IEC 62386-103

Edition 2.0 2022-11
COMMENTED VERSION

INTERNATIONAL STANDARD

colour inside

Digital addressable lighting interface –
Part 103: General requirements – Control devices

IEC 62386-103:2022-11 CMV(en)

IEC Secretariat
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
info@iec.ch
www.iec.ch

About the IEC
The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications
The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigendum or an amendment might have been published.

IEC publications search - webstore.iec.ch/advsearchform
The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee, …). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and once a month by email.

IEC Customer Service Centre - webstore.iec.ch/csc
If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: sales@iec.ch.

IEC Products & Services Portal - products.iec.ch
Discover our powerful search engine and read freely all the publications previews. With a subscription you will always have access to up to date content tailored to your needs.

Electropedia - www.electropedia.org
The world's leading online dictionary on electrotechnology, containing more than 22 300 terminological entries in English and French, with equivalent terms in 19 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC 62386-103

Edition 2.0 2022-11
COMMENTED VERSION

INTERNATIONAL STANDARD

colour inside

Digital addressable lighting interface – Part 103: General requirements – Control devices

INTERNATIONAL ELECTROTECHNICAL COMMISSION

ICS 29.140.50; 29.140.99

ISBN 978-2-8322-6104-0

Warning! Make sure that you obtained this publication from an authorized distributor.

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

DIGITAL ADDRESSABLE LIGHTING INTERFACE –

Part 103: General requirements – Control devices

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

This commented version (CMV) of the official standard IEC 62386-103:2022 edition 2.0 allows the user to identify the changes made to the previous IEC 62386-103:2014+ AMD1:2018 CSV edition 1.1. Furthermore, comments from IEC TC 34 experts are provided to explain the reasons of the most relevant changes, or to clarify any part of the content.

A vertical bar appears in the margin wherever a change has been made. Additions are in green text, deletions are in strikethrough red text. Experts' comments are identified by a blue-background number. Mouse over a number to display a pop-up note with the comment.

This publication contains the CMV and the official standard. The full list of comments is available at the end of the CMV.

IEC 62386-103 has been prepared by IEC technical committee 34: Lighting. It is an International Standard.

This second edition cancels and replaces the first edition published in 2014 and Amendment 1:2018. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

a) the scope has been updated;
b) quiescent mode has been updated;
c) non-volatile memory (NVM) save time has been added, and SAVE PERSISTENT VARIABLES command removed;
d) memory bank 0 has been modified, and common memory bank requirements have been added;
e) IDENTIFY DEVICE has been updated;
f) version number has been changed;
g) bus unit configuration has been added; and
h) instance types and configuration have been added.

The text of this International Standard is based on the following documents:

Draft	Report on voting
34/946/FDIS	34/990/RVD

Full information on the voting for its approval can be found in the report on voting indicated in the above table.

The language used for the development of this International Standard is English.

This document was drafted in accordance with ISO/IEC Directives, Part 2, and developed in accordance with ISO/IEC Directives, Part 1 and ISO/IEC Directives, IEC Supplement, available at www.iec.ch/members_experts/refdocs. The main document types developed by IEC are described in greater detail at www.iec.ch/standardsdev/publications.

This Part 103 of IEC 62386 is intended to be used in conjunction with Part 101, which contains general requirements for the relevant product type (system), and with the appropriate Parts 3xx (particular requirements for control devices) containing clauses to supplement or modify the corresponding clauses in Part 101 and Part 103 in order to provide the relevant requirements for each type of product.

A list of all parts in the IEC 62386 series, published under the general title *Digital addressable lighting interface*, can be found on the IEC website.

The committee has decided that the contents of this document will remain unchanged until the stability date indicated on the IEC website under webstore.iec.ch in the data related to the specific document. At this date, the document will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

IMPORTANT – The "colour inside" logo on the cover page of this document indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

INTRODUCTION

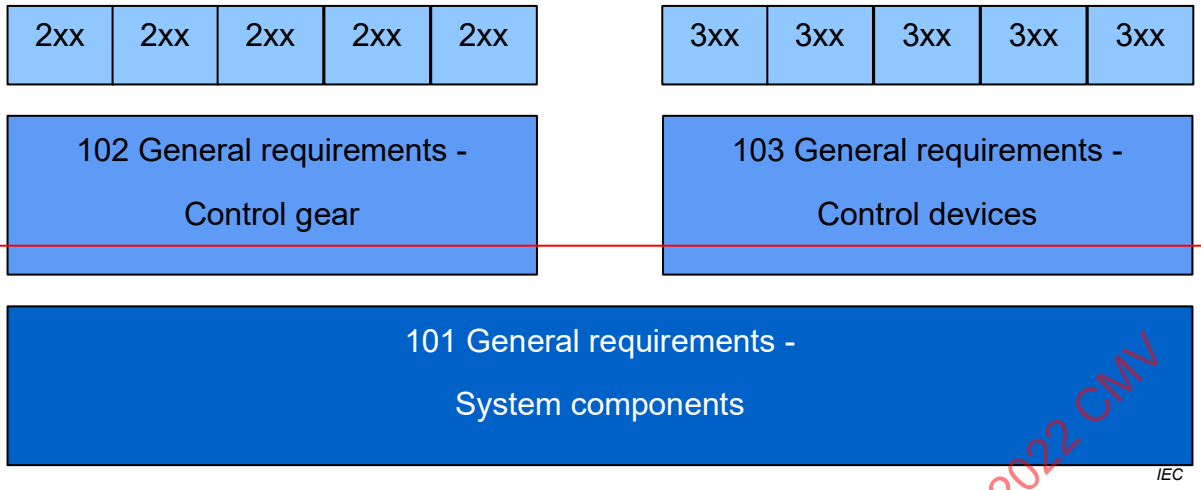
IEC 62386 contains several parts, referred to as series. The IEC 62386 series specifies a bus system for control by digital signals of electronic lighting equipment. The IEC 62386-1xx series includes the basic specifications. Part 101 contains general requirements for system components, Part 102 extends this information with general requirements for control gear and Part 103 extends it further with general requirements for control devices. Part 104 and Part 105 can be applied to control gear or control devices. Part 104 gives requirements for wireless and alternative wired system components. Part 105 describes firmware transfer. Part 150 gives requirements for an auxiliary power supply which can be stand-alone, or built into control gear or control devices.

The IEC 62386-2xx series extends the general requirements for control gear with lamp specific extensions (mainly for backward compatibility with Edition 1 of IEC 62386) and with control gear specific features.

The IEC 62386-3xx series extends the general requirements for control devices with input device specific extensions describing the instance types as well as some common features that can be combined with multiple instance types.

This ~~first~~ second edition of IEC 62386-103 is intended to be used in conjunction with IEC 62386-101~~:2014, IEC 62386-101:2014/AMD1:2018, IEC 62386-102:2014, IEC 62386-102:2014/AMD1:2018~~ and ~~with the various parts that make up the IEC 62386-2xx series for control gear, together~~ with the various parts that make up the IEC 62386-3xx series of particular requirements for control devices, and can be used together with IEC 62386-102 and with the various parts that make up the IEC 62386-2xx series for control gear. The division into separately published parts provides for ease of future amendments and revisions. Additional requirements will be added as and when a need for them is recognised.

The setup of the standards is graphically represented in Figure 1 below.



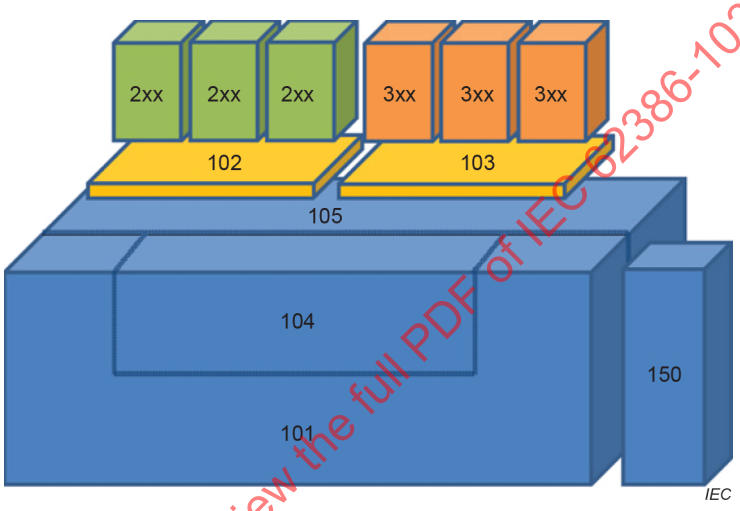


Figure 1 – IEC 62386 graphical overview 1

When this part of IEC 62386 refers to any of the clauses of the other ~~two~~ parts of the IEC 62386-1xx series, the extent to which such a clause is applicable ~~and the order in which the tests are to be performed are~~ is specified. The other parts also include additional requirements, as necessary.

All numbers used in this document are decimal numbers unless otherwise noted. Hexadecimal numbers are given in the format 0xVV, where VV is the value. Binary numbers are given in the format XXXXXXXXb or in the format XXXX XXXX, where X is 0 or 1, "x" in binary numbers means "don't care".

The following typographic expressions are used:

Variables: *variableName* or *variableName[3:0]*, giving only bits 3 to 0 of *variableName*;

Range of values: [lowest, highest];

Command: "COMMAND NAME".

DIGITAL ADDRESSABLE LIGHTING INTERFACE –

Part 103: General requirements – Control devices

1 Scope

This part of IEC 62386 is applicable to control devices ~~– in a bus system~~ for control by digital signals of electronic lighting equipment ~~which is in line with the requirements of IEC 61347 (all parts), with the addition of DC supplies~~.

~~NOTE Tests in this standard are type tests. Requirements for testing individual products during production are not included.~~

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.

IEC 62386-101:~~2014~~2022, *Digital addressable lighting interface – Part 101: General requirements – System components*
~~IEC 62386-101:2014/AMD1:2018~~

IEC 62386-102:~~2014~~2022, *Digital addressable lighting interface – Part 102: General requirements – Control gear*
~~IEC 62386-102:2014/AMD1:2018~~

IEC 62386-3xx *(all parts), Digital addressable lighting interface – Part 3xx: Particular requirements for control devices*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 62386-101 and the following apply.

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

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

3.1
broadcast
type of address used to simultaneously address all control devices in the system ~~at once~~

3.2
broadcast unaddressed
type of address used to simultaneously address all control devices in the system that have no short address ~~at once~~

3.3
device command
command which addresses the control device and has a value of 0xFE in the instance byte of the command frame but is not an event message

3.4
device group
type of address used to address a group of control devices in the system at once

3.5
DTR
data transfer register
multipurpose register used to exchange data

3.6
event
instance report, characterized by its event number, of a change or a defined sequence of changes of its input value

Note 1 to entry: The event number is specific to the type of instance that sends the report.

3.7
event scheme
characterisation of the information, as provided by an instance when producing an event message, that identifies the source of the event

3.8
feature
optional extension at instance and/or device level

3.9
feature command
command which addresses one or more features of an input device or device instance and has a value different from 0xFE in the instance byte of the command frame but is not an instance command or an event message

3.10
GTIN
global trade item number
number used for the unique identification of trade items worldwide

Note 1 to entry: For further information see http://en.wikipedia.org/wiki/GTIN.

Note 2 to entry: The global trade item number is comprised of a GS1 or U.P.C. company prefix followed by an item reference number and a check digit. It is described in the "GS1 General Specifications" (see [1]).

3.11
input signal
physical value that an instance of an input device is designed to detect and process

Note 1 to entry: Examples for physical values are ~~"light level"~~ "illuminance" and "button state".

3.12
identification
temporary state used during commissioning that allows the installer to identify particular control devices

3.13
input value
encoded data, representing the input signal

Note 1 to entry: The way in which the input signal is encoded depends on the instance type.

3.14
instance command
command which addresses one or more instances of an input device and has a value different from 0xFE in the instance byte of the command frame but is not a feature command or an event message

3.15
MASK
~~the value 0Xff~~
value with all binary digits set to 1

Note 1 to entry: This means that an 8-bit backward frame of MASK is a value of 0xFF, and a multi-byte memory location of 24 bits containing MASK is a value of 0xFFFFFF.

3.16
NO
answer ~~used to deny or refuse~~ to a query where no backward frame is sent

Note 1 to entry: If a query is asked where the answer is NO, there will be no response, such that the sender of the query will conclude "no backward frame" following IEC 62386-101:~~2014 and IEC62386-101:2014/AMD1:2018~~2022, 8.2.5.

Note 2 to entry: The answer NO could also be triggered by a missed query.

3.17
NVM
non-volatile memory
non-volatile read/write memory, the content of which can be changed and will not be lost due to a power cycle

3.18
NVM-RO
NVM that cannot be written using any command

3.19
NVM-RW
NVM that can be modified using one or more commands

3.20
opcode
operation code
that part of a command frame that identifies the command to be executed

3.21
operating mode
set of states identified by a number in the range [0,255], characterised by a collection of variables and memory settings, and used to select a set of functionalities to be exhibited by a control device, including its required reaction to commands

Note 1 to entry: Control devices ~~may~~ can support more than one operating mode.

3.22
PING
16-bit forward frame with bits [15:0] equal to 0xAD00

Note 1 to entry: As specified in IEC 62386-102, PING has no meaning for control gear.

3.23
quiescent mode
temporary mode in which the device does not send forward frames

3.24
RAM
volatile read/write memory, the content of which can be changed and will be lost due to a power cycle

3.25
RAM-RO
RAM that cannot be written using any command

3.26
RAM-RW
RAM that can be modified using one or more commands

3.27
random address
random 24-bit number generated by the control device on request during system initialisation

3.28
reset state
state in which all NVM variables of the control device have their reset value, except those that are marked "no change" or are otherwise explicitly excluded

3.29
ROM
non-volatile read-only memory, the content of which is fixed

Note 1 to entry: In this document read-only is meant from a system perspective. A ROM variable may actually be implemented in NVM, but this document does not provide any mechanism to change its value.

3.30
search address
24-bit number used to identify an individual control device in the system during initialisation

3.31
short address
type of address used to address an individual control device in the system

3.32
TMASK 2
value with the least significant bit set to 0 and all other binary digits set to 1

Note 1 to entry: This means that an 8-bit backward frame of TMASK is a value of 0xFE, and a multi-byte memory location of 24 bits containing TMASK is a value of 0xFFFFFE.

3.33
YES
answer ~~used to accept or affirm~~ to a query where a backward frame of MASK is sent

~~Note 1 to entry: If a query is asked where the answer is YES, the response will be a backward frame containing the value of MASK.~~

4 General

4.1 General

The requirements of IEC 62386-101:~~2014 and IEC62386-101:2014/AMD1:2018~~2022, Clause 4 apply, with the restrictions, changes and additions identified below.

4.2 Version number

This subclause replaces IEC 62386-101:~~2014 and IEC62386-101:2014/AMD1:2018~~2022, 4.2.

The version shall be in the format "x.y", where the major version number x is in the range of 0 to 62 and the minor version number y is in the range of 0 to 2. When the version number is encoded into a byte, the major version number x shall be placed in bits 7 to 2 and the minor version number y shall be placed in bits 1 to 0.

At each amendment to an edition of IEC 62386-103 the minor version number shall be incremented by one.

At a new edition of IEC 62386-103 the major version number shall be incremented by one and the minor version number shall be set to 0.

The current version number is "*versionNumber*" as defined in Table 19.

NOTE ~~Normally 2 amendments on IEC documents are made before a new edition is created.~~ IEC documents are generally subject to two amendments before a new edition is prepared.

5 Electrical specification

The requirements of IEC 62386-101:~~2014 and IEC 62386-101:2014/AMD1:2018~~2022, Clause 5 apply.

6 ~~Interface~~ Bus power supply

If a bus power supply is integrated into a control ~~gear~~ device, the requirements of IEC 62386-101:~~2014 and IEC62386-101:2014/AMD1:2018~~2022, Clause 6 apply.

7 Transmission protocol structure

7.1 General

The requirements of IEC 62386-101:~~2014 and IEC62386-101:2014/AMD1:2018~~2022, Clause 7 apply with the following additions.

7.2 24-bit forward frame encoding

7.2.1 Frame format for instructions and queries

7.2.1.1 General

For 7.2.1, commands shall be interpreted as instructions and queries, and exclude event messages 3. The 24-bit forward frame shall be encoded as shown in Table 1 and Table 2.

Table 1 – 24-bit command frame encoding

<table>
<tr><th colspan="8">Bytes/Bits</th><th rowspan="3">Device addressing</th></tr>
<tr><th colspan="7">Address byte</th><th>Instance byte</th><th>Opcode byte</th></tr>
<tr><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15...8</th><th>7...0</th></tr>
<tr><td>0</td><td colspan="6">64 short addresses</td><td>1</td><td rowspan="4">Device ~~command~~ or instance ~~address~~ or feature, see Table 2</td><td></td><td>Short addressing</td></tr>
<tr><td>1</td><td>0</td><td colspan="5">32 device group addresses</td><td>1</td><td></td><td>Device group addressing</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td></td><td>Broadcast unaddressed</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td></td><td>Broadcast</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td colspan="4">16 special command spaces</td><td>1</td><td colspan="2">Command specific</td><td>Special command</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>x</td><td>x</td><td>x</td><td>1</td><td colspan="2" rowspan="3">Reserved</td><td rowspan="3">Reserved</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>x</td><td>x</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>x</td><td>1</td></tr>
</table>

Table 2 – Instance byte in a command frame

<table>
<tr><th colspan="8">Instance byte</th><th rowspan="2">Addressing</th></tr>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>09</th><th>08</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td colspan="5">32 Instance numbers</td><td>Instance number</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td colspan="5">32 Instance groups</td><td>Instance group</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td colspan="5">32 Instance types</td><td>Instance type</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td colspan="5">32 Instance numbers</td><td>Feature on instance number level</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td colspan="5">32 Instance groups</td><td>Feature on instance group level</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td colspan="5">32 Instance types</td><td>Feature on instance type level</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>Feature broadcast</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>Feature on instance broadcast level</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>Instance broadcast</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>Feature on device level</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>Device</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td rowspan="5">Reserved</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>x</td><td>x</td><td>x</td><td>x</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>x</td><td>x</td><td>x</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>x</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
</table>

7.2.1.2 Address byte

The address byte provides

- the method of device addressing used by the transmitter;
- the indication that a command, not an event message, is being transmitted: bit 16 is set for commands;
- 16 special command spaces;
- reserved device addresses. Reserved addresses shall not be used by the transmitter.

7.2.1.3 Instance byte

The instance byte provides

- for standard commands, the indication of whether a device command, feature command or an instance command is being transmitted;
- for standard instance commands, the method of instance addressing used by the transmitter;
- command-specific information for special commands;
- for standard commands, reserved instance addresses. Reserved instance addresses shall not be used by the transmitter;
- for standard feature commands, the feature that is being addressed;
- reserved information for reserved commands.

