX XXXX₂, 1100 1001₂
            XXXX XXXX₂, 1110 0011₂
            XXXX XXXX₂, 0010 1100₂
            XXXX XXXX₂, 0111 1110₂
```

Performing a Set operation (1)

```
MyArray.Set(Position = <x = 04₁₆, y = 00₁₆>, 1011 0000₂,1000 0001₂)
```

Result

```
MyArray =   XXXX XXXX₂, 1100 1001₂
            XXXX XXXX₂, 1110 0011₂
            XXXX XXXX₂, 0010 1100₂
            XXXX XXXX₂, 1000 1110₂
```

8.1.10 PL — String

REQ	6.37 PL – Data and basic data types – String attributes and values
The attributes and values for data type `String` shall be according to Table 52.	

Table 52 — String

Attribute	Value
Name	`String`
Summary	Null-terminated sequence of characters with a particular encoding.
Size	Arbitrary size up to 2^{32}-1 data bytes, including the Terminator.
Encoding	String = (`Identifier, Data, Terminator`)
	`Identifier`: `Unsigned Byte`; specifies the subsequent encoding of the `String`.
	`Data`: Arbitrary; contains the sequence of characters of the `String`.
	`Terminator`: null character; indicates the end of the `String`.

Table 52 *(continued)*

Attribute	Value
Description	Every String starts with an `Identifier`, which specifies the encoding of the rest of the String. In contrast to Length-coded Strings, all instances of data type String are null-terminated. Strings are terminated with the null character of the respective encoding (e.g. 00_{16} for the ISO/IEC 8859-15[4] encoding), which may be represented by multiple bytes. See Table 53 for the list of currently specified `Identifiers` and their corresponding encodings and Table 54 for a list of representations of empty Strings. Since all Strings are null-terminated, character sets that use a null character are not used. Strings are transmitted with the most significant byte of a multi-byte character first.

Table 53 specifies the list of `Identifiers` and their corresponding String encodings.

Table 53 — String encodings

Code	String type	ASCII compatible
00_{16}	The Unicode Standard[14], UTF-16	No
01_{16}	ISO/IEC 8859-15[4], 8 bit	Yes
02_{16}	The Unicode Standard[14], UTF-8	Yes
03_{16}	IEC 62106:1999[6], RDS	No
04_{16}	ETSI EN 300 401[11], DAB Charset 0001	No
05_{16}	ETSI EN 300 401[11], DAB Charset 0010	No
06_{16}	ETSI EN 300 401[11], DAB Charset 0011	Yes
07_{16}	JIS X0201:1997[12], JIS X0208:1997[13], SHIFT_JIS	No
08_{16} to BF_{16}	Reserved	---
$C0_{16}$ to EF_{16}	Network owner (e.g. vehicle manufacturer)	---
$F0_{16}$ to FF_{16}	Device/ECU supplier	---

Table 54 specifies the representation of the empty `String` in various encodings.

Table 54 — Empty Strings

Character encoding	Empty String
The Unicode Standard[14], UTF-16	00_{16} 00_{16} 00_{16}
ISO/IEC 8859-15[4], 8 bit	01_{16} 00_{16}
The Unicode Standard[14], UTF-8	02_{16} 00_{16}
IEC 62106:1999[6], RDS	03_{16} 00_{16}
ETSI EN 300 401[11], DAB Charset 0001	04_{16} 00_{16}
ETSI EN 300 401[11], DAB Charset 0010	05_{16} 00_{16}
ETSI EN 300 401[11], DAB Charset 0011	06_{16} 00_{16}
JIS X0201:1997[12], JIS X0208:1997[13], SHIFT_JIS	07_{16} 00_{16} 00_{16}

The attribute in Table 55 is relevant when describing String data elements in the MOST FBlock library.

Table 55 — Attributes for Strings

MOST FBlock library definition	Data type	Description
MaxSize	Unsigned Long	Maximum size of the string in characters

8.1.11 PL — Short Stream

REQ	6.38 PL – Data and basic data types – Short Stream attributes and values
The attributes and values for data type `Short Stream` shall be in accordance with Table 56.	

Table 56 — Short Stream

Attribute	Value
Name	`Short Stream`
Summary	Length-coded sequence of up to 255 bytes of unformatted data.
Size	1 byte length information + up to 255 data bytes = 256 bytes maximum.
Encoding	`Short Stream:` (`Length, Data`) `Length:` `Unsigned Byte`; specifies the length of the `Data` field in bytes. `Data:` up to 255 bytes; contains the sequence of unformatted data bytes.
Description	The `Short Stream` data type supports the same structuring mechanisms as the `Stream` data type, that is, stream cases and stream signals.

8.1.12 PL — Classified Stream

REQ	6.39 PL – Data and basic data types – Classified Stream attributes and values
The attributes and values for data type `Classified Stream` shall be in accordance with Table 57.	

Table 57 — Classified Stream

Attribute	Value
Name	`Classified Stream`
Summary	Length-coded sequence of up to 2^{16} - 1 bytes with media type information.