7.2.1.4 Opcode byte

The opcode byte provides

- for standard commands, the opcode;
- command-specific information for special commands;
- reserved information for reserved commands.

7.2.2 Frame format for event messages

7.2.2.1 General

For event messages, the 24-bit forward frame shall be encoded as shown in Table 3.

Table 3 – 24-bit event message frame encoding

<table>
<tr><th colspan="14">Bits</th><th colspan="2" rowspan="2">Event scheme[a] / Source</th></tr>
<tr><th colspan="13">Event source information</th><th>Event information</th></tr>
<tr><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9...0</th><th></th><th></th></tr>
<tr><td>0</td><td colspan="6">64 short addresses</td><td>0</td><td>0</td><td colspan="5">32 instance types</td><td rowspan="5">Event</td><td>1</td><td>Device</td></tr>
<tr><td>0</td><td colspan="6">64 short addresses</td><td>0</td><td>1</td><td colspan="5">32 instance numbers</td><td>2</td><td>Device and instance</td></tr>
<tr><td>1</td><td>0</td><td colspan="5">32 device groups</td><td>0</td><td>0</td><td colspan="5">32 instance types</td><td>3</td><td>Device group</td></tr>
<tr><td>1</td><td>0</td><td colspan="5">32 instance types</td><td>0</td><td>1</td><td colspan="5">32 instance numbers</td><td>0</td><td>Instance</td></tr>
<tr><td>1</td><td>1</td><td colspan="5">32 instance groups</td><td>0</td><td>0</td><td colspan="5">32 instance types</td><td>4</td><td>Instance group</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>x</td><td>x</td><td>x</td><td>x</td><td>0</td><td>1</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td rowspan="8">Reserved</td><td colspan="2" rowspan="8">Reserved</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>x</td><td>x</td><td>x</td><td>0</td><td>1</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>x</td><td>x</td><td>0</td><td>1</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>x</td><td>0</td><td>1</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>!</td><td>1</td><td>0</td><td>0</td><td>1</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>!</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>x</td><td>x</td><td>x</td><td>x</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>x</td><td>x</td><td>x</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td colspan="4">Short address and device group information, refer to 9.7.2.</td><td colspan="2">Device power cycle</td></tr>
<tr><td colspan="16">[a] Refer to 9.7.3 for further information on event schemes.</td></tr>
</table>

7.2.2.2 Event source information

The event source information in Table 3 provides:

- the indication that an event message, not an instruction or query, is being transmitted: bit 16 ~~is~~ shall be clear for event messages;
- relevant event instance type information, such that the receiver of an event message will be able to understand the meaning of the event;
- relevant event source information, such that the receiver of an event message ~~may be~~ is able to understand where the message is coming from;
- reserved values.

Events are instance type specific. ~~This means that the event source information must be such that a receiver can derive – either explicitly or implicitly – the instance type of the transmitting instance. The identified event source schemes in Table 3 (and only these) satisfy this condition.~~ Application controllers can derive, either explicitly or implicitly, the instance type of the transmitting instance, from the event source information.

NOTE The event source schemes are not equally valuable in terms of telling the receiver where the event message originated.

7.2.2.3 Event information

The event information provides the 10-bit event number and/or event data. Event information is instance type specific and is defined in the applicable parts of the IEC 62386-3xx series that describe the instance type.

8 Timing

The requirements of IEC 62386-101:~~2014 and IEC62386-101:2014/AMD1:2018~~2022, Clause 8 apply.

9 Method of operation

9.1 General

The requirements of IEC 62386-101:~~2014 and IEC62386-101:2014/AMD1:2018~~2022, Clause 9 apply with the following additions.

9.2 Device features

The IEC 62386 series allows for the future publication of feature extensions that extend the requirements in this document, or exempt particular requirements.

The features for a device can be queried by "QUERY FEATURE TYPE" and "QUERY NEXT FEATURE TYPE" while the instance byte is set to "feature on device level" (see Table 2).

Table 5 shows the feature type encoding. For further information on the different feature types see parts of the IEC 62386-3xx series.

9.3 Application controller

9.3.1 General

An application controller is that part of a control system that makes the system "work":

- ~~it is~~ an application controller ~~that commissions~~ can commission and configure~~s~~ the system (including available control gear);
- ~~it is~~ an application controller ~~that makes~~ can make the system react to changes in the environment (based on information coming from input devices);
- ~~it is~~ an application controller ~~that changes~~ can change the system response of control gear in the system (possibly using any command defined in IEC 62386-102:~~2014 and IEC 62386-102:2014/AMD1:2018~~2022).

9.3.2 Single-master application controller

A single-master application controller is not intended to share the bus with other control devices.

A single-master application controller ~~may~~ can try to configure other control devices on the bus, and/or change the system response of control gear in the system, thereby using any command defined in IEC 62386-102~~:2014 and IEC 62386-102:2014/AMD1:2018~~ and/or instructions and queries defined in ~~IEC 62386-103:2014 and IEC 62386-103:2014/AMD1:2018~~ this document.

NOTE Especially if the single-master application controller does not handle collisions appropriately, any such attempt ~~may~~ can fail and affect the system negatively.

On the other hand, a single-master application controller is not required to have a receiver on board. For this reason, the following holds:

~~For all following subclauses~~ From 9.3.3 onwards, this document assumes a control device to be a multi-master control device.

In order to make itself known as a possibly anonymous transmitting bus unit, a single-master application controller shall transmit a PING message at regular intervals of 10 min ± 1 min. The first such PING message shall appear at a random time between 5 min and 10 min after completion of the power-on procedure.

9.3.3 Multi-master application controller

~~For all following subclauses~~ From 9.3.3 onwards, this document assumes a control device to be a multi-master control device.

A control device that includes an application controller shall have *"applicationControllerPresent"* set to TRUE. *"applicationControllerPresent"* shall be set to FALSE otherwise.

NOTE 1 *"applicationControllerPresent"* can be observed through "QUERY DEVICE CAPABILITIES".

In most cases, a system will have only one application controller active (refer to 9.10.1), but multiple application controllers can be operational in a single system.

An application controller shall accept commands (from other application controllers) according to Table 23 and Table 24. It is part of the system integration to ensure that the application controllers will ~~do this~~ accept commands in such a way that it results in a correctly functioning system ~~results~~.

NOTE 2 System integrity is easiest to achieve by allowing only a single application controller to do commissioning and configuration.

NOTE 3 An application controller ~~might~~ can be commissioned through alternative interfaces.

An application controller shall not transmit event messages other than for the device power cycle event.

NOTE 4 If an application controller is active, it can send 24-bit forward frames for purposes other than transmitting events.

An application controller shall not transmit PING messages.

9.4 Input device

Input devices make a system sensitive to changes in its environment, by transmitting event messages.

Input devices shall be multi-master control devices and shall allow commissioning and configuration by an application controller.

Input devices shall use forward frames only to transmit event messages.

9.5 Instances of input devices

9.5.1 General

An input device shall have at least one instance and a maximum of 32 instances, as shall be indicated by *"numberOfInstances"*, which can be queried using "QUERY NUMBER OF INSTANCES".

A control device that is only an application controller shall have a *"numberOfInstances"* equal to 0.

9.5.2 Instance number

Each instance shall have a unique *"instanceNumber"* in the range [0,*"numberOfInstances"* – *1*].

9.5.3 Instance type

The instance type for each of the instances of an input device can be different. It can be queried by "QUERY INSTANCE TYPE" while the instance byte is set to either "instance type" or " instance number" scheme (see Table 2). The meaning of event information transmitted by means of "INPUT NOTIFICATION (*device/instance, event*)" depends on the instance type.

Table 4 shows the instance type encoding. For further information on the different instance types see the relevant parts of the IEC 62386-3xx series.

Table 4 – Instance types

Instance type	IEC 62386	Used for
0	Part 103	Generic purpose, input devices that are not defined. Another method of identifying the device shall be implemented, to allow application controller to interpret the events.
1 to 31	Part 301 to Part 331	These IEC 62386-3xx parts describe instance types, where xx ranges from ~~1~~ 01 to 31.

9.5.4 Instance features

The IEC 62386 series allows for the future publication of feature extensions that extend the requirements in this document, or exempt particular requirements.

The features for each of the instances of an input device can be different. They can be queried by “QUERY FEATURE TYPE” and “QUERY NEXT FEATURE TYPE” while the instance byte is set to either “feature on instance type level” or “feature on instance number level” scheme (see Table 2).

Table 5 shows the feature type encoding. For further information on the different feature types see the relevant parts of the IEC 62386-3xx series.

Table 5 – Feature types

Feature type	IEC 62386	Used for
32 to 96	Part 332 to Part 396	These IEC 62386-3xx parts describe feature extensions, where xx ranges from 32 to 96.

9.5.5 Instance groups

Instance groups are a means for an application controller to put instances into logical groups, across input devices. Consequently, these logical groups can be used to configure multiple instances at once.

An application controller can use up to 32 such groups, numbered in the range [0,31]. Each instance can be declared to be a member of up to 3 instance groups and shall expose instance group variables as given in Table 6.

Table 6 – Instance group variables

Variable	Description
"instanceGroup0"	Primary instance group number, MASK if no membership defined.
"instanceGroup1"	Additional instance group number, MASK if no membership defined.
"instanceGroup2"	Additional instance group number, MASK if no membership defined.

Instance groups are assigned and queried by using the following instance operations:

- "SET PRIMARY INSTANCE GROUP (*DTR0*)", "QUERY PRIMARY INSTANCE GROUP";
- "SET INSTANCE GROUP 1 (*DTR0*)", "QUERY INSTANCE GROUP 1";
- "SET INSTANCE GROUP 2 (*DTR0*)", "QUERY INSTANCE GROUP 2".

The primary group is special in the sense that only this number shall be used when reporting events (if instance group event reporting is used). Additional groups are a means of configuring multiple instances at once.

9.6 Commands excluding event messages 4

9.6.1 General

A control device shall check the device addressing scheme to see if it is addressed by a command. The control device shall accept the command, unless any of the following conditions hold:

- the command is sent using short addressing, and the given short address is not equal to *"shortAddress"*;
- the command is sent using device group addressing, and the given device group does not match any of the groups identified by *"deviceGroups"*;
- the command is sent using broadcast unaddressed addressing and *"shortAddress"* is not MASK;
- the command is sent using reserved addressing;

- the command is not defined;
- the command is sent using feature addressing, and the given feature is not implemented.

NOTE For instance and feature commands, additional conditions for command acceptance hold. These are given in 9.6.3 and 9.6.4.

9.6.2 Device commands

The instance byte shall be 0xFE for device commands. If the instance byte is not equal to 0xFE, the control device shall not accept these commands.

NOTE This addressing mechanism allows the opcode values for device commands and instance commands to overlap.

9.6.3 Instance commands

For instance commands that are accepted by an input device (refer to 9.5), the instance addressing scheme determines the intended (set of) receiving instances within that device. An instance shall accept the instance command, unless any of the following additional conditions hold:

- the command is sent using instance number addressing and the given instance number is not equal to *"instanceNumber"*;
- the command is sent using instance group addressing, and the given instance group does not match any of the groups identified by *"instanceGroup0"*, *"instanceGroup1"* and *"instanceGroup2"* (see Table 6);
- the command is sent using instance type addressing and the given instance type is not equal to *"instanceType"*;
- the command is sent using reserved addressing.

9.6.4 Feature commands

For feature commands that are accepted by a device or instance (refer to 9.5), the feature addressing scheme determines the intended (set of) receiving features within that device (see Table 2).

A feature on device level shall accept the feature command unless any of the following additional conditions hold:

- the command is sent using feature addressing other than feature addressing on the device level;
- the command is sent with instance byte reserved addressing.

A feature on instance level shall accept the feature command unless any of the following additional conditions hold:

- the command is sent using feature on instance number level addressing and the given instance number is not equal to "*instanceNumber*";
- the command is sent using feature on instance group level addressing and the given instance group does not match any of the groups identified by "*instanceGroup0*", "*instanceGroup1*" and "*instanceGroup2*";
- the command is sent using feature on instance type level addressing and the given instance type is not equal to "*instanceType*";
- the command is sent using feature on device level addressing;
- the command is sent with instance addressing;
- the command is sent with device addressing;
- the command is sent with instance byte reserved addressing.

A feature broadcast command shall be used to address a given feature both on device level and instance level. A feature broadcast shall be accepted unless any of the following additional conditions hold:

- the command is sent using feature addressing other than feature broadcast;
- the command is sent with instance byte reserved addressing.

If the feature command is accepted, the opcode byte determines which feature is addressed.

9.7 Event messages

9.7.1 Response to event messages

An application controller ~~or input device~~ is free to act upon reception of any event message or to ignore the message.

NOTE ~~If an application controller or input device is disabled, it is not allowed to send any response~~ A disabled application controller does not send any forward frames, but can still update its internal state based on messages received.

9.7.2 Device power cycle event

Since the power cycle event (see 9.13.2) is a device event, it does not adhere to the default event frame format. Bits 12 through 0 shall carry device address information as indicated in Table 7.

Table 7 – Device address information in power cycle event

Bits												
12	11	10	09	08	07	06	05	04	03	02	01	00
1 = device group valid	Lowest device group					1 = short address valid	Short address					

Bit 12 shall be set if and only if the transmitting control device is member of at least one device group. Bits [11:7] shall indicate the lowest device group number of membership in that case. If bit 12 is not set, bits [11:7] shall be clear.

Bit 6 shall be set if and only if the transmitting control device has a *"shortAddress"* different from MASK. Bits [5:0] shall indicate the device short address in that case. If bit 6 is not set, bits [5:0] shall be clear.

9.7.3 Input notification event

An instance of an input device shall, when transmitting an event message, use the selected event source addressing scheme as defined in Table 8.

Table 8 – Event addressing schemes

"eventScheme"	Description
0 (default)	Instance addressing, using instance type and number.
1	Device addressing, using short address and instance type.
2	Device~~/~~ and instance addressing, using short address and instance number.
3	Device group addressing, using device group and instance type.
4	Instance group addressing, using instance group and type.

An application controller can set and query the *"eventScheme"* by means of "SET EVENT SCHEME (*DTR0*)" and "QUERY EVENT SCHEME" respectively.

NOTE 1 An instance can only implement an event scheme while certain conditions have been satisfied by the application controller as well. Instance addressing is the only addressing scheme that will work under all circumstances.

In the following situations, the instance shall immediately revert *"eventScheme"* to the default instance addressing scheme shown in Table 8:

- *"eventScheme"* has been set to 1 or 2 whereas the containing ~~device~~ logical unit has no short address;
- *"eventScheme"* has been set to 3 whereas the containing ~~device~~ logical unit is not a member of a device group;
- *"eventScheme"* has been set to 4 whereas the instance has no primary instance group membership (see 9.5.5).

NOTE 2 The above situations can occur because of a new "SET EVENT SCHEME (*DTR0*)" command and/or because of a change of conditions.

~~Once reverted to the default event scheme:~~

- ~~"QUERY EVENT SCHEME" shall reflect this.~~
- ~~Only a new "SET EVENT SCHEME (*DTR0*)" command may change the actual event scheme.~~ **5**

NOTE 3 This implies that the command "SET EVENT SCHEME (*DTR0*)" can "fail", rather than ~~that it expresses~~ setting a preference that ~~may be~~ is granted ~~sooner or~~ sometime later. ~~The application controller is recommended to~~ To avoid this failure, application controllers can set the desired event scheme ~~only~~ after completing those configuration aspects that influence event scheme operation.

Furthermore, and given a viable addressing scheme, the instance shall

- only refer to *"instanceNumber"* as the instance number;
- only refer to *"instanceType"* as the instance type;
- only refer to *"instanceGroup0"* as the instance group;
- only refer to *"shortAddress"* as the ~~containing device~~ short address of the containing logical unit;
- only refer to the lowest device group number ~~of membership of the containing device~~ that the containing logical unit is a member of.

9.7.4 Event message filter

The event message filter can be used to enable and disable specific events. While the event filter of a specific event is disabled, this specific event shall not be generated. To enable or disable all events see 9.10.3.

An application controller can set the *"eventFilter"* by means of SET EVENT FILTER (*DTR2:DTR1:DTR0*) and can query the variable by means of QUERY EVENT FILTER 0-7, QUERY EVENT FILTER 8-15 and QUERY EVENT FILTER 16-23 respectively.

The relevant parts of the IEC 62386-3xx series shall define the meaning of the bits in "*eventFilter*", and can reduce the width of the variable *"eventFilter"* if needed. If the width is reduced to 2 bytes, DTR2 shall be ignored for SET EVENT FILTER (*DTR2:DTR1:DTR0*) and QUERY EVENT FILTER 16-23 shall answer NO. Similarly, if the width is reduced to 1 byte, DTR1 shall additionally be ignored for SET EVENT FILTER (*DTR2:DTR1:DTR0*) and QUERY EVENT FILTER 8-15 shall also answer NO.

9.8 Input signal ~~and input~~, measured value and "*inputValue*" 6

9.8.1 General

An instance shall process its input signal into ~~an input~~ a measured value and expose this value to the system, as described in 9.8.2 to 9.8.4.

9.8.2 Input resolution

The processing shall be done with a precision which is indicated by *"resolution"*. The actual resolution used for a particular instance (type) can be subject to requirements in relevant parts of the IEC 62386-3xx series and/or manufacturer choice.

~~The result of the conversion shall be available~~ The measured value shall be contained in the N-byte variable *"inputValue"*, where N is the minimum number of bytes needed to contain at least *"resolution"* bits.

NOTE N is computed as (*"resolution"*/8) rounded up to the nearest integer. With *"resolution"* in the range [1,255], *"inputValue"* can span up to 32 bytes.

~~The result of the conversion and the *"inputValue"* shall be MSB-aligned.~~ The measured value shall be most significant bit (msb)-aligned in *"inputValue"*. Unused bits in *"inputValue"* shall contain a repeating pattern of the most significant bit(s) of the ~~result of the conversion~~ measured value.

Table 9 provides an example, which shows a ~~signal level~~ measured value of ~~just below~~ almost 50 % latched into a 1-byte *"inputValue"* after being processed with a *"resolution"* of 3, 4 and 5 bits respectively.

Table 9 – ~~Signal level (~50%) versus resolution and input value~~
Measured value (≈ 50 %) versus resolution and "*inputValue*"

		Bits								
Resolution	**~~Signal level~~ Measured value**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**~~Input value~~ "*inputValue*"**
3-bits	3 of [0, 7]	0	1	1	0	1	1	0	1	109
4-bits	7 of [0, 15]	0	1	1	1	0	1	1	1	119
5-bits	15 of [0, 31]	0	1	1	1	1	0	1	1	123
NOTE The grey shaded bits are (part of) the (first) repetition of the significant bits.										

This method allows an application controller to interpret the input value correctly as an 8-bit value, regardless of the actual instance resolution or sensor precision. The minimum value of all bytes in *"inputValue"* is always 0, the maximum value 0xFF, for all resolutions. The relative ~~signal level~~ measured value corresponds (albeit with variable accuracy) to the relative input value.

9.8.3 Getting the input value

An instance shall support a latching mechanism that allows an application controller to obtain a consistent multi-byte input value. An example of such latching scenario is given in Table 10.

Application controllers ~~must~~ can start reading a multi-byte value by sending the command "QUERY INPUT VALUE". This command shall trigger a latch that contains a copy of *"inputValue"* such that the remaining bytes can be read using a sequence of "QUERY INPUT VALUE LATCH" queries. After having returned the last byte of the latch, the instance shall not answer further "QUERY INPUT VALUE LATCH" queries until after the next "QUERY INPUT VALUE".

Table 10 – Example of querying sequence to read a 4-byte input value

Input signal	*"inputValue"*	Command	Answer	Latched *"inputValue"*
"12340000"	0x12340000	…	…	unspecified
"12345678"	0x12345678"	"QUERY INPUT VALUE"	0x12	0x12345678
"852"	0x00000852	"QUERY INPUT VALUE LATCH"	0x34	0x12345678
"124852"	0x00124852	"QUERY INPUT VALUE LATCH"	0x56	0x12345678
"124852"	0x00124852	"QUERY INPUT VALUE LATCH"	0x78	0x12345678
"124852"	0x00124852	"QUERY INPUT VALUE LATCH"	NO	0x12345678

The *"inputValue"* that is latched is the *"inputValue"* at the moment "QUERY INPUT VALUE" is executed.

NOTE 1 This implies that if an application controller queries the *"inputValue"* because of an event message it has just received, the value obtained is not necessarily the same value that triggered the event.

The latched value shall be updated only when the next "QUERY INPUT VALUE" is executed. If the application controller uses "QUERY INPUT VALUE LATCH" without having used "QUERY INPUT VALUE" as the command before this one, the answer ~~may~~ can contain old or invalid data.

~~The application controller shall transmit the necessary queries for this scenario within a transaction.~~

~~NOTE 2 Using a transaction prevents concurrent access to the latched data.~~

~~An application controller may exit the scenario at any point.~~

NOTE 2 To prevent concurrent access to the latched data, an application controller can transmit the necessary queries for this scenario within a transaction, exiting the scenario at any point.

NOTE 3 If an application controller can work sufficiently accurately with 16-bit input for the given instance type, it can stop after having received the most significant 16 bits of input value, and handle those bits as if they were delivered by an instance with *"resolution"* equal to 16. This allows straightforward resolution-independent algorithm implementation.

9.8.4 Notification of changes

A change or a sequence of changes in the input signal of an instance shall result in an event message as required by this document or that part of the IEC 62386-3xx series that describes the *"instanceType"* (see 9.5.3) of that instance.

The event message shall be sent using "INPUT NOTIFICATION (*device/instance, event*)", as described in 11.3.1.

NOTE The manufacturer of the input device ~~should~~ can ensure that no event is lost by providing a queue. Parts of the IEC 62386-3xx series ~~may~~ can impose additional restrictions, e.g. to avoid event flooding.

9.9 System failure

An application controller should detect system failure and recovery. Preferably, it should act upon any bus power failure with a duration longer than 40 ms, thus anticipating a power cycle of bus powered devices.

NOTE Bus powered devices ~~may shutdown at~~ can shut down due to a power outage of more than 40 ms.

Next, when the system failure is resolved, the application controller should ensure that the system resumes normal operation.

9.10 Operating a control device

9.10.1 Enable/disable the application controller

If present, the application controller is either active or not-active, as shall be reflected by *"applicationActive"*. While deactivated, the application controller shall not send any forward frames, except possibly a power cycle notification (see 9.13.2).

"applicationActive" shall have no influence on the response to incoming forward transmissions, including the transmission of backward frames following queries.

NOTE This allows the application controller to monitor the bus, but the application controller cannot use forward frames to react.

"applicationActive" shall be stored in the NVM of the application controller. The default value shall be TRUE in case there is an application controller present, which can be changed by another application controller using the commands ENABLE APPLICATION CONTROLLER and DISABLE APPLICATION CONTROLLER. "*applicationActive*" can be queried using "QUERY APPLICATION CONTROLLER ENABLED".

9.10.2 Application controller always active

If an application controller is present it ~~may~~ can be always active. This shall be reflected by "*applicationControllerAlwaysActive*" being TRUE.

When "*applicationControllerAlwaysActive*" is TRUE, "*applicationControllerPresent*" and "*applicationActive*" shall always be TRUE.

"*applicationControllerAlwaysActive*" can be observed through "QUERY DEVICE CAPABILITIES" and "QUERY APPLICATION CONTROLLER ALWAYS ACTIVE".

9.10.3 Enable/disable event messages

Event messages are either enabled or disabled, as shall be reflected by *"instanceActive"*. While deactivated, the instance shall not send any forward frames. That is, the instance will not produce any event messages.

"instanceActive" shall have no influence on the response to incoming forward transmissions, including the transmission of backward frames following queries.

"instanceActive" shall be stored in the persistent memory of the input device. The default value shall be TRUE, which can be changed by an application controller using the commands "ENABLE INSTANCE" and "DISABLE INSTANCE". "*instanceActive*" can be queried using "QUERY INSTANCE ENABLED".

To limit the event messages when enabled, filtering is also available, see 9.7.4.

NOTE Queries are the only way to get information from an instance when event messages are disabled.

9.10.4 Quiescent mode

In quiescent mode, the control device shall not produce any forward frames except as a possible result of execution of SEND TESTFRAME **7**. No commands (see also 9.10.1), and no event messages (see also 9.10.3) shall be transmitted, regardless of *"applicationActive"* or any *"instanceActive"*. Event messages shall not be queued and shall not be set as pending. This means that such events are discarded.

Quiescent mode is a temporary mode which is started or restarted with the command "START QUIESCENT MODE". It ~~ends~~ shall end automatically 15 min ± 1,5 min after the last

"START QUIESCENT MODE" command was executed. Additionally, the command "STOP QUIESCENT MODE" shall terminate quiescent mode immediately.

In quiescent mode, a control device shall still respond to commands. "QUERY QUIESCENT MODE" can be used to determine whether or not a control device is in quiescent mode.

At power on of the control device, quiescent mode shall be DISABLED.

NOTE 1 Quiescent mode can be used by the application controller during initialisation (see 9.15) to ensure that random address comparisons are not frustrated by forward frames from other devices on the bus.

NOTE 2 Quiescent mode works independently from *"applicationActive"* and *"instanceActive"*. This implies that ending quiescent mode does not necessarily enable forward frame transmissions.

9.10.5 Modes of operation

9.10.5.1 General

Different operating modes can be selected at device level by means of command "SET OPERATING MODE (*DTR0*)". The currently selected *"operatingMode"* can be queried by means of "QUERY OPERATING MODE".

Operating modes 0x00 to 0x7F are defined in this document. At least operating mode 0x00 shall be available. Operating modes 0x80 to 0xFF are manufacturer specific. The query "QUERY MANUFACTURER SPECIFIC MODE" can be used to determine whether the control device is in an IEC 62386 standard operating mode, or in a manufacturer-specific mode.

9.10.5.2 Operating mode 0x00: standard mode

If a device is in *"operatingMode"* 0x00, its behaviour shall be as is required by this document, until it is set in an operating mode different from 0x00.

9.10.5.3 Operating mode 0x01 to 0x7F: reserved

Operating modes 0x01 to 0x7F are reserved and shall not be used.

9.10.5.4 Operating mode 0x80 to 0xFF: manufacturer-specific modes

Manufacturer-specific modes should only be used if the features required by the application are not covered by the IEC 62386 series. If a control device is in a manufacturer-specific operating mode, the behaviour of the control device may be manufacturer specific as well, with the following exceptions:

- ~~as far as the control device accesses the bus, it shall adhere to IEC 62386-101:2014 and IEC 62386-101:2014/AMD1:2018.~~
- with regard to bus access, the control device shall adhere to IEC 62386-101:2022.
- the control device shall adhere to this document at least as far as the following commands are concerned:
  - "SET OPERATING MODE (*DTR0*)", and "QUERY OPERATING MODE" and "QUERY MANUFACTURER SPECIFIC MODE".
  - All special commands (see 11.10) except WRITE MEMORY LOCATION (*DTR1, DTR0, data*), WRITE MEMORY LOCATION – NO REPLY (*DTR1, DTR0, data*) and DIRECT WRITE MEMORY (*DTR1, offset, data*).

For the above commands, ~~the various addressing methods shall apply, see 7.2.1.2~~ the requirements of 7.2.1 shall apply.

It is recommended that even in manufacturer-specific modes, the commands as specified in all implemented parts of the IEC 62386 series still be obeyed.

9.11 Memory banks 8

9.11.1 General

Memory banks are freely accessible memory spaces ~~defined for e.g. identification~~ containing information or configuration settings of the control device ~~in a system~~. Not all consecutive memory banks need to be implemented. Also within a memory bank not all consecutive locations need to be implemented. All implemented memory bank locations of all implemented memory banks are readable using memory access commands. Part of the memory is read-only and either programmed by the manufacturer of the control device or updated by the control device itself. For all other parts, write access using memory access commands can be enabled by the manufacturer. ~~Write access to a memory bank location can be locked. memory banks can be implemented using RAM, ROM or NVM.~~ Except for location 0x02, all writable locations of a memory bank are lockable unless specified in the corresponding memory bank table. Locations within the memory banks shall be implemented using the memory types shown in Table 11.

Table 11 – Memory types 9

Memory type	Accessibility via the bus[a]	Volatility[b]	May be changed autonomously by the control device during run time	Description
ROM	RO	NV	No	For all fixed values that will not change during run time of the control device. NOTE ROM is read-only by its nature, but can be changed during production programming.
RAM-RO	RO	V	Yes	For values that will not be retained through a power-cycle. Cannot be changed over the interface.
RAM-RW	RW	V	Yes	For values that will not be retained through a power-cycle. Can be changed over the interface
NVM-RO	RO	NV	Yes	For values that will be retained through a power-cycle. Cannot be changed over the interface.
NVM-RW	RW	NV	Yes	For values that will be retained through a power-cycle. Can be changed over the interface.

[a] RO: Read-only, RW: Read-write.

[b] V: volatile (not retained through a power-cycle). NV: non-volatile (retained through a power-cycle).

The addressable memory space is limited to a maximum of ~~almost 64 kBytes, organized in maximum~~ 256 memory banks of maximum 255 bytes each (approximately 64 kBytes). As this document ~~prescribes~~ specifies how to implement memory bank 0 and 1 (if present), and reserves memory banks 200 to 255, this leaves room for 198 memory banks for manufacturer-specific purposes in the range of [2,199].

9.11.2 Memory map

If a manufacturer-specific memory bank in the range of [2,199] is implemented, allocation of its content shall comply with the memory map provided in Table 12.

Table 12 – Basic memory map of memory banks

Address	Description	Default value (factory)	Reset value[b]	Memory type
0x00	Address of last accessible memory location	factory burn-in, range [0x03,0xFE]	no change	ROM
0x01	Indicator byte[a]	[a]	[a]	any[a]
0x02	Memory bank lock byte. Lockable bytes in the memory bank shall be read-only while the lock byte has a value different from 0x55.	0xFF	0xFF[c]	RAM-RW
[0x03,0xFE]	Memory bank content[a]	[a]	[a]	any[a]
0xFF	Reserved – not implemented	answer NO	no change	n.a.

[a] The purpose, default~~/~~ values, power on~~/~~ values, reset values, and memory ~~access~~ types of these bytes shall be defined by the manufacturer.

[b] Reset value after "RESET MEMORY BANK".

[c] Also used as power on value unless explicitly stated otherwise.

The byte in location 0x00 of each bank contains the address of the last accessible memory location of the bank. The value shall be in the range [0x03,0xFE].

The byte in location 0x01 is manufacturer specific. If implemented, the usage of this byte should be described by the manufacturer (as well as the entire content of the memory bank).

NOTE 1 It could be used for example to store a checksum in case of a memory bank with static content. Using a checksum on a memory bank where the content is changed by the control device is not useful.

The byte in location 0x02 shall be used to lock write access. Memory location 0x02 itself shall never be locked for writing. While this memory location contains any value different from 0x55, all memory locations marked "(lockable)" of the corresponding memory bank shall be read only. The control device shall not change the value of the lock byte other than as a consequence of a power cycle or of a "RESET MEMORY BANK (*DTR0*)" command or other command affecting the lock byte.

Location 0xFF is a reserved location in every memory bank, and is not accessible. This location shall not be implemented as a normal memory bank location. When addressed, the control device shall respond as if this location is not implemented, and it shall not increment "*DTR0*".

NOTE 2 This location is reserved in order to stop the auto increment of "*DTR0*".

9.11.3 Selecting a memory bank location

In order to select a memory bank location a combination of memory bank number and location inside the memory bank is required.

The memory bank shall be selected by setting the memory bank number in *"DTR1"*. The location in the memory bank shall be selected by the value in *"DTR0"*.

9.11.4 Protectable memory locations 10

Memory bank locations marked as "protectable" can have read or write protection applied or removed using a manufacturer-specific method.

Protectable locations with read protection enabled shall reply MASK as a result of "READ MEMORY LOCATION (*DTR1, DTR0*)". In this case, the implementation or provision of the related feature is optional.

NOTE 1 Read protection can be enabled by default. If read protection is enabled, this means that the related feature is possibly not implemented and there is no obligation for the manufacturer to provide a mechanism to deactivate read protection.

Protectable locations with write protection enabled cannot be overwritten.

NOTE 2 This means no reply to the WRITE MEMORY LOCATION command when attempting to write to a write-protected location.

9.11.5 Memory bank reading

9.11.5.1 General

A selected memory bank location can be read using command "READ MEMORY LOCATION (*DTR1, DTR0*)". The answer shall be the value of the byte at the addressed memory bank location.

If the selected memory bank is not implemented, the command shall be discarded. ~~If the memory bank exists, and selected memory bank location is~~

~~• not implemented, or~~

~~• above the last accessible memory location,~~

~~the answer shall be NO.~~ **11**

If the selected memory bank location is below location 0xFF, *"DTR0"* shall be incremented by one, even if the memory location is not implemented. Otherwise, *"DTR0"* shall not change. This mechanism allows for easy consecutive reading of memory bank locations.

~~To ensure consistent data when reading a multi-byte value from a memory bank, it is recommended that a mechanism be implemented that latches all bytes of the multi-byte value when the first byte of the multi-byte value is read and that unlatches the bytes on receipt of any command other than "READ MEMORY LOCATION (*DTR1, DTR0*)".~~ **12**

After reading a number of bytes from a memory bank, the application controller should check the value of "*DTR0*" to verify it is at the expected~~/~~ or desired location. Any mismatch indicates an error while reading.

9.11.5.2 Reading multi-byte values **13**

To ensure consistent data when reading a multi-byte value from a memory bank, a mechanism shall be implemented that latches all bytes of the multi-byte value when the first byte of the multi-byte value is read, and holds them latched until the first byte of any multi-byte or single byte value in any memory bank of the same logical unit is read.

Reading from a multi-byte location shall reply with the stored value, and not values from the write-buffer that is used to buffer values before the complete value is written to memory (see 9.11.6.3).

9.11.5.3 Unimplemented locations **14**

If a memory bank exists, and the selected memory bank location is

- not implemented and MASK is not shown in the allowed range of values, or
- above the last accessible memory location,

the answer to "READ MEMORY LOCATION (*DTR1, DTR0*)" shall be NO.

If a memory bank exists, and the selected memory bank location is

- not implemented, and
- the allowed range for the value stored in the memory bank location includes MASK,

the answer to "READ MEMORY LOCATION (*DTR1, DTR0*)" shall be MASK.

NOTE The use of MASK in the range of allowed values shown in a memory bank table, effectively allows the implementation of the location to be optional.

9.11.5.4 Temporarily unavailable locations 15

For memory locations which include TMASK in the range of allowed values in a memory bank table, the answer to "READ MEMORY LOCATION (*DTR1, DTR0*)" shall be TMASK if the value is temporarily unavailable. Except in fault conditions, the control device shall provide a valid value within 30 s.

EXAMPLE After an external supply power cycle, memory bank values such as measurements can be temporarily unavailable due to the settling time of digital filters, and is indicated by the use of TMASK.

NOTE If TMASK is returned repeatedly over a period of time, application controllers can deduce that there is a fault in the control device.

9.11.5.5 Latching a complete memory bank for reading 16

Memory bank tables that show the lock byte value 0xAA latches the complete bank, shall operate as follows:

If the lock byte contains a value other than 0xAA, writing the following values to the lock byte shall cause the stated result:

- 0xAA: All locations in the memory bank shall be latched and shall not change until the lock byte is written, or a power cycle occurs.
- Other values: The memory bank latch shall not be affected.

If the lock byte contains the value 0xAA, writing the following values to the lock byte shall cause the stated result:

- 0xAA: All locations in the memory bank shall be re-latched (updated) and shall not change again until the lock byte is written, or a power cycle occurs.
- Other values: The memory bank latch shall be removed. Memory reads shall result in the latest values being returned.

An attempt to write to any location other than the lock byte of a latched memory bank shall result in the same behaviour as if the memory location is not implemented.

Latching of a full memory bank shall not affect reading or writing of other memory banks.

EXAMPLE The following example demonstrates two application controllers accessing a latched memory bank: If application controller A latches a full memory bank 202, then application controller B needs to be aware that memory bank 202 is latched. When application controller B reads from memory bank 202, application controller B will get the latched data. Application controller B can determine this by either monitoring the bus activity related to writing the lock byte, or by reading the lock byte before reading other locations.

9.11.6 Memory bank writing

9.11.6.1 General

Write commands are special commands and therefore not addressable. In order to select the correct control device(s) the addressable command "ENABLE WRITE MEMORY" shall be used. Upon execution of "ENABLE WRITE MEMORY", the addressed control device(s) shall set *"writeEnableState"* to ENABLED.

Only while *"writeEnableState"* is ENABLED, and the addressed memory bank is implemented, the control device shall execute the following commands to write to a selected memory bank location:

- "WRITE MEMORY LOCATION (*DTR1, DTR0, data*)": The control device shall confirm writing a memory location with an answer equal to the value *data*.

NOTE 1 The value that can be read from the memory bank location is not necessarily *data*.

- "WRITE MEMORY LOCATION – NO REPLY (*DTR1, DTR0, data*)": Writing a memory location shall not cause the control device to reply.
- "DIRECT WRITE MEMORY (*DTR1, offset, data*)": The address of the memory location inside the selected bank is given by the content of the instance byte. *offset* is copied to "*DTR0*", after which the command is treated as "WRITE MEMORY LOCATION (*DTR1, DTR0, data*)". The control device shall confirm writing a memory location by replying with an answer equal to *data*.