Size	2 bytes Length information + up to 2^{16} - 1 bytes for `MediaType` and `Data` combined = 2^{16} + 1 bytes maximum.
Encoding	`Classified Stream:` (`Length, MediaType, Data`) `Length:` `Unsigned Word`; specifies the length of the `MediaType` and `Data` fields in bytes. `MediaType:` Variable size; specifies the media type of the content in the `Data` field. `Data:` Variable size; contains the sequence of payload of the `Classified Stream`.
Description	The data type `Classified Stream` acts as a container for objects of different media types. The media type of the object that is transported in a `Classified Stream` is identified by the MediaType, which stores the media type as a null-terminated string according to ISO/IEC 6937[2] (without encoding identifier) containing the MIME type of the object. The format used for this is specified in the MIME Specification RFC 2045:1996, 5.1[6]. The values for <Type>, <Subtype>, and <Parameter> are specified by the Internet Assigned Numbers Authority (IANA). `MediaType := <Type> "/" <Subtype> * (";" <Parameter>)`

EXAMPLE Figure 38 shows the structure for `MediaType text/plain`.

	Length		MediaType											Content			
Byte #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Data	00_{16}	$0F_{16}$	t	e	x	t	/	p	l	a	i	n	00_{16}	M	O	S	T

Figure 38 — Classified stream example-text/plain

REQ	6.40 PL – Data and basic data types – Empty string in MediaType of Classified Stream
When `MediaType` is an empty string, `application/octet-stream` shall be assumed.	

EXAMPLE Figure 39 shows the structure for an empty string as `MediaType`.

	Length		MediaType	Content											
Byte #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Data	00_{16}	$0D_{16}$	00_{16}	< binary data >											

Figure 39 — Classified stream example-application/octet-stream

8.2 PL — Function classes

8.2.1 PL — Purpose

Functions are categorized into function classes. The function class determines restrictions regarding the messages of the function, for example data structure, permitted OPTypes, and error handling.

Some function classes have attributes, for example maximum size or number of elements. These attributes are specified in the MOST FBlock library. In the following subclauses, these are labelled as "MOST FBlock library definitions".

When describing function classes, properties and methods are differentiated. Properties are further distinguished into such with one parameter, and of such with multiple parameters in messages with the `OPTypes Set, SetGet`, and `Status`.

Incrementing and decrementing is only applicable for numeric data types.

REQ	6.41 PL – Function classes – Increment and Decrement only for numeric data types
The `OPTypes Increment` and `Decrement` shall not be used for data types other than integer numeric data types.	

8.2.2 PL — Properties with a single parameter

8.2.2.1 PL — Overview

Function classes for properties with a single parameter depend on the data type of the Value parameter in the following subclauses. The function classes in Table 58 exist in MOST.

In the MOST function catalogue, the `Value` parameter can have any name.

Table 58 — Classes of functions with a single parameter

Function Class	Data type
Switch	`Boolean`
Number	`Unsigned Byte, Signed Byte, Unsigned Word, Signed Word, Unsigned Long, Signed Long, Unsigned Long Long, Signed Long Long, Float, Double`
Text	`String, Length-coded String`
Enumeration	`Enum`
BoolField	`Unsigned Byte, Unsigned Word, Unsigned Long, Unsigned Long Long`
Container	`Stream, Short Stream, Classified Stream`
BitSet	`BitField`

8.2.2.2 PL — Function class Switch

REQ	6.42 PL – Function classes – Switch OPTypes and parameters
The `OPTypes` and parameters used in function class Switch shall be in accordance with Table 59.	

Table 59 — `OPTypes` and parameters for function class Switch

OPType	Parameters
`Set`	`Value`[a]
`Get`	`---`
`SetGet`	`Value`[a]
`Status`	`Value`[a]
`Error`	`ErrorCode, ErrorInfog`
[a] Value is of the data type `Boolean`.	

8.2.2.3 PL — Function class Number

The operations for incrementing and decrementing are designed for use with fixed-point numbers and not suitable for floating-point numbers.

REQ	6.43 PL – Function classes – No Increment and Decrement for floating-point data types
The `OPTypes Increment` and `Decrement` shall not be used for the Float and Double data types.	

REQ	6.44 PL – Function classes – Number OPTypes and parameters
The `OPTypes` and parameters used in function class Number shall be in accordance with Table 60.	

Table 60 — OPTypes and parameters for function class Number

OPType	Parameters
`Set`	`Value`[a]
`Get`	`---`
`SetGet`	`Value`[a]
`Increment`	`NSteps`[b]
`Decrement`	`NSteps`[b]
`Status`	`Value`[a]
`Error`	`ErrorCode, ErrorInfo`
[a] Value is of one of the following data types: `Unsigned Byte, Signed Byte, Unsigned Word, Signed Word, Unsigned Long, Unsigned Long Long, Signed Long, Signed Long Long, Float, Double`. [b] `NSteps` is of data type `Unsigned Byte`.	

8.2.2.4 PL — Function class Text

REQ	6.45 PL – Function classes – Text OPTypes and parameters
The `OPTypes` and parameters used in function class Text shall be in accordance with Table 61.	