A control device shall set *"writeEnableState"* to DISABLED if any command other than one of the following commands is accepted:

- "WRITE MEMORY LOCATION (*DTR1, DTR0, data*)",
  "WRITE MEMORY LOCATION – NO REPLY (*DTR1, DTR0, data*)",
  "DIRECT WRITE MEMORY (*DTR1, offset, data*)";
- "DTR0 (*data*)", "DTR1 (*data*)", "DTR1:DTR0 (*data1, data0*)", DTR2 (*data*), DTR2:DTR1 (*data2, data1*);
- "QUERY CONTENT DTR0", "QUERY CONTENT DTR1", "QUERY CONTENT DTR2".

NOTE 2 This means that commands other than those shown above, which are accepted and then discarded, still set "*writeEnableState*" to DISABLED.

9.11.6.2 Write failures

If the selected memory bank location is

- not implemented, or
- above the last accessible memory location, or
- locked (see 9.11.2), or
- not writeable, or
- is a protectable location that is currently write protected (see 9.11.4), or
- there is an attempt to write a value outside of the permitted range,

the answer to "WRITE MEMORY LOCATION (*DTR1, DTR0, data*)" and "DIRECT WRITE MEMORY (*DTR1, offset, data*)" shall be NO and no memory location shall be written to. If this applies when writing to a multi-byte location, the reply of NO shall occur after attempting to write to the least significant byte (LSB).

If the selected memory bank location is below 0xFF, *"DTR0"* shall be incremented by one. Otherwise, *"DTR0"* shall not change. This mechanism allows for easy consecutive writing to memory bank locations.

9.11.6.3 Writing multi-byte values 17

To ensure consistent data when writing a multi-byte value into a memory bank, ~~it is recommended that a mechanism be implemented that only accepts the new multi-byte value for writing after all bytes of the multi-byte value have been accepted~~ a RAM buffer shall be used such that the buffer stores the temporary bytes being written until the LSB of the multi-byte value is written, at which point the complete value is written to the memory bank locations.

NOTE 1 The contents of buffers used to store temporary bytes for multi-byte writing can be lost when a write operation is started to any other multi-byte value in the same memory bank, or during a power-cycle, or as a result of execution of RESET MEMORY BANK (*DTR0*). This means that it is sufficient to include only one such RAM buffer for writing, per memory bank.

After writing a number of bytes to a memory bank, the application controller should check the value of "*DTR0*" to verify it is at the expected/ or desired location. Any mismatch indicates an error while writing.

NOTE 2 "*DTR0*" is also incremented if a non-implemented memory bank location is addressed before 0xFF is reached.

9.11.7 Memory bank 0

Memory bank 0 contains information about the control device. Memory bank 0 shall be implemented in all multi-master control devices.

Memory bank 0 shall be implemented using the memory map shown in Table 13, with at least the memory locations up to address 0x7F implemented, excluding reserved locations.

Table 13 – Memory map of memory bank 0

Address	Description	Default value (factory)	Memory type
0x00	Address of last accessible memory location	factory burn-in	ROM
0x01	Reserved – not implemented	answer NO	n.a.
0x02	Number of last accessible memory bank	factory burn-in, range [0,0xFF]	ROM
0x03	GTIN byte 0 (MSB)[a]	factory burn-in	ROM
0x04	GTIN byte 1	factory burn-in	ROM
0x05	GTIN byte 2	factory burn-in	ROM
0x06	GTIN byte 3	factory burn-in	ROM
0x07	GTIN byte 4	factory burn-in	ROM
0x08	GTIN byte 5 (LSB)	factory burn-in	ROM
0x09	Firmware version (major)	factory burn-in	ROM
0x0A	Firmware version (minor)	factory burn-in	ROM
0x0B	Identification number byte 0 (MSB)	factory burn-in	ROM
0x0C	Identification number byte 1	factory burn-in	ROM
0x0D	Identification number byte 2	factory burn-in	ROM
0x0E	Identification number byte 3	factory burn-in	ROM
0x0F	Identification number byte 4	factory burn-in	ROM
0x10	Identification number byte 5	factory burn-in	ROM
0x11	Identification number byte 6	factory burn-in	ROM
0x12	Identification number byte 7 (LSB)	factory burn-in	ROM
0x13	Hardware version (major)	factory burn-in	ROM
0x14	Hardware version (minor)	factory burn-in	ROM
0x15	101 version number[b]	factory burn-in, according to implemented version number	ROM
0x16	102 version number of all integrated control gear[c]	factory burn-in, according to implemented version number	ROM
0x17	103 version number of all integrated control devices[d]	factory burn-in, according to implemented version number	ROM
0x18	Number of logical control device units in the bus unit	factory burn-in, range [1,64]	ROM
0x19	Number of logical control gear units in the bus unit	factory burn-in, range [0,64]	ROM
0x1A	Index number of this logical control device unit	factory burn-in, range [0,(location 0x18) − 1]	ROM

0x1B[f]	Current bus unit configuration[f] **18**	factory burn-in[g]	ROM
[~~0x1B~~0x1C,0x7F]	Reserved – not implemented	answer NO	n.a.
[0x80,0xFE]	Additional control device information[e]	[e]	ROM
0xFF	Reserved – not implemented	answer NO	n.a.

Key

GTIN global trade item number

LSB least significant byte

MSB most significant byte

[a] It is recommended that the product GTIN is not re-used within the expected lifetime of the product after installation.

[b] Format of the version number is defined in IEC 62386-101:~~2014 and IEC 62386-101:2014/AMD1:2018~~2022, 4.2.

[c] Format of the version number is defined in IEC 62386-102:~~2014 and IEC 62386-102:2014/AMD1:2018~~2022, 4.2. If not implemented, this is indicated by 0xFF.

[d] Format of the version number is defined in 4.2.

[e] Purpose and (default) value of these bytes shall be defined by the manufacturer.

[f] See 9.20. If this location is not implemented, the answer shall be NO.

[g] The current bus unit configuration can be changed by a manufacturer-specific method.

If there is more than one logical unit built into one bus unit, all logical units shall have the same values in memory bank locations 0x03 up to and including 0x19. All control device logical units shall have the same values in location 0x1B.

A bus unit ~~might~~ can contain both control gear and control devices. They share various numbers (e.g. GTIN, identification number~~, etc~~). To avoid problems when reading, and getting different answers depending on the addressing scheme used, the memory bank layout is the same for control gear and for control devices up to and including location 0x19. The data shall be the same as well. The application controller can use either the commands specified in IEC 62386-102:— or the ~~103~~ commands specified in this document to identify the basic data, provided both are implemented.

The bytes in locations 0x03 to 0x08 ("GTIN 0" to "GTIN 5") shall contain the global trade item number (GTIN)~~, e.g. the EAN,~~ in binary (see [1][1]). The bytes shall be stored most significant first and filled with leading zeroes.

The bytes in locations 0x09 and 0x0A ("firmware version") shall contain the firmware version of the bus unit.

The bytes in locations 0x0B to 0x12 ("identification number byte 0" to "identification number byte 7") shall contain 64 bits of an identification number of the bus unit~~, prefereably~~. It is recommended that this identification number is the serial number. The identification number shall be stored with the least significant byte in "identification number byte ~~8~~ 7" and unused bits shall be filled with 0.

The combination of the identification number and the GTIN number shall be unique.

The byte in location 0x13 and 0x14 ("hardware version") shall contain the hardware version of the bus unit.

[1] Numbers in square brackets refer to the Bibliography.

The byte in location 0x15 shall contain the implemented IEC 62386-101 version number of the bus unit.

The byte in location 0x16 shall contain the implemented IEC 62386-102 version number of the bus unit. If no control gear is implemented, the version number shall be 0xFF.

The byte in location 0x17 shall contain the implemented IEC 62386-103 version number of the bus unit. ~~If no control device is implemented, the version number shall be 0xFF.~~

The byte in location 0x18 shall contain the number of logical control device units integrated into the bus unit. The number of logical units shall be in the range of 1 to 64.

The byte in location 0x19 shall contain the number of logical control gear units integrated into the bus unit. The number of logical units shall be in the range of 0 to 64.

The byte in location 0x1A shall represent the unique index number of the logical control device unit that implements that memory bank. The valid range of this index number is 0 to the total number of logical control device units in the bus unit minus one.

~~NOTE As an example there might be a product containing three logical devices with three different short addresses. Each of these control devices has the same GTIN and identification number, each reports as number of devices the value 3 and the index of the three control devices is reported as 0, 1 or 2 respectively. Reading location 0x1A using broadcast yields a backward frame according to IEC62386-101:2014 and IEC 62386-101:2014/AMD1:2018, 9.5.2 (overlapping backward frame).~~

EXAMPLE A product can contain three control device logical units with three different short addresses. Each of these logical units has the same GTIN and identification number, each reports the number of logical control device units with the value 3 and the index of the three control device logical units is reported as 0, 1 or 2 respectively. Reading location 0x1A using broadcast yields a backward frame according to IEC62386-101:2022, 9.6.2 (corrupted backward frame).

The byte in location 0x1B, if present, shall contain the current bus unit configuration, which shall indicate the currently selected implementation of application controllers and logical units. See 9.20.

9.11.8 Memory bank 1 (optional)

Memory bank 1 is reserved for use by an original equipment manufacturer (OEM), (e.g. a luminaire manufacturer) to store additional information, which has no impact on the functionality of the control device. ~~The control device manufacturer may implement~~ Implementation of memory bank 1 is optional.

If implemented, memory bank 1 shall at least implement the memory locations up to and including address 0x10. The fixed usage for location 0x00 to 0x02 and the recommended memory map usage for locations 0x03 to 0x10 is shown in Table 14.

Table 14 – Memory map of memory bank 1

Address	Description	Default value (factory)	Reset value[b]	Memory type
0x00	Address of last accessible memory location	factory burn-in, range [0x10,0xFE]	no change	ROM
0x01	Indicator byte[a]	[a]	[a]	any[a]
0x02	Memory bank 1 lock byte. Lockable bytes in the memory bank shall be read-only while the lock byte has a value different from 0x55.	0xFF	0xFF[c]	RAM-RW
0x03	OEM GTIN byte 0 (MSB)	0xFF	no change	NVM-RW (lockable)

Address	Description	Default value (factory)	Reset value[b]	Memory type
0x04	OEM GTIN byte 1	0xFF	no change	NVM-RW (lockable)
0x05	OEM GTIN byte 2	0xFF	no change	NVM-RW (lockable)
0x06	OEM GTIN byte 3	0xFF	no change	NVM-RW (lockable)
0x07	OEM GTIN byte 4	0xFF	no change	NVM-RW (lockable)
0x08	OEM GTIN byte 5 (LSB)	0xFF	no change	NVM-RW (lockable)
0x09	OEM identification number byte 0 (MSB)	0xFF	no change	NVM-RW (lockable)
0x0A	OEM identification number byte 1	0xFF	no change	NVM-RW (lockable)
0x0B	OEM identification number byte 2	0xFF	no change	NVM-RW (lockable)
0x0C	OEM identification number byte 3	0xFF	no change	NVM-RW (lockable)
0x0D	OEM identification number byte 4	0xFF	no change	NVM-RW (lockable)
0x0E	OEM identification number byte 5	0xFF	no change	NVM-RW (lockable)
0x0F	OEM identification number byte 6	0xFF	no change	NVM-RW (lockable)
0x10	OEM identification number byte 7 (LSB)	0xFF	no change	NVM-RW (lockable)
≥ 0x11	Additional control device information[a]	a	a	a
0xFF	Reserved – not implemented	answer NO	no change	n.a.

Key

GTIN global trade item number

LSB least significant byte

MSB most significant byte

OEM original equipment manufacturer

[a] The purpose, default/ value, power on/ value, reset value and memory ~~access~~ type of these bytes shall be defined by the manufacturer.

[b] Reset value after "RESET MEMORY BANK".

[c] Also used as power on value.

The bytes in locations 0x03 to 0x08 ("OEM GTIN 0" to "OEM GTIN 5") should be used to identify the product containing the control device. If the bytes are used for the GTIN, the bytes shall be stored most significant bit first and filled with leading zeroes. These bytes should be programmed by the OEM.

The bytes in locations 0x09 to 0x10 ("OEM identification number byte 0" to "OEM identification number byte 7") should contain 64 bits of an identification number of the OEM product. If the bytes are used for the identification number, it shall be stored with the least significant byte in "identification number byte 7" and unused bits shall be filled with 0. These bytes should be programmed by the OEM.

The combination of OEM GTIN and OEM identification number should be unique.

9.11.9 Manufacturer-specific memory banks

The manufacturer may use additional memory banks in the range of 2 to 199 to store additional information. The memory map of additional banks shall comply with Table 12.

9.11.10 Reserved memory banks

Memory banks 200 to 255 are reserved for future use ~~and shall not be implemented~~, or are described in the IEC 62386-3xx series. Implementation or use other than described in the IEC 62386-3xx series is not permitted.

9.12 Reset

9.12.1 Reset operation

A control device shall implement a reset operation to set all device variables and instance variables (see Table 19 and Table 20) to their reset values.

NOTE For some variables this operation could have no effect at all.

The reset operation shall take at most 300 ms to complete. While the reset operation is in progress, the control device ~~may or may not~~ can respond to ~~any~~ commands or not. However, until the reset operation is complete, none of the affected variables needs to have a defined value.

An application controller can trigger the reset operation using the "RESET" instruction and should wait at least 350 ms to ensure all control devices have finished the reset operation.

9.12.2 Reset memory bank operation

A control device shall implement a reset operation to set the content of all unlocked memory banks to their reset values (see 9.11), followed by locking the memory banks.

NOTE For some memory bank locations this operation ~~may~~ could have no effect at all.

The reset memory bank operation shall take at most 10 s to complete. While this reset operation is in progress, the control device ~~may or may not~~ can respond to ~~any~~ commands or not. However, until this reset memory bank operation is complete, none of the affected memory locations have a defined value.

An application controller can trigger the reset operation for a specific memory bank, or for all implemented memory banks, using the "RESET MEMORY BANK (*DTR0*)" instruction and it should then wait for at least 10,1 s to ensure all control devices have ~~finished~~ enough time to finish the reset memory bank operation.

9.13 Power on behaviour

9.13.1 Power on

After an external power cycle (see IEC 62386-101:~~2014 and IEC 62386-101:2014/AMD1:2018~~2022, 4.11.1 and 4.11.5), the device shall maintain its most recent configuration of NVM variables saved ~~as a result of the execution of "SAVE PERSISTENT VARIABLES",~~ according to 9.18 **19**, with the following exceptions:

- all variables mentioned in Table 19 and Table 20 shall be set to the value indicated in the power on value column. The variables that are marked with 'no change' in the power on value column shall not be considered. The variables defined in implemented parts of the IEC 62386-~~2xx~~3xx series shall be included;
- the memory bank write enable state shall be disabled for all memory banks and the lock byte shall be set to 0xFF;

- quiescent mode shall be cancelled (see 9.10.4);
- all running timers shall be stopped and cancelled (reset);
- *"powerCycleSeen"* shall be set to TRUE.

"powerCycleSeen" can be observed through "QUERY DEVICE STATUS".

In order to observe a subsequent power cycle, the application controller should clear *"powerCycleSeen",* using the command "RESET POWER CYCLE SEEN.

~~In a system with multiple application controllers all application controllers may need power cycle information of other control devices in the system. Clearing *"powerCycleSeen"* should be done with some consideration.~~

NOTE In a system with multiple application controllers that make use of power cycle information of other control devices in the system, application controllers can take consideration when clearing *"powerCycleSeen"*.

9.13.2 Power cycle notification

After completing its ~~external~~ power ~~cycle,~~ on behaviour according to 9.13.1, a bus unit shall generate one power cycle event message per device if the *"powerCycleNotification"* is ENABLED for at least one of its logical units.

An application controller can use "ENABLE POWER CYCLE NOTIFICATION" and "DISABLE POWER CYCLE NOTIFICATION" to enable/disable power cycle events for specific control devices. *"powerCycleNotification"* shall be DISABLED by default.

NOTE 1 The power cycle notification is not inhibited by *"applicationActive"* nor by any *"instanceActive"*.

The event shall be generated using the "POWER NOTIFICATION (*device*)" message as described in 11.2. The event message shall be sent once using priority 2 and with a uniformly distributed delay between ~~1s, 3s and 5 s~~ 1,3 s and 5,0 s after completion of the power-on procedure.

NOTE 2 Applying a random delay helps to avoid collisions of power cycle notifications.

9.14 Priority use

9.14.1 General

The purpose of forward frame priorities is to facilitate appropriate system behaviour within a multi-master system. Priorities ensure that transmissions for time critical system reaction will have precedence over transmissions for non-time critical system operation.

- ~~Priority 1 shall be used for all forward frames within a transaction (see IEC62386-101:2014 and IEC 62386-101:2014/AMD1:2018, 9.3), except for the first forward frame. Priority 1 shall neither be used for forward frames that are not part of a transaction, nor for those that start a transaction.~~
- The first forward frame in a transaction (see IEC62386-101:2022, 9.3) shall be sent with priority 2 to 5. All other forward frames in a transaction shall be sent with priority 1. **20**
- Forward frames that are not part of a transaction shall be sent with priority 2 to 5.
- Priority 2 should be used to execute user instigated actions for switching or dimming the lights. This implies appropriate event messages and ~~arc power commands~~ level instructions. Priority 2 ~~might~~ can also be used during commissioning (e.g. addressing).

~~NOTE~~ EXAMPLE 1 Switching or dimming actions triggered via push-button or presence detector.

- Priority 3 should be used for configuration of a bus unit and for those event messages that are not covered by Priorities 2 and 4.

~~NOTE~~ EXAMPLE 2 Writing to memory banks or feedback events.

- Priority 4 should be used to execute automatic actions for switching or dimming the lights. This means sending appropriate event messages and ~~arc power commands~~ level instructions.

~~NOTE~~ EXAMPLE 3 Switching or dimming actions triggered by a light sensor.

- Priority 5 should be used for periodic query commands.

9.14.2 Priority of input notifications

An instance shall use a default *"eventPriority"* equal to Priority 4 when transmitting an event message to produce an "INPUT NOTIFICATION (*device/instance, event*)". For particular instance types, this default priority is subject to change by the relevant parts of the IEC 62386-3xx series.

In a system, the default *"eventPriority"* can be overruled by the application controller using "SET EVENT PRIORITY (*DTR0*)". "QUERY EVENT PRIORITY" can be used to observe the currently active *"eventPriority"*.

9.15 Assigning short addresses

9.15.1 General

"shortAddress" shall be derived from *data* or *"DTR0"* depending on the command used. It shall be set on ~~receipt~~ execution of "PROGRAM SHORT ADDRESS (*data*)" or "SET SHORT ADDRESS (*DTR0*)" as follows:

- if *data* or "*DTR0*" = MASK: MASK (effectively deleting the short address);
- if *data* or *"DTR0"* < 0x40: *data* or *"DTR0"*;
- in all other cases: no change.

9.15.2 Random address allocation

A control device shall implement an initialisation state, only in which, apart from the other operations identified in this document, a set of commands are enabled that allow an application controller to detect and uniquely identify control devices available on the bus and assign short addresses to these devices.

The initialisation state is a temporary state which is entered with the command "INITIALISE (*device*)". It ~~ends~~ shall end automatically 15 min ± 1,5 min after the last "INITIALISE (*device*)" command was executed. Additionally, a power cycle or the command "TERMINATE" shall cause the control device to leave the initialisation state immediately.

The control device shall have three possible values for *"initialisationState"*:

- DISABLED, not in initialisation state;
- ENABLED, in initialisation state;
- WITHDRAWN, in initialisation state, yet identified and withdrawn.

The following (special) commands are initialisation commands:

- "RANDOMISE", "COMPARE" and "WITHDRAW";
- "SEARCHADDRH (*data*)", "SEARCHADDRM (*data*)" and "SEARCHADDRL (*data*)";
- "PROGRAM SHORT ADDRESS (*data*)", "VERIFY SHORT ADDRESS (*data*)" and "QUERY SHORT ADDRESS";
- "IDENTIFY DEVICE".

NOTE "IDENTIFY DEVICE" is by itself not an initialisation command, but typically used during initialisation.

9.15.3 Identification of a device

No variables shall be affected by the identification procedure. Normal processing shall continue, i.e. events can be generated on a change of the input value; this shall not stop identification. Where appropriate, variables can be temporarily ignored, so that after the identification has ended, there are no side effects.

Identification shall be stopped upon execution of any instruction other than INITIALISE (*device*) or "IDENTIFY DEVICE".

Identification can be started by sending the instruction "IDENTIFY DEVICE". This shall start or restart a 10 s ± 1 s timer. While the timer is running, a procedure enabling an observer to identify the selected control device shall run. If the timer expires, identification shall stop.

NOTE The actual procedure is manufacturer specific.

When identification is stopped by an application controller, the corresponding timer shall be cancelled immediately.

9.16 Exception handling

Control devices and instances shall expose whether an error has occurred by setting (in case of error) and resetting (in case of no error) the following flags.

- An application controller shall change *"applicationControllerError"*. This status can be queried through "QUERY DEVICE STATUS" (see 9.17.2). Detailed error information ~~may~~ can be ~~available~~ obtained from "QUERY APPLICATION CONTROLLER ERROR" (see 11.6.4).
- A control device that is not an application controller shall have *"applicationControllerError"* set to FALSE.
- An input device shall change *"inputDeviceError"* to indicate an error in any of the input device instances of the logical unit. This status can be queried through "QUERY DEVICE STATUS" (see 9.17.2). Detailed error information ~~may~~ can be ~~available~~ obtained from "QUERY INPUT DEVICE ERROR" (see 11.6.5).
- A control device that is not an input device shall have *"inputDeviceError"* set to FALSE.
- An instance shall change *"instanceError"*. This status can be queried through "QUERY INSTANCE STATUS" (see 9.17.3). Detailed error information ~~may~~ can be ~~available~~ obtained from "QUERY INSTANCE ERROR" (see 11.9.4).

9.17 Device capabilities and status information

9.17.1 Device capabilities

Each control device shall expose its features as a combination of device capabilities as given in Table 15:

Table 15 – Control device capabilities

Bit	Description	Value	See Subclause
0	*"applicationControllerPresent"* is TRUE?	"1" = "Yes"	9.3.3
1	*"numberOfInstances"* is greater than 0?	"1" = "Yes"	9.5.2
2	*"applicationControllerAlwaysActive"* is TRUE?	"1" = "Yes"	9.10.2
3 to 4	Reserved for IEC 62386-104 (see [2])	"0" = default value	
5	At least one instance supports configuration of *"instanceType"* or *"instanceConfiguration[]"* **21**	"1" = "Yes"	9.19
~~3~~ 6 to 7	unused	"0" = default value	

The device capabilities can be queried using "QUERY DEVICE CAPABILITIES".

9.17.2 Device status

Each control device shall expose its status as a combination of device properties as given in Table 16:

Table 16 – Control device status

Bit	Description	Value	See Subclause
0	*"inputDeviceError"* is TRUE?	"1" = "Yes"	9.16
1	*"quiescentMode"* is ENABLED?	"1" = "Yes"	9.10.4
2	*"shortAddress"* is MASK?	"1" = "Yes"	9.15.1
3	*"applicationActive"* is TRUE?	"1" = "Yes"	9.10.1
4	*"applicationControllerError"* is TRUE?	"1" = "Yes"	9.16
5	*"powerCycleSeen"* is TRUE?	"1" = "Yes"	9.13
6	*"resetState"* is TRUE?	"1" = "Yes"	~~9.16.2.1~~ See below
7	~~unused~~ Reserved for IEC 62386-104 (see [2])	"0" = default value	

The device status can be queried using "QUERY DEVICE STATUS".

Bit 6: ~~Reset state~~

"resetState" shall be set to TRUE if all the NVM variables mentioned in Table 19 and Table 20 are at their reset value. The NVM variables that are marked with 'no change' in the reset value column shall not be considered. NVM variables defined in implemented parts of the IEC 62386-3xx series shall be included. In all other cases the bit shall be set to FALSE.

9.17.3 Instance status

Each instance shall expose its status as a combination of instance properties as given in Table 17:

Table 17 – Instance status

Bit	Description	Value	See subclause
0	*"instanceError"* is TRUE?	"1" = "Yes"	9.16
1	*"instanceActive"* is TRUE?	"1" = "Yes"	9.10.3
2 to 7	unused	"0" = default value	

The instance status can be observed using "QUERY INSTANCE STATUS".

9.18 Non-volatile memory 22

~~Physical non-volatile memory typically supports a limited number of write cycles. Since many variables are NVM type, the physical limitations need some attention.~~

~~A control device should store NVM variables in such a way that their content is never lost and the intended lifetime of the device can be reached. This means that it may not be possible to physically write every change in a variable immediately. There may be situations in which the control device is not able to physically write the variables to NVM, especially if a particular NVM variable is changed very frequently.~~

~~Since the application controller cannot know the control device's internal mechanism for physically saving persistent variables, the instruction "SAVE PERSISTENT VARIABLES" is defined to force the control device to physically write all variables of type NVM to memory. This command is an addition to the normal writing of NVM variables. Its intended use is to ensure that important changes made by an application controller cannot be lost, e.g. after assigning all short addresses or setting other important (and stable) configuration data. Clearly it is not intended to be used after every value change. Typically, this command is used only a handful of times for an entire installation.~~

~~NOTE Typically the command can be used a few thousand times before causing physical damage to the control device's NVM.~~

~~Physically saving the variables in response to the instruction shall take at most 300 ms to complete. While the saving operation is on-going, the control device may or may not respond to any command. However, until the operation is complete, the value of the affected variables may be undefined.~~

~~An application controller can trigger the save operation using the "SAVE PERSISTENT VARIABLES" instruction and should wait at least 350 ms to ensure all devices have finished the operation.~~

After any change to an NVM variable, the new value shall be restored after a power cycle, provided a period of at least 30 s followed the NVM variable change, before the power cycle.

In case the above condition is not met, it is possible that the changed NVM variable will not be restored with the new value.

NOTE The method used to ensure NVM variables are saved is manufacturer specific. One possible method is to save some NVM variables within 30 s of them being changed, with others being saved at the time the external power failure occurs. The method chosen is likely to influence the lifetime of the non-volatile memory.

9.19 Instance types and configuration 23

Instances may optionally support a change of *"instanceType"* with the instruction "SET INSTANCE TYPE (*DTR0*)".

The current instance type can be determined by using "QUERY INSTANCE TYPE", and all available instance types can be discovered by use of "QUERY AVAILABLE INSTANCE TYPES".

Instances may optionally support a change of *"instanceConfiguration[]"* with the instruction "SET INSTANCE CONFIGURATION (*DTR0, DTR2:DTR1*)".

The current instance configuration can be discovered by using "QUERY INSTANCE CONFIGURATION (*DTR0*)".

Any standardised values of *"instanceConfiguration[]"* are published in the appropriate parts of the IEC 62386-3xx series. A manufacturer-specific range of *"instanceConfiguration[]"* may be used, but use of such values shall not remove any requirements in this document.

If instance configuration is supported, the manual shall state which locations within *"instanceConfiguration[]"* are supported, shall describe the use of such locations in the case that these are in the manufacturer-specific range, and shall state their memory type (readable and/or writeable).

EXAMPLE 1 Changing the instance type: An instance with volt-free inputs can support a change of instance type between type 1 (push-button) and type 2 (absolute/switch-input device).

EXAMPLE 2 Changing the instance configuration: An occupancy sensor instance with multiple sensor technologies such as PIR, microwave/doppler and ultra-sonic can allow configuration of which sensor technologies are currently active.

9.20 Current bus unit configuration 24

If present, the value of current bus unit configuration from memory bank 0 shall indicate the currently active implementation of:

- the value of "*applicationControllerPresent*".

For the given value of current bus unit configuration, the implementation shall be as shown in Table 18.

Table 18 – Current bus unit configuration

Current bus unit configuration	"*applicationControllerPresent*"	Description
0 to 191[a]	Reserved	
192 to 255	Manufacturer-specific	
[a] Values reserved for future updates to this document.		

A manufacturer-specific method is permitted for selection of the current configuration. After changing, the bus unit can require a certain time, power-cycle or other requirement before the new configuration becomes active.

After changing the configuration, the value of "Current bus unit configuration" in memory bank 0 shall indicate the active configuration, and the bus unit operation shall be updated according to the new configuration.

10 Declaration of variables

Table 19 shows the default values, the reset values, the range of validity and the type of memory of the defined instance independent variables.

Table 20 shows the default values, the reset values, the range of validity and the type of memory of the defined variables of each of the instances.

The variables that are declared in this Clause 10 shall not be made available for writing through a memory bank.

Table 19 – Declaration of device variables

Variable	Default value (factory)	Reset value	Power on value	Range of validity	Memory type
"shortAddress"	MASK (no address)	no change	no change	[0,63], MASK	NVM
"deviceGroups"	0x0000 0000	0x0000 0000	no change	[0,0xFFFF FFFF]	NVM
"searchAddress"	[a]	0xFF FF FF	0xFF FF FF	[0,0xFF FF FF]	RAM
"randomAddress"	0xFF FF FF	0xFF FF FF	no change	[0,0xFF FF FF]	NVM
"DTR0"	[a]	no change	0x00	[0,0xFF]	RAM
"DTR1"	[a]	no change	0x00	[0,0xFF]	RAM
"DTR2"	[a]	no change	0x00	[0,0xFF]	RAM
"numberOfInstances"	factory burn-in	no change	no change	[0,32]	ROM
"operatingMode"	factory burn-in	no change	no change	0, [0x80,0xFF]	NVM
"quiescentMode"	[a]	DISABLED	DISABLED	[ENABLED, DISABLED]	RAM
"applicationActive"	*applicationControllerPresent*	no change	no change	[TRUE, FALSE]	NVM
"writeEnableState"	[a]	DISABLED	DISABLED	[ENABLED, DISABLED]	RAM
"applicationControllerPresent"	factory burn-in	no change	no change	[TRUE, FALSE]	ROM
"applicationControllerAlwaysActive"	factory burn-in	no change	no change	[TRUE, FALSE]	ROM
"powerCycleSeen"	[a]	FALSE	TRUE	[TRUE, FALSE]	RAM
"powerCycleNotification"	DISABLED	no change	no change	[ENABLED, DISABLED]	NVM
"initialisationState"	[a]	no change	DISABLED	[ENABLED, DISABLED, WITHDRAWN]	RAM
"applicationControllerError"	[a]	[c]	FALSE[b]	[TRUE, FALSE]	RAM
"inputDeviceError"	[a]	[c]	FALSE[b]	[TRUE, FALSE]	RAM
"resetState"	TRUE	TRUE	TRUE[b]	[TRUE, FALSE]	RAM
"eventPriority"[d]	4	no change	no change	[2,5]	NVM
"versionNumber"	~~2.1~~ 3.0	no change	no change	~~00001001b~~ 00001100b	ROM

[a] Not applicable.

[b] The value should reflect the actual situation as soon as possible.

[c] The value could change as a consequence of the RESET command execution.

[d] This variable is independent of the instance variable(s) "*eventPriority*". 25

Table 20 – Declaration of instance variables

Variable	Default value (factory)	Reset value	Power on value	Range of validity	Memory type
"instanceGroup0"	MASK	MASK	no change	[0,31], MASK	NVM
"instanceGroup1"	MASK	MASK	no change	[0,31], MASK	NVM
"instanceGroup2"	MASK	MASK	no change	[0,31], MASK	NVM
"instanceActive"	TRUE	no change	no change	[TRUE, FALSE]	NVM
"instanceType"	factory burn-in	no change	no change	[0,31]	ROM or NVM
"resolution"	factory burn-in	no change	no change	[1,255]	ROM
"inputValue"	[a]	no change	no change[b]	$[0,2^{N*8} - 1]$[c]	RAM
"instanceNumber"	factory burn-in	no change	no change	[0, *"numberOfInstances"* – 1]	ROM
"eventFilter"[d]	0xFF FF FF	0xFF FF FF	no change	[0, 0xFF FF FF]	NVM
"eventScheme"	0	0	no change	[0,4]	NVM
"eventPriority"[d f]	4	no change	no change	[2,5]	NVM
"instanceError"	[a]	[e]	FALSE[b]	[TRUE, FALSE]	RAM
"instanceConfiguration[]"[g] **26**	factory burn-in	no change	no change	[0, 0xFFFF][g]	NVM or ROM

[a] Not applicable.

[b] The value should reflect the actual situation as soon as possible.

[c] *N* computed as (*"resolution"*/8) rounded up to the nearest integer.

[d] For particular instance types, the values belonging to this variable can be changed by the relevant part of the IEC 62386-3xx series.

[e] The value could change as a consequence of the RESET command execution.

[f] Instance variable(s) "*eventPriority*" are independent of the device variable "*eventPriority*". **27**

[g] The size of this array is manufacturer-specific, from 0 to 256 locations. Implemented locations are up to 16 bits in size.

11 Definition of commands

11.1 General

Unused opcodes are reserved for future needs.

11.2 Overview sheets

Table 21 to Table 24 give an overview of the control device's event messages, standard commands and special commands.

Table 21 – Instance event messages

Event message name	Event source information	Event information	References	Command subclause
INPUT NOTIFICATION (*device/instance, event*)	*device/instance*	*event*	9.7.3 and 9.8.4	11.3.1

Table 22 – Device event messages

Event message name	Bits [23,13]	Bits [12,0]	References	Command subclause
POWER NOTIFICATION (*device*)	0x7F7	*device*	9.7.2 and 9.13.2	11.3.2

Table 23 – Standard commands

Command name	Address byte	Instance byte			Opcode byte	App Ctrl	Input device	DTR0	DTR1	DTR2	Answer	Send twice	References	Command subclause
		Device	Instance	Feature										
IDENTIFY DEVICE	*Device*	✓			0x00	✓	✓					✓	9.15.3	11.4.2
RESET POWER CYCLE SEEN	*Device*	✓			0x01	✓	✓					✓	9.13.1	11.4.3
RESET	*Device*	✓			0x10	✓	✓					✓	9.12.1	11.5.2
RESET MEMORY BANK (*DTR0*)	*Device*	✓			0x11	✓	✓	✓				✓	9.12.2	11.5.3
SET SHORT ADDRESS (*DTR0*)	*Device*	✓			0x14	✓	✓	✓				✓	9.15.1	11.5.4
ENABLE WRITE MEMORY	*Device*	✓			0x15	✓	✓					✓	9.11.6	11.5.5
ENABLE APPLICATION CONTROLLER	*Device*	✓			0x16	✓						✓	9.10.1	11.5.6
DISABLE APPLICATION CONTROLLER	*Device*	✓			0x17	✓						✓	9.10.1	11.5.7
SET OPERATING MODE (*DTR0*)	*Device*	✓			0x18	✓	✓	✓				✓	9.10.5	11.5.8
ADD TO DEVICE GROUPS 0-15 (*DTR2:DTR1*)	*Device*	✓			0x19	✓	✓		✓	✓		✓		11.5.9
ADD TO DEVICE GROUPS 16-31 (*DTR2:DTR1*)	*Device*	✓			0x1A	✓	✓		✓	✓		✓		11.5.10
REMOVE FROM DEVICE GROUPS 0-15 (*DTR2:DTR1*)	*Device*	✓			0x1B	✓	✓		✓	✓		✓		11.5.11
REMOVE FROM DEVICE GROUPS 16-31 (*DTR2:DTR1*)	*Device*	✓			0x1C	✓	✓		✓	✓		✓		11.5.12
START QUIESCENT MODE	*Device*	✓			0x1D	✓	✓					✓	9.10.4	11.5.13
STOP QUIESCENT MODE	*Device*	✓			0x1E	✓	✓					✓	9.10.4	11.5.14
ENABLE POWER CYCLE NOTIFICATION	*Device*	✓			0x1F	✓	✓					✓	9.13.2	11.5.15
DISABLE POWER CYCLE NOTIFICATION	*Device*	✓			0x20	✓	✓					✓	9.13.2	11.5.16
~~SAVE PERSISTENT VARIABLES~~	~~*Device*~~	~~✓~~			~~0x21~~	~~✓~~	~~✓~~					~~✓~~	~~9.17~~	~~11.5.17~~
Reserved[a]	*Device*	✓			0x21[a]	✓	✓					✓		

Command name	Address byte	Instance byte			Opcode byte	App Ctrl	Input device	DTR0	DTR1	DTR2	Answer	Send twice	References	Command subclause
		Device	Instance	Feature										
Reserved for IEC 62386-104 (see [2])	*Device*	✓			0x22	✓		✓						
Reserved for IEC 62386-104 (see [2])	*Device*	✓			0x23	✓								
Reserved for IEC 62386-104 (see [2])	*Device*	✓			0x24	✓								
QUERY DEVICE STATUS	*Device*	✓			0x30	✓	✓				✓		9.17.2	11.6.3
QUERY APPLICATION CONTROLLER ERROR	*Device*	✓			0x31	✓					✓		9.16	11.6.4
QUERY INPUT DEVICE ERROR	*Device*	✓			0x32		✓				✓		9.16	11.6.5
QUERY MISSING SHORT ADDRESS	*Device*	✓			0x33	✓	✓				✓			11.6.6
QUERY VERSION NUMBER	*Device*	✓			0x34	✓	✓				✓		4.2	11.6.7
QUERY NUMBER OF INSTANCES	*Device*	✓			0x35	✓	✓				✓		9.5	11.6.9
QUERY CONTENT DTR0	*Device*	✓			0x36	✓	✓	✓			✓			11.6.8
QUERY CONTENT DTR1	*Device*	✓			0x37	✓	✓		✓		✓			11.6.10
QUERY CONTENT DTR2	*Device*	✓			0x38	✓	✓			✓	✓			11.6.11
QUERY RANDOM ADDRESS (H)	*Device*	✓			0x39	✓	✓				✓			11.6.12
QUERY RANDOM ADDRESS (M)	*Device*	✓			0x3A	✓	✓				✓			11.6.13
QUERY RANDOM ADDRESS (L)	*Device*	✓			0x3B	✓	✓				✓			11.6.14
READ MEMORY LOCATION (*DTR1, DTR0*)	*Device*	✓			0x3C	✓	✓	✓	✓		✓		9.11.5	11.6.15
QUERY APPLICATION CONTROLLER ENABLED	*Device*	✓			0x3D	✓					✓		9.10.1	11.6.16
QUERY OPERATING MODE	*Device*	✓			0x3E	✓	✓				✓		9.10.5	11.6.17
QUERY MANUFACTURER SPECIFIC MODE	*Device*	✓			0x3F	✓	✓				✓		9.10.5	11.6.18
QUERY QUIESCENT MODE ~~Refer to 9.10.5 for further information~~	*Device*	✓			0x40	✓	✓				✓		9.10.4	11.6.19

Command name	Address byte	Instance byte			Opcode byte	App Ctrl	Input device	DTR0	DTR1	DTR2	Answer	Send twice	References	Command subclause
		Device	Instance	Feature										
QUERY DEVICE GROUPS 0-7	*Device*	✓			0x41	✓	✓				✓			11.6.20
QUERY DEVICE GROUPS 8-15	*Device*	✓			0x42	✓	✓				✓			11.6.21
QUERY DEVICE GROUPS 16-23	*Device*	✓			0x43	✓	✓				✓			11.6.22
QUERY DEVICE GROUPS 24-31	*Device*	✓			0x44	✓	✓				✓			11.6.23
QUERY POWER CYCLE NOTIFICATION	*Device*	✓			0x45	✓	✓				✓		9.13.2	11.6.24
QUERY DEVICE CAPABILITIES	*Device*	✓			0x46	✓	✓				✓		9.17.1	11.6.2
QUERY EXTENDED VERSION NUMBER(*DTR0*)	*Device*	✓			0x47	✓	✓	✓			✓			11.6.25
QUERY RESET STATE	*Device*	✓			0x48	✓	✓				✓		9.17.2	11.6.26
QUERY APPLICATION CONTROLLER ALWAYS ACTIVE	*Device*	✓			0x49	✓					✓		9.10.2	11.6.27
SET EVENT PRIORITY (*DTR0*)	*Device*	✓	✓		0x61		✓	✓				✓	9.14.2	11.8.8, 11.5.17
ENABLE INSTANCE	*Device*		✓		0x62		✓					✓	9.10.3	11.8.2
DISABLE INSTANCE	*Device*		✓		0x63		✓					✓	9.10.3	11.8.3
SET PRIMARY INSTANCE GROUP (*DTR0*)	*Device*		✓		0x64		✓	✓				✓	9.5.5	11.8.4
SET INSTANCE GROUP 1 (*DTR0*)	*Device*		✓		0x65		✓	✓				✓	9.5.5	11.8.5
SET INSTANCE GROUP 2 (*DTR0*)	*Device*		✓		0x66		✓	✓				✓	9.5.5	11.8.6
SET EVENT SCHEME (*DTR0*)	*Device*		✓		0x67		✓	✓				✓	9.7.3	11.8.7
SET EVENT FILTER (*DTR2, DTR1, DTR0*)	*Device*		✓		0x68		✓	✓	✓	✓		✓	9.7.4	11.8.9
SET INSTANCE TYPE (*DTR0*)	*Device*		✓		0x69		✓	✓				✓	9.19	11.8.10
SET INSTANCE CONFIGURATION (*DTR0, DTR2-DTR1*)	*Device*		✓		0x6A		✓	✓	✓	✓		✓	9.19	11.8.11
QUERY INSTANCE TYPE	*Device*		✓		0x80		✓				✓		9.5.3	11.9.2

Command name	Address byte	Instance byte			Opcode byte	App Ctrl	Input device	DTR0	DTR1	DTR2	Answer	Send twice	References	Command subclause
		Device	Instance	Feature										
QUERY RESOLUTION	*Device*		✓		0x81		✓				✓		9.8.2	11.9.3
QUERY INSTANCE ERROR	*Device*		✓		0x82		✓				✓		9.16	11.9.4
QUERY INSTANCE STATUS	*Device*		✓		0x83		✓				✓		9.17.3	11.9.5
QUERY EVENT PRIORITY	*Device*	✓	✓		0x84		✓				✓		9.14.2	11.9.13, 11.6.30
QUERY INSTANCE ENABLED	*Device*		✓		0x86		✓				✓		9.10.3	11.9.6
QUERY PRIMARY INSTANCE GROUP	*Device*		✓		0x88		✓				✓		9.5.5	11.9.7
QUERY INSTANCE GROUP 1	*Device*		✓		0x89		✓				✓		9.5.5	11.9.8
QUERY INSTANCE GROUP 2	*Device*		✓		0x8A		✓				✓		9.5.5	11.9.9
QUERY EVENT SCHEME	*Device*		✓		0x8B		✓				✓		9.7.3	11.9.10
QUERY INPUT VALUE	*Device*		✓		0x8C		✓				✓		9.8.3	11.9.11
QUERY INPUT VALUE LATCH	*Device*		✓		0x8D		✓				✓		9.8.3	11.9.12
QUERY FEATURE TYPE	*Device*	✓	✓		0x8E	✓	✓				✓		9.2, 9.5.4	11.9.14, 11.6.28
QUERY NEXT FEATURE TYPE	*Device*	✓	✓		0x8F	✓	✓				✓		9.2, 9.5.4	11.9.15, 11.6.29
QUERY EVENT FILTER 0-7	*Device*		✓		0x90		✓				✓		9.7.4	11.9.16
QUERY EVENT FILTER 8-15 ~~Refer to 9.7.4 for further information~~	*Device*		✓		0x91		✓				✓		9.7.4	11.9.17
QUERY EVENT FILTER 16-23 ~~Refer to 9.7.4 for further information~~	*Device*		✓		0x92		✓				✓		9.7.4	11.9.18
QUERY INSTANCE CONFIGURATION (*DTR0*)	*Device*		✓		0x93		✓	✓	✓	✓	✓		9.19	11.9.19

Command name	Address byte	Instance byte			Opcode byte	App Ctrl	Input device	DTR0	DTR1	DTR2	Answer	Send twice	References	Command subclause
		Device	Instance	Feature										
QUERY AVAILABLE INSTANCE TYPES	*Device*		✓		0x94		✓	✓	✓	✓	✓		9.19	11.9.20
[a] Reserved to maintain backward compatibility due to use in Edition 1 of IEC 62386-103:2014 (see [3]).														

Table 24 – Special commands (implemented by both application controller and input device)

Command name	Address byte	Instance byte	Opcode byte	DTR0	DTR1	DTR2	Answer	Send twice	References	Command subclause
TERMINATE	0xC1	0x00	0x00							11.10.2
INITIALISE (*device*)	0xC1	0x01	*device*					✓	9.15	11.10.3
RANDOMISE	0xC1	0x02	0x00					✓	9.15	11.10.4
COMPARE	0xC1	0x03	0x00				✓		9.15	11.10.5
WITHDRAW	0xC1	0x04	0x00						9.15	11.10.6
SEARCHADDRH (*data*)	0xC1	0x05	*data*						9.15	11.10.7
SEARCHADDRM (*data*)	0xC1	0x06	*data*						9.15	11.10.8
SEARCHADDRL (*data*)	0xC1	0x07	*data*						9.15	11.10.9
PROGRAM SHORT ADDRESS (*data*)	0xC1	0x08	*data*						9.15	11.10.10
VERIFY SHORT ADDRESS (*data*)	0xC1	0x09	*data*				✓		9.15	11.10.11
QUERY SHORT ADDRESS	0xC1	0x0A	0x00				✓		9.15	11.10.12
Reserved for IEC 62386-104 (see [2])	0xC1	0x0B	*data*	✓			✓			
Reserved for IEC 62386-104 (see [2])	0xC1	0x0C	*data*							
Reserved for IEC 62386-104 (see [2])	0xC1	0x0D	*data*							
WRITE MEMORY LOCATION (*DTR1, DTR0, data*)	0xC1	0x20	*data*	✓	✓		✓		9.11.6	11.10.13
WRITE MEMORY LOCATION – NO REPLY (*DTR1, DTR0, data*)	0xC1	0x21	*data*	✓	✓				9.11.6	11.10.14
DTR0 (*data*)	0xC1	0x30	*data*	✓						11.10.15

Command name	Address byte	Instance byte	Opcode byte	DTR0	DTR1	DTR2	Answer	Send twice	References	Command subclause
DTR1 (*data*)	0xC1	0x31	*data*		✓					11.10.16
DTR2 (*data*)	0xC1	0x32	*data*			✓				11.10.17
SEND TESTFRAME (*data)*	0xC1	*0x33*	*data*	✓	✓	✓				11.10.21
DIRECT WRITE MEMORY (*DTR1, offset, data*)	0xC5	*offset*	*data*	✓	✓		✓		9.11.6	11.10.18
DTR1:DTR0 (*data1, data0*)	0xC7	*data1*	*data0*	✓	✓					11.10.19
DTR2:DTR1 (*data2, data1*)	0xC9	*data2*	*data1*		✓	✓				11.10.20

11.3 Event messages

11.3.1 INPUT NOTIFICATION (*device/instance, event*)

The event message notifies of a change or a series of changes of *"inputValue"* at the instance of an input device as required by this document or by the relevant part of the IEC 62386-3xx series corresponding to the *"instanceType"* of the instance.

The transmitting instance shall

- generate the event message only while *"instanceActive"* is TRUE;
- generate the event message only while it is not in an error condition that prevents operation (see 9.16);
- use the currently active *"eventScheme"*;
- use the requested *"eventPriority"*.

Refer to 9.7 and 9.8 for further information.

11.3.2 POWER NOTIFICATION (*device*)

The event notifies of a control device power cycle completion and shall be generated following the requirements as stated in 9.7.2 and 9.13.2.

11.4 Device control instructions

11.4.1 General

Device control instructions are used to modify property values of a control device. For this reason a device control instruction shall be discarded, unless it is accepted twice according to the requirements ~~as stated~~ specified in IEC 62386-101:~~2014 and IEC 62386-101:2014/AMD1:2018~~2022, 9.4.

Unless explicitly stated otherwise in the description of the particular device control instruction, the following holds:

- the instruction shall be ignored if so required by the provisions of 9.6;
- the control device shall not reply to the instruction;
- the instruction shall apply to device variables. **28**

11.4.2 IDENTIFY DEVICE

The control device shall start or restart a 10 s ± 1 s identification procedure which shall enable an observer to distinguish any control device(s) running this process from any devices (of the same type) which are not running it. On expiry of this timer, the identification procedure shall stop.

The identification shall be stopped immediately upon execution of any instruction other than "INITIALISE (*device*)" or "IDENTIFY DEVICE".

NOTE 1 Identification can be used during commissioning, allowing an installer to locate devices and allocate the particular identified device to a particular device group.

The indication can be done in various ways, such as by flashing an LED, by producing a sound, other visual or audible means, or alternative methods such as a wireless transmission to a smart device or tool. The exact process used to identify is manufacturer specific and should be described in the manual. In choosing the method, consideration should be given for the availability of the required tools for the intended lifetime of the product.

Support for IDENTIFY DEVICE is optional provided all of the following conditions are met: **29**

- there is no controllable emitter that could be used for identification purposes (such as an LED, buzzer or wireless transmitter), and
- the device can issue an INPUT NOTIFICATION in response to a user generated trigger, such as a push-button or light sensor, and
- a statement is included in the product documents, "This control device does not support identification by means of an LED, buzzer or other emitter."

If any of these conditions are not met, support for IDENTIFY DEVICE shall be implemented.

For the case when IDENTIFY DEVICE is not implemented, it is recommended that the GTIN and serial (identification) number, as used in memory bank 0, are shown on the product label in decimal format. If serial numbers are formatted in hexadecimal, then it is recommended to prefix with "0x".

NOTE 2 Application controllers can support event messages from input devices, as a method of identifying a device during commissioning.

NOTE 3 The application controller can also stop the identification process using a "RESET" command.

Refer to 9.15.3 for further information.

11.4.3 RESET POWER CYCLE SEEN

~~This command shall reset *"powerCycleSeen"* of the receiving control device to FALSE.~~

"powerCycleSeen" shall be set to FALSE.

Refer to 9.13.1 for further information.

11.5 Device configuration instructions

11.5.1 General

Device configuration instructions are used to change the configuration and/or the mode of operation of the control device. For this reason a device configuration instruction shall be discarded, unless it is accepted twice according to the requirements ~~as stated~~ specified in IEC 62386-101:~~2014 and IEC 62386-101:2014/AMD1:2018~~2022, 9.4.

Unless explicitly stated otherwise in the description of the particular device configuration instruction, the following holds:

- the instruction shall be ignored if so required by the provisions of 9.6;
- the control device shall not reply to the instruction;
- the instruction shall apply to device variables. **30**

11.5.2 RESET

All variables shall be changed to their reset values. Control devices shall start to react properly to commands no later than 300 ms after the execution of the instruction has started.

If during a reset mains power fails, it is not guaranteed that "RESET" is completed.

Refer to 9.12.1, Table 19 and Table 20 for further information.

11.5.3 RESET MEMORY BANK (*DTR0*)

The command shall trigger the process to change the memory bank content to its reset values as follows:

- if *"DTR0"* = 0: all implemented and unlocked memory banks except memory bank 0 shall be reset;
- in all other cases: the memory bank identified by *"DTR0"* shall be reset, provided it is implemented and unlocked.

A memory bank needs to be unlocked to allow both lockable and non-lockable locations to be reset.

The control device shall start to react properly to commands no later than 10 s after the execution of the instruction has started.

Refer to 9.12.2 for further information.

11.5.4 SET SHORT ADDRESS (*DTR0*)

The *"shortAddress"* shall be set to *"DTR0"*.

The command shall be discarded if *"DTR0"* does not contain a valid *"shortAddress"* value.

Refer to 9.15.1 for further information.

11.5.5 ENABLE WRITE MEMORY

"writeEnableState" shall be set to ENABLED.

NOTE There is no command to explicitly disable memory write access, since any command that is not directly involved with writing into memory banks will reset *"writeEnableState"* back to DISABLED.

Refer to 9.11.6 for further information.

11.5.6 ENABLE APPLICATION CONTROLLER

If "*applicationControllerPresent*" is TRUE, *"applicationActive"* shall be set to TRUE, otherwise this command shall be discarded.

Refer to 9.10.1 for further information.

11.5.7 DISABLE APPLICATION CONTROLLER

If "*applicationControllerAlwaysActive*" is TRUE, this command shall be discarded.

If "*applicationControllerPresent*" is TRUE, *"applicationActive"* shall be set to FALSE, otherwise this command shall be discarded.

Refer to 9.10.1 and 9.10.2 for further information.

11.5.8 SET OPERATING MODE (*DTR0*)

"operatingMode" shall be set to *"DTR0"*.

If *"DTR0"* does not correspond to an implemented operating mode, the command shall be discarded.

Refer to 9.10.5 for further information.

11.5.9 ADD TO DEVICE GROUPS 0-15 (*DTR2:DTR1*)

The control device shall set those bits in *"deviceGroups[15:0]"* that are set in [*"DTR2:DTR1"*]. The other bits shall not change.

11.5.10 ADD TO DEVICE GROUPS 16-31 (*DTR2:DTR1*)

The control device shall set those bits in *"deviceGroups[31:16]"* that are set in [*"DTR2:DTR1"*]. The other bits shall not change.

11.5.11 REMOVE FROM DEVICE GROUPS 0-15 (*DTR2:DTR1*)

The control device shall clear those bits in *"deviceGroups[15:0]"* that are set in [*"DTR2:DTR1"*]. The other bits shall not change.

11.5.12 REMOVE FROM DEVICE GROUPS 16-31 (*DTR2:DTR1*)

The control device shall clear those bits in *"deviceGroups[31:16]"* that are set in [*"DTR2:DTR1"*]. The other bits shall not change.

11.5.13 START QUIESCENT MODE

The control device shall start or restart quiescent mode by setting *"quiescentMode"* to ENABLED and (re-)triggering the timer.

Refer to 9.10.4 for further information.

11.5.14 STOP QUIESCENT MODE

"quiescentMode" shall be set to DISABLED.

Refer to 9.10.4 for further information.

11.5.15 ENABLE POWER CYCLE NOTIFICATION

"powerCycleNotification" shall be set to ENABLED.

Refer to 9.13.2 for further information.

11.5.16 DISABLE POWER CYCLE NOTIFICATION

"powerCycleNotification" shall be set to DISABLED.

Refer to 9.13.2 for further information.

~~11.5.17 SAVE PERSISTENT VARIABLES~~ 31

~~The control device shall physically store all device and instance variables identified in Table 17 and Table 18 as non-volatile memory (NVM). This shall include all device and instance NVM variables defined in the applicable Parts 3xx.~~

~~The control device might not react to commands after execution of this command. Control devices shall start to react properly to commands no later than 300 ms after the execution of the instruction has started.~~

~~This command is recommended to be used typically after commissioning. Due to the limited number of write-cycles of persistent memory, application controlers should limit the use of this command. Internal processing of data might be suspended while this command is being executed.~~

~~Refer to Table 17, Table 18 and 9.17 for further information.~~

11.5.17 SET EVENT PRIORITY (*DTR0*) 32

The command shall be discarded if *"DTR0"* is not in the range [2,5].

"eventPriority" shall be set to *"DTR0"*.

11.6 Device queries

11.6.1 General

Device queries are used to retrieve device property values from a control device. The addressed control device returns the queried property value in a backward frame.

Unless explicitly stated otherwise in the description of the particular device query, the following holds:

- the query shall be ignored if so required by the provisions of 9.6;
- the query shall apply to device variables. 33

When applicable, the query shall be discarded if any of the parameter values (in *"DTR0"*, *"DTR1"* and *"DTR2"*) are outside the range of validity of the addressed device variables, as given in Table 19.

11.6.2 QUERY DEVICE CAPABILITIES

The answer shall be a combination of control device capabilities.

Refer to 9.17.1 for further information.

11.6.3 QUERY DEVICE STATUS

The answer shall be the status, which is formed by a combination of control device properties.

Refer to 9.17.2 for further information.

11.6.4 QUERY APPLICATION CONTROLLER ERROR

The answer shall be the detailed error information regarding an application controller:

- if an error in the application controller has occurred (as indicated by *"applicationControllerError"*), but the device is not able to give detailed error information: MASK;
- if an error in the application controller has occurred (as indicated by *"applicationControllerError"*), and the device is able to give detailed error information: error number [0,254];
- if no application controller error has occurred: NO.

Detailed error information is manufacturer specific and should be described in product documentation.

Refer to 9.16 for further information.

11.6.5 QUERY INPUT DEVICE ERROR

The answer shall be the detailed error information regarding an input device:

- if an error in the input device has occurred (as indicated by *"inputDeviceError"*), but the device is not able to give detailed error information: MASK;
- if an error in the input device has occurred (as indicated by *"inputDeviceError"*), and the device is able to give detailed error information: error number [0,254];
- if no input device error has occurred: NO.

Detailed error information is manufacturer specific and should be described in product documentation.

Refer to 9.16 for further information.

11.6.6 QUERY MISSING SHORT ADDRESS

The answer shall be YES if *"shortAddress"* is equal to MASK and NO otherwise.

NOTE Since the control device answers only if no short address is stored, the use of the command is useful only in broadcast mode or when device group addressing is used.

11.6.7 QUERY VERSION NUMBER

The answer shall be ~~the content of memory bank 0 location 0x17~~ "*versionNumber*".

See 4.2 and Table 19 for more information.

11.6.8 QUERY CONTENT DTR0

The answer shall be *"DTR0"*.

11.6.9 QUERY NUMBER OF INSTANCES

The answer shall be *"numberOfInstances"*.

Refer to 9.5 for further information.

11.6.10 QUERY CONTENT DTR1

The answer shall be *"DTR1"*.

11.6.11 QUERY CONTENT DTR2

The answer shall be *"DTR2"*.

11.6.12 QUERY RANDOM ADDRESS (H)

The answer shall be *"randomAddress[23:16]"*.

11.6.13 QUERY RANDOM ADDRESS (M)

The answer shall be *"randomAddress[15:8]"*.

11.6.14 QUERY RANDOM ADDRESS (L)

The answer shall be *"randomAddress[7:0]"*.

11.6.15 READ MEMORY LOCATION (*DTR1, DTR0*)

The query shall be discarded if the memory bank identified by *"DTR1"* is not implemented.

If executed, the answer shall be the content of the memory location identified by offset *"DTR0"* within memory bank *"DTR1"*.

The control device shall answer NO if the addressed memory location is not implemented.

NOTE 1 This allows ~~holes~~ gaps in the memory bank implementation.

If the addressed offset is below location 0xFF in the bank, the control device shall increment *"DTR0"* by one.

NOTE 2 This allows efficient multi-byte reading within a transaction.

Refer to 9.11.5 for further information.

11.6.16 QUERY APPLICATION CONTROLLER ENABLED

The answer shall be YES if *"applicationActive"* is TRUE, NO otherwise.

Refer to 9.10.1 for further information.

11.6.17 QUERY OPERATING MODE

The answer shall be *"operatingMode"*.

Refer to 9.10.5 for further information.

11.6.18 QUERY MANUFACTURER SPECIFIC MODE

The answer shall be YES when *"operatingMode"* is in the range [0x80,0xFF] and NO otherwise.

Refer to 9.10.5 for further information.

11.6.19 QUERY QUIESCENT MODE

The answer shall be YES if *"quiescentMode"* is ENABLED, and NO otherwise.

Refer to 9.10.4 for further information.

11.6.20 QUERY DEVICE GROUPS 0-7

The answer shall be *"deviceGroups[7:0]"*.

11.6.21 QUERY DEVICE GROUPS 8-15

The answer shall be *"deviceGroups[15:8]"*.

11.6.22 QUERY DEVICE GROUPS 16-23

The answer shall be *"deviceGroups[23:16]"*.

11.6.23 QUERY DEVICE GROUPS 24-31

The answer shall be *"deviceGroups[31:24]"*.

11.6.24 QUERY POWER CYCLE NOTIFICATION

The answer shall be YES if *"powerCycleNotification"* is ENABLED, and NO otherwise.

Refer to 9.13.2 for further information.

11.6.25 QUERY EXTENDED VERSION NUMBER(*DTR0*)

The answer shall be the version number of the relevant part of the IEC 62386-3xx series where xx is given by *DTR0*.

The answer shall be:

- if the Part 3xx given by *DTR0* is not implemented: NO;
- if the Part 3xx given by *DTR0* is implemented: the version number of the Part 3xx.

Refer to the relevant part of the IEC 62386-3xx series for further information.

11.6.26 QUERY RESET STATE

The answer shall be YES if *"resetState"* is TRUE, and NO otherwise.

Refer to 9.17.2 for further information.

11.6.27 QUERY APPLICATION CONTROLLER ALWAYS ACTIVE

The answer shall be YES if "*applicationControllerAlwaysActive*" is TRUE, and NO otherwise.

Refer to 9.10.2 for further information.

11.6.28 QUERY FEATURE TYPE

See 11.9.14 for command execution and 9.2 for information on device features.

11.6.29 QUERY NEXT FEATURE TYPE

See 11.9.15 for command execution and 9.2 for information on device features.

11.6.30 QUERY EVENT PRIORITY 34

See 11.9.13 for command execution.

11.7 Instance control instructions

Instance control instructions are used to modify property values of an instance of an input device.

Unless explicitly stated otherwise in the description of the particular instance control instruction, the following holds:

- the instruction shall be ignored if so required by the provisions of 9.6;
- the input device shall not reply to the instruction;
- the instruction shall apply to instance variables. 35

NOTE This document does not describe any instance control instructions. However, the above requirements do apply to instance instructions described in the relevant parts of the IEC 62386-3xx series.

11.8 Instance configuration instructions

11.8.1 General

Instance configuration commands are used to change the configuration and/or the mode of operation of an instance within the input device. For this reason an instance configuration instruction shall be discarded, unless it is accepted twice according to the requirements ~~as stated~~ specified in IEC 62386-101:~~2014 and IEC 62386-101:2014/AMD1:2018~~2022, 9.4.

Unless explicitly stated otherwise in the description of the particular instance configuration instruction, the following holds:

- the instruction shall be ignored if so required by the provisions of 9.6;
- the input device shall not reply to the instruction;
- the instruction shall apply to instance variables. **36**

11.8.2 ENABLE INSTANCE

"instanceActive" shall be set to TRUE.

Refer to 9.10.3 for further information.

11.8.3 DISABLE INSTANCE

"instanceActive" shall be set to FALSE.

Refer to 9.10.3 for further information.

11.8.4 SET PRIMARY INSTANCE GROUP (*DTR0*)

The instance shall have the primary membership to an instance group assigned or removed, by setting *"instanceGroup0"* to *"DTR0"*.

The command shall be discarded if *"DTR0"* is not in the range [0,31] and different from MASK.

Refer to 9.5.5 for further information.

11.8.5 SET INSTANCE GROUP 1 (*DTR0*)

The instance shall have an additional membership to an instance group assigned or removed, by setting *"instanceGroup1"* to *"DTR0"*.

The command shall be discarded if *"DTR0"* is not in the range [0,31] and different from MASK.

Refer to 9.5.5 for further information.

11.8.6 SET INSTANCE GROUP 2 (*DTR0*)

The instance shall have an additional membership to an instance group assigned or removed, by setting *"instanceGroup2"* to *"DTR0"*.

The command shall be discarded if *"DTR0"* is not in the range [0,31] and different from MASK.

Refer to 9.5.5 for further information.

11.8.7 SET EVENT SCHEME (*DTR0*)

The instance shall, if the provisions identified in 9.7.3 allow so, apply a new event addressing scheme for subsequent "INPUT NOTIFICATION (*device/instance, event*)" events by setting *"eventScheme"* to *"DTR0"*.

NOTE Circumstances ~~may~~ can dictate that the operation cannot be granted by the receiving instance, meaning that the answer to the corresponding "QUERY EVENT SCHEME" can be different from the event scheme requested here.

The command shall be discarded if *"DTR0"* is not in the range [0,4].

Refer to 9.7.3 for further information.

11.8.8 SET EVENT PRIORITY (*DTR0*)

The instance shall apply a new message priority for subsequent "INPUT NOTIFICATION (*device/instance, event*)" events by setting *"eventPriority"* to *"DTR0"*.

The command shall be discarded if *"DTR0"* is not in the range [2,5].

Refer to 9.14 for further information.

11.8.9 SET EVENT FILTER (*DTR2:DTR1:DTR0*)

The ~~control device~~ instance shall set *"eventFilter[23:0]"* to [*"DTR2:DTR1:DTR0"*].

Refer to 9.7.4 for further information.

11.8.10 SET INSTANCE TYPE (*DTR0*) 37

The instance shall discard this command if any of the following conditions are true:

- configuration of the instance type is not supported by the instance, or
- *"DTR0[7:5]"* is non-zero, or
- *"DTR0[4:0]"* is not a supported instance type.

If executed, *"instanceType"* shall be set to *"DTR0[4:0]"*.

NOTE Application controllers can query the instance configuration after execution of this command, to check which instance configuration was activated.

A change of *"instanceType"* shall cause the instance variables of the addressed instance to be set to their reset values, and can cause the device to re-start with all other variables being set to the power-on values. In the case the device re-starts, the system start-up timing shall be met (see IEC 62386-101:2022, 4.11.6).

11.8.11 SET INSTANCE CONFIGURATION (*DTR0, DTR2:DTR1*) 38

The instance shall discard this command if any of the following conditions are true:

- configuration of *"instanceConfiguration[DTR0]"* is not supported by the instance, or
- *"DTR0"* < 191, and points to a value of *"instanceConfiguration[]"* that is not defined in the corresponding part of the IEC 62386-3xx series, or
- *"DTR0"* is 191, and *"DTR2:DTR1"* is not 0x55CC.

NOTE 1 *"DTR0"* values in the range [192, 255] allow manufacturer-specific instance configuration.

If executed and *"DTR0"* is not 191, then *"DTR2:DTR1"* shall be written to *"instanceConfiguration[DTR0]"*, with any unused bits of the 16-bit value discarded.

If *"DTR0"* is 191, and *"DTR2:DTR1"* is 0x55CC, then all implemented locations of *"instanceConfiguration[]"* shall be set to their factory default values.

A change of *"instanceConfiguration[DTR0]"* can cause the device to re-start with all variables being set to the power-on values. In the case the device re-starts, all further frames may be discarded until the re-start has completed. The system start-up timing shall be met (see IEC 62386-101:2022, 4.11.6).

NOTE 2 Application controllers can send query commands to determine if the device is ready to receive further frames, so allowing for the possibility of the device re-starting.

11.9 Instance queries

11.9.1 General

Instance queries are used to retrieve instance property values from an instance of an input device. The addressed input device returns the property value queried in a backward frame.

Unless explicitly stated otherwise in the description of the particular instance query, the following holds:

- the query shall be ignored if so required by the provisions in 9.6;
- if the query addresses multiple instances within the input device, the query shall be answered as if each instance is a logical device (see IEC 62386-101:~~2014 and IEC 62386-101:2014/AMD1:2018~~2022, 9.6.2)
- the query shall apply to instance variables. **39**

11.9.2 QUERY INSTANCE TYPE

The answer shall be *"instanceType"*.

Refer to 9.5.3 for further information.

11.9.3 QUERY RESOLUTION

The answer shall be *"resolution"*.

Refer to 9.8.2 for further information.

11.9.4 QUERY INSTANCE ERROR

The answer shall be detailed error information:

- if an error has occurred (as indicated by *"instanceError"*), but the instance is not able to give detailed error information: 0;
- if an error has occurred (as indicated by *"instanceError"*), and the instance is able to give detailed error information: error number [1,255];
- if no error has occurred: NO.

NOTE Detailed error information is instance type specific and is described in the IEC 62386-3xx series.

Refer to 9.16 for further information.

11.9.5 QUERY INSTANCE STATUS

The command queries the status of a combination of instance properties.

Refer to 9.17.3 for further information.

11.9.6 QUERY INSTANCE ENABLED

The answer shall be YES, if *"instanceActive"* is TRUE in at least one of the addressed instances, and NO otherwise.

Refer to 9.10.3 for further information.

11.9.7 QUERY PRIMARY INSTANCE GROUP

The answer shall be *"instanceGroup0"*.

Refer to 9.5.5 for further information.

11.9.8 QUERY INSTANCE GROUP 1

The answer shall be *"instanceGroup1"*.

Refer to 9.5.5 for further information.

11.9.9 QUERY INSTANCE GROUP 2

The answer shall be *"instanceGroup2"*.

Refer to 9.5.5 for further information.

11.9.10 QUERY EVENT SCHEME

The answer shall be *"eventScheme"*.

Refer to 9.7.3 for further information.

11.9.11 QUERY INPUT VALUE

The instance shall latch a new *"inputValue"* and reply with the most significant byte of the latched input value.

If the query addresses multiple instances within the input device, it shall be ~~ignored~~ discarded.

Refer to 9.8.3 for further information.

11.9.12 QUERY INPUT VALUE LATCH

The instance shall reply with the next byte from a latched *"inputValue"*.

Following the least significant byte of the latched input value, the answer shall be NO, until a new "QUERY INPUT VALUE" has been executed.

If the query addresses multiple instances within the input device, it shall be ~~ignored~~ discarded.

Refer to 9.8.3 for further information.

11.9.13 QUERY EVENT PRIORITY

The answer shall be *"eventPriority"*.

Refer to 9.14 for further information.

11.9.14 QUERY FEATURE TYPE

The answer shall be:

- if no feature 3xx is implemented: 254;
- if one device/instance feature is supported: the device/instance feature number;
- if more than one device/instance feature is supported: MASK.

Refer to 9.5.4 for further information.

11.9.15 QUERY NEXT FEATURE TYPE

The answer shall be:

- if directly preceded by "QUERY FEATURE TYPE", and more than one device/instance feature is supported: the first and lowest device/instance feature number;
- if directly preceded by "QUERY NEXT FEATURE TYPE", and not all device/instance features have been reported: the next lowest device/instance feature number;
- if directly preceded by "QUERY NEXT FEATURE TYPE", and all feature types have been reported: 254;
- in all other cases: NO.

The sequence of commands shall only be accepted as long as they use the same combination of address byte and instance byte ~~combination~~. Multi-master transmitters ~~shall~~ should send such sequences as a transaction.

Refer to 9.5.4 for further information.

11.9.16 QUERY EVENT FILTER 0-7

The answer shall be *"eventFilter[7:0]"*.

Refer to 9.7.4 for further information.

11.9.17 QUERY EVENT FILTER 8-15

The answer shall be ~~*"eventFilter[8:15]"*~~ *"eventFilter[15:8]"*.

Refer to 9.7.4 for further information.

11.9.18 QUERY EVENT FILTER 16-23

The answer shall be ~~*"eventFilter[16:23]"*~~ *"eventFilter[23:16]"*.

Refer to 9.7.4 for further information.

11.9.19 QUERY INSTANCE CONFIGURATION (*DTR0*) 40

If *"instanceConfiguration[DTR0]"* is not implemented, there shall be no reply, otherwise the answer shall be the least significant byte of *"instanceConfiguration[DTR0]"*, and "*DTR2:DTR1*" shall be set to *"instanceConfiguration[DTR0]"*, padded with zeros for unused bits.

If *"DTR0"* is 191, the reply shall be MASK and "*DTR2:DTR1*" shall be set to 0xFFFF if all other implemented values of *"instanceConfiguration[]"* are at the factory default values, otherwise the reply shall be 0xAA and "*DTR2:DTR1*" shall be set to 0xAAAA.

EXAMPLE If *DTR0* = 200, and *instanceConfiguration*[200] only has possible values in the range [0, 15], then this query will result in *DTR1* and the answer being set to the corresponding value in the range [0, 15], and *DTR2* being set to 0.

Refer to 9.19 for further information.

11.9.20 QUERY AVAILABLE INSTANCE TYPES 41

The answer shall be a bit-field containing one bit for each instance type 0 to 31. For each available instance type, the corresponding bit shall be set to 1, otherwise the corresponding bit shall be 0. The 32 bits are encoded in the answer and in "*DTR0*" through "*DTR2*" as follows:

- answer: bits [7:0] representing instance types 7 to 0,
- "*DTR0*": bits [15:8] representing instance types 15 to 8,
- "*DTR1*": bits [23:16] representing instance types 23 to 16,
- "*DTR2*": bits [31:24] representing instance types 31 to 24.

Refer to 9.19 for further information.

11.10 Special commands

11.10.1 General

All special mode commands shall be interpreted as instructions unless explicitly stated otherwise.

11.10.2 TERMINATE

The following processes shall be terminated immediately upon execution of this ~~command~~ instruction:

- Initialisation, *"initialisationState"* shall be set to DISABLED;
- Identification, as ~~a standard operation~~ started by "IDENTIFY DEVICE".

The ~~command~~ instruction could also terminate other processes as identified in the relevant parts of the IEC 62386-3xx series.

11.10.3 INITIALISE (*device*)

This ~~command~~ instruction shall be discarded, unless it is accepted twice according to the requirements ~~as stated~~ specified in IEC 62386-101:~~2014 and IEC 62386-101:2014/AMD1:2018~~2022, 9.4.

Only devices matching the given *device* shall respond to the ~~command~~ instruction, as indicated in Table 25:

Table 25 – Device addressing with "INITIALISE (*device*)"

device	Responsive device(s)
00AAAAAAb	Device(s) with *"shortAddress"* equal to 00AAAAAAb
01111111b	Devices with *"shortAddress"* equal to MASK
MASK	All devices
Other	None

The ~~command~~ instruction shall start or prolong the initialisation state, by setting *"initialisationState"* to ENABLED if it was DISABLED and (re-)trigger the timer. There shall be no answer.

Refer to 9.15 for further information.

11.10.4 RANDOMISE

This instruction shall be discarded, unless it is accepted twice according to the requirements ~~as stated~~ specified in IEC 62386-101:~~2014 and IEC 62386-101:2014/AMD1:2018~~2022, 9.4.

The instruction shall be ~~ignored~~ discarded if *"initialisationState"* is equal to DISABLED.

If executed, the instruction shall generate a random value for "*randomAddress*", in the range of [0x000000,0xFFFFFE] which shall be available within 100 ms for use.

If there are multiple logical units present and the ~~command~~ instruction is executed using broadcast addressing, the generated random addresses within the bus unit shall be unique, i.e. every logical unit shall have a ~~random address~~ value of "*randomAddress*" that is not found in any of the other logical units contained in the bus unit.

There shall be no reply to this instruction.

Refer to 9.15 for further information.

11.10.5 COMPARE

The query shall be discarded unless *"initialisationState"* is ENABLED.

If executed, the control device shall answer:

- if "randomAddress" ≤ "searchAddress": YES;
- in all other cases: NO.

Refer to 9.15 for further information.

11.10.6 WITHDRAW

The instruction shall be discarded unless the following conditions hold:

- *"initialisationState"* is equal to ENABLED, and
- *"randomAddress"* is equal to *"searchAddress"*.

If the instruction is executed, the control device shall change *"initialisationState"* to WITHDRAWN.

NOTE 1 Before withdrawing a control device, the application controller ~~may want to~~ can assign it a short address using "PROGRAM SHORT ADDRESS (*data*) ".

NOTE 2 The effect is that the control device is excluded from subsequent "COMPARE" operations, thus allowing the application controller to conduct a (binary) search operation across all devices until the "COMPARE" query leads to no answer (from any control device) on the bus.

Refer to 9.15 for further information.

11.10.7 SEARCHADDRH (*data*)

The instruction shall be discarded if *"initialisationState"* is equal to DISABLED.

If the instruction is executed, *"searchAddress[23:16]"* shall be set to the given *data*.

Refer to 9.15 for further information.

11.10.8 SEARCHADDRM (*data*)

The instruction shall be discarded if *"initialisationState"* is equal to DISABLED.

If the instruction is executed, *"searchAddress[15:8]"* shall be set to the given *data*.

Refer to 9.15 for further information.

11.10.9 SEARCHADDRL (*data*)

The instruction shall be discarded if *"initialisationState"* is equal to DISABLED.

If the instruction is executed, *"searchAddress[7:0]"* shall be set to the given *data*.

Refer to 9.15 for further information.

11.10.10 PROGRAM SHORT ADDRESS (*data*)

The instruction shall be discarded unless the following conditions hold:

- *"initialisationState"* is equal to ENABLED or WITHDRAWN;
- *"randomAddress"* is equal to *"searchAddress"*;
- *data* contains a valid *"shortAddress"* value.

NOTE The format for *data* for PROGRAM SHORT ADDRESS (*data*) is different from the format used for the equivalent command in IEC 62386-102. **42**

If the instruction is executed, the *"shortAddress"* shall be set to *data*.

Refer to 9.15 for further information.

11.10.11 VERIFY SHORT ADDRESS (*data*)

The query shall be discarded if *"initialisationState"* is equal to DISABLED.

If the query is executed, the answer shall be YES if *"shortAddress"* is equal to the given *data* (*data* given in 00AAAAAAb format), and NO otherwise.

Refer to 9.15 for further information.

11.10.12 QUERY SHORT ADDRESS

The query shall be discarded if:

- *"initialisationState"* is equal to DISABLED, or
- *"randomAddress"* is not equal to *"searchAddress"*.

If the query is executed, the answer shall be *"shortAddress"*.

Refer to 9.15 for further information.

11.10.13 WRITE MEMORY LOCATION (*DTR1, DTR0, data*)

The instruction shall be discarded if any of the following conditions hold:

- the addressed memory bank is not implemented, or
- *"writeEnableState"* is DISABLED.

NOTE 1 This operation is a broadcast operation. Selective control device addressing can be achieved by setting the write enable condition selectively.

If the ~~command~~ instruction is executed, the control device shall write *data* into the memory location identified by offset *"DTR0"* within memory bank *"DTR1"* and return *data* as an answer.

NOTE 2 Simultaneous writing to multiple control devices will probably lead to framing errors because of colliding answers.

NOTE 3 The value that can be read from the memory bank location is not necessarily *data*.

If the selected memory bank location is

- not implemented, or
- above the last accessible memory location, or
- locked (see 9.11.2), or
- not writeable,

the answer to WRITE MEMORY LOCATION (*DTR1, DTR0, data*) shall be NO and no memory location shall be written to.

If the addressed ~~offset~~ location is below location 0xFF in the bank, the control device shall increment *"DTR0"* by one.

NOTE 4 This allows efficient multi-byte writing within a transaction.

Refer to 9.11.6 for further information.

11.10.14 WRITE MEMORY LOCATION – NO REPLY (*DTR1, DTR0, data*)

This instruction is identical to the "WRITE MEMORY LOCATION (*DTR1, DTR0, data*)" command (see 11.10.13), except that the receiving control device shall not reply to the command.

Refer to 9.11.6 for further information.

11.10.15 DTR0 (*data*)

"DTR0" shall be set to the given *data*.

11.10.16 DTR1 (*data*)

"DTR1" shall be set to the given *data*.

11.10.17 DTR2 (*data*)

"DTR2" shall be set to the given *data*.

11.10.18 DIRECT WRITE MEMORY (*DTR1, offset, data*)

The instruction shall be discarded if any of the following conditions hold:

- the memory bank identified by *"DTR1"* is not implemented, or
- *"writeEnableState"* is DISABLED.

NOTE This operation is a broadcast operation. Selective control device addressing can be achieved by setting the write enable condition selectively.

If the command is executed, the control device shall copy the value of *offset* to "*DTR0*", then execute WRITE MEMORY LOCATION (*DTR1, DTR0, data*).

Refer to 9.11.6 for further information.

11.10.19 DTR1:DTR0 (*data1, data0*)

"DTR1" shall be set to the given *data1*; *"DTR0"* shall be set to the given *data0*.

11.10.20 DTR2:DTR1 (*data2, data1*)

"DTR2" shall be set to the given *data2*; *"DTR1"* shall be set to the given *data1*.

11.10.21 SEND TESTFRAME (*data*)

data shall be interpreted as *data*(CTARRPPPb).

The instruction shall be executed unless any of the following conditions hold:

- Cb ≠ 0;
- PPPb > 5;
- PPPb < 1;
- Ab = 1 and "*applicationControllerPresent*" = FALSE.

If executed, and Ab = 0: a 24-bit forward frame shall be sent with the following content:

- Address byte: *"DTR0";*
- Instance byte: *"DTR1";*
- Opcode byte: *"DTR2".*

If executed, and Ab = 1: a 16-bit forward frame shall be sent with the following content:

– Address byte: *"DTR0";*
– Opcode byte: *"DTR1".*

As the command is not addressable, the command shall be executed once per bus unit independent of its number of instances and/or logical devices. This also implies that if at least one application controller is present, the 16-bit forward frames shall be sent.

The forward frame shall be sent using priority PPPb and shall then be repeated RRb times. If Tb = 1, the frames shall be sent in a transaction. If Tb=0, the repeated frames shall be sent using the priority set by PPPb.

This command is used ~~within the test procedures~~ to test collision detection for multi-master application controller and input devices. If a collision is detected while the test frame is transmitted, the collision shall be dealt with. In the case of a transaction, the entire transaction shall be retransmitted. In the case of repeating frames, only the frame(s) containing the collision(s) shall be repeated.

~~12 Test procedures~~ 43

~~12.1 General notes on test~~

~~12.1.1 General~~

~~The requirements of IEC 62386-101:2014, 12.1 apply also for control device tests.~~

~~12.1.2 Test execution~~

~~Subclause 12.2 is meant to prepare the DUT for testing, by~~

- ~~setting the global variables;~~
- ~~assigning a short address to each logical unit;~~
- ~~getting information on which logical units need to be tested.~~

~~Each test sequence indicates whether it shall be run for all logical devices in parallel or per selected logical device.~~

~~If a bus powered device containing an internal bus power supply is tested according to this part, such a device shall be handled as a bus powered device in all tests. There shall be no external power supply, effectively keeping the internal bus power supply off during testing.~~

~~If a device contains a bus power supply, the tests as defined in IEC 62386-101 shall be executed.~~

~~Before a test is executed, the nominal voltage *GLOBAL_internalVoltage* and current *GLOBAL_Ibus* shall be restored.~~

~~12.1.3 Data transmission~~

~~12.1.3.1 Device addressing based on test category~~

~~If not mentioned otherwise, each command shall be sent using one of the following device addressing modes:~~

- ~~broadcast, if the test shall be run for all logical units in parallel;~~
- ~~short address of the logical unit under test, if the test shall be run for each selected logical unit.~~

~~When a command has to be sent to a different device address, the address shall be given in the form as follows:~~

- ~~COMMAND, send to device *address byte*.~~

~~12.1.3.2 Instance addressing~~

~~When an instance command has to be sent the address shall be given as follows:~~

- ~~COMMAND, send to instance *instance byte*.~~

~~12.1.3.3 Feature addressing~~

~~When a feature command has to be sent the address shall be given as follows:~~

- COMMAND, send to instance feature *instance byte*.
- COMMAND, send to device feature *address byte*.

12.1.3.4 Based on type of command

If not mentioned otherwise, the configuration commands shall be sent twice. When a command needs to be sent once, it is noted as:

- COMMAND, send once.

12.1.4 Test setup

The DUT shall be connected to the bus interface and mains power (if applicable).

The power supply shall be set to the values defined in 12.2 by GLOBAL_VBusHigh, GLOBAL_VBusLow, GLOBAL_Ibus.

If not mentioned otherwise, the tester shall use the fall time, rise time, half bit time, double half time, and settling time (between any frame and a forward frame) given in the global variables. Moreover, the power supply shall be adjusted to the values defined by the global variables.

12.1.5 Test output

Each output message of the tests executed on a logical unit shall be preceded by the string "LogicalUnit" followed by the short address of the logical unit under test. The output message shall look as:

error number LogicalUnit 1: string
report number LogicalUnit 1: string
warning number LogicalUnit 1: string
halt number LogicalUnit 1: string

12.1.6 Test notation

12.1.6.1 Command tables

A dash ("-") given in any table for "command" should be interpreted as "send nothing".

12.1.6.2 Send special waveforms

This instruction is used to send a waveform with special timing requirements, like particular settling times or pulse times.

The very next frame after sending the complete waveform will be returned. If there is no activity until the timeout is reached, NO will be returned.

next_frame = **SendWaveform** (COMMAND, e.g. specific settling time, another COMMAND, …)

This function is used, to define a trigger condition on which an active state pulse is send by the tester.

Additionally the time between the trigger event and the start of the pulse can be defined and also the duration of the pulse itself. The pulse is sent once after the trigger condition is met and is not repeated on other occurrences of the trigger condition.

SetPulseTrigger(Trigger Condition, Pulse Start Delay, Pulse Duration)

12.1.6.3 Bus **recording**

This command is used to start a bus recording for later analysis of the recorded signal. This command clears previous recordings.

StartBusRecording (Recording Name)

Additionally a trigger condition can be provided to define the exact starting point of the recording. This command also clears previous recordings.

StartBusRecordingTrigger (Trigger Condition, Recording Name)

This command is used to immediately stop an active recording.

StopBusRecording (Recording Name)

12.1.6.4 Signal anlysis

This command is used to parse a recording for a certain command. The return value shall be the number of occurences of the specified command within this part of the recording or zero if no occurences found.

NumberContained = **FindFrame**(Recording Name, Frame)

This command returns the time difference between a specified reference point and the start – first falling edge – of the first occurrence of the specified frame after the reference point within a recording. If after the reference point in the recording no searched frame is contained the command shall return a negative number.

Time = **FindFrameStart**(Trigger Condition, Recording Name, Frame)

This command returns the time difference between a specified reference point and the start of the first occurrence of a stop condition after the reference point within a recording. If the bus is already idle at the reference point and continues to be idle for the time period of a stop condition, the command shall return zero.

If after the reference point in the recording no stop condition is contained the command shall return a negative number.

Time = **FindStopConditionStart**(Trigger Condition, Recording Name)

This command is used to parse a recording for a break condition (minimum 1,2 ms active state). The return value shall be true if a break condition is part of the recording at least once or false if not.

Contains = **FindBreakCondition**(Recording)

This command returns the time difference between a specified reference point and the start of the first occurrence of a break condition (minimum 1,2 ms active state) after the reference point within a recording. If the bus is already in active state at the reference point and continues to be idle for the break time, the command shall return zero.

If after the reference point in the recording no break condition is contained, the command shall return a negative number.

Time = **FindBreakConditionStart**(Trigger Condition, Recording)

This command returns the time difference between two specified reference points within a recording. The time difference is calculated on the basis of Trigger Condition 2 – Trigger Condition 1.

Time = **TimeDifference** (Trigger Condition 1, Trigger Condition 2, Recording)

12.1.7 Test execution limitations

The current test procedures can be executed on a DUT with a maximum of 63 logical units. Short address 63 is reserved for testing purpose.

12.1.8 Test results

A DUT shall be claimed to be compliant to the IEC 62386 standard only if all tests are passed without any error for all logical units.

12.1.9 Exception handling

Whenever within a test procedure an unexpected incident occurs, making it senseless to continue the test procedure, the current test procedure – or series of test procedures – shall be aborted at this point.

12.1.10 Unexpected answer

Whenever within a test procedure a command is sent, a series of possible outcomes can follow:

- Backward Frame;
- No Answer;
- Any Violation (Bit Timing, Frame Sequence, Frame Size).

Depending on the command, the answer has to follow certain constraints:

- a query for a value has to be answered with a valid backward frame containing a value within a certain range of validity;
- a Yes/No query has to be answered with “No Answer” or a valid backward frame containing 255;
- other commands have no answer.

If there is any unexpected outcome, a general error shall be reported followed by an exception handling (see 12.1.9).

Often such incidents do not indicate a malfunction against the subject of the current test procedure, but a communication problem in general.

Table 24 shows all commands and their unexpected outcomes.

Table 24 – Unexpected outcome

Command name	unexpected		
	Violation	Value	No Answer
QUERY DEVICE STATUS	✓	[64, 255]	✓
QUERY APPLICATION CONTROLLER ERROR	✓		
QUERY INPUT DEVICE ERROR	✓		
QUERY MISSING SHORT ADDRESS	✓	[0, 254]	
QUERY VERSION NUMBER	✓	[0, 7], [8, 255]	✓
QUERY NUMBER OF INSTANCES	✓	[32, 255]	✓
QUERY CONTENT DTR0	✓		✓
QUERY CONTENT DTR1	✓		✓
QUERY CONTENT DTR2	✓		✓
QUERY RANDOM ADDRESS (H)	✓		✓
QUERY RANDOM ADDRESS (M)	✓		✓
QUERY RANDOM ADDRESS (L)	✓		✓
READ MEMORY LOCATION (*DTR1, DTR0*)	✓		
QUERY APPLICATION CONTROLLER ENABLED	✓	[0, 254]	
QUERY OPERATING MODE	✓		✓
QUERY MANUFACTURER SPECIFIC MODE	✓	[0, 254]	
QUERY QUIESCENT MODE	✓	[0, 254]	
QUERY DEVICE GROUPS 0-7	✓		✓
QUERY DEVICE GROUPS 8-15	✓		✓
QUERY DEVICE GROUPS 16-23	✓		✓
QUERY DEVICE GROUPS 24-31	✓		✓
QUERY POWER CYCLE NOTIFICATION	✓	[0, 254]	
QUERY DEVICE CAPABILITIES	✓	0, [4, 255]	✓
QUERY EXTENDED VERSION NUMBER (*DTR0*)			
QUERY RESET STATE			
QUERY INSTANCE TYPE	✓	[32, 255]	✓
QUERY RESOLUTION	✓	0	✓
QUERY INSTANCE ERROR	✓		
QUERY INSTANCE STATUS	✓	[4, 255]	✓
QUERY EVENT PRIORITY	✓	[0, 1], [6, 255]	✓
QUERY INSTANCE ENABLED	✓	[0, 254]	✓
QUERY PRIMARY INSTANCE GROUP	✓	[32, 254]	✓
QUERY INSTANCE GROUP 1	✓	[32, 254]	✓
QUERY INSTANCE GROUP 2	✓	[32, 254]	✓
QUERY EVENT SCHEME	✓	[5, 255]	✓
QUERY INPUT VALUE	✓		✓
QUERY INPUT VALUE LATCH	✓		
QUERY FEATURE TYPE	✓	[0, 31], [97, 253]	✓

Command name	unexpected		
	Violation	Value	No Answer
QUERY NEXT FEATURE TYPE	✓	[0, 31], [97, 253], 255	
QUERY EVENT FILTER 0-7	✓		
QUERY EVENT FILTER 8-15	✓		
QUERY EVENT FILTER 16-23	✓		
COMPARE	✓	[0, 254]	
VERIFY SHORT ADDRESS (data)	✓	[0, 254]	
QUERY SHORT ADDRESS	✓	[64, 254]	
WRITE MEMORY LOCATION (DTR1, DTR0, data)	✓		
DIRECT WRITE MEMORY (DTR1, offset, data)	✓		
SEND TESTFRAME (*data*)			✓
Any other command	✓	[0,255]	

If in certain cases a test procedure has to check on an exception, e.g. no answer when using a different address than the DUT is configured for, this can be indicated by using the "**accept**" statement followed by the exceptions which shall be excluded from the general exception handling. The exception then is returned to the function caller and can be handled there individually.

12.2 Preamble

12.2.1 Test preamble

The test preamble sets the global parameters, checks the default values if the device is factory new, and assigns to each logical unit a short address equal to its index. For each logical unit the following information is stored:

- instance Types (an array showing the types of the implemented instances);
- feature Types (a sub-array of instances, showing the types of the implemented features);
- extendedVersionNumber (an array of extended version numbers of Parts 103 and 3xx);
- runTests (indicates whether tests shall be performed for that logical unit).

Test sequence shall be run for all logical units in parallel.

Test description:

// Set global parameters
GLOBAL_VbusHigh = 16 // in V - Default high voltage for testing
GLOBAL_VbusLow = 0 // in V - Default low voltage for testing
GLOBAL_Ibus = 250 // in mA - Default current for testing
GLOBAL_fallTime = 3 // in µs - Default fall time
GLOBAL_riseTime = 3 // in µs - Default rise time
GLOBAL_halfBitTime = 417 // in µs - Default half bit time
GLOBAL_doubleHalfBitTime = 833 // in µs - Default double half bit time
GLOBAL_settlingTime = 15 // in ms - Default settling time, between any frame and a forward frame

GLOBAL_busPowered = **UserInput** (Is DUT a bus-powered device?, YesNo)
GLOBAL_internalBPS = **UserInput** (Has device a bus power supply unit integrated?, YesNo)
GLOBAL_MultiMasterControlDevice = **UserInput** (Is the device a multi master control device?, YesNo)
if (*GLOBAL_internalBPS* == Yes)
    if (*GLOBAL_busPowered* == Yes)
        GLOBAL_internalBPS = No
        UserInput (This DUT shall not be connected to any external power supply for all tests, *OK*)
        continueWith101tests = **UserInput** (Shall the 103 test being aborted in order to execute the 101 test procedures on the integrated bus power supply? YesNo)
        if (*continueWith101tests* == Yes)
            if (*GLOBAL_MultiMasterControlDevice* == Yes)
                UserInput (Please consider, that afterwards the DUT will not be factory new any more, OK)
                ResetDevice (false)

            endif

            halt 1 The 103 test was aborted in order to execute the 101 test procedures. In case of a multi master control device the DUT has been prepared for the 101 tests by means of disabling any application controller and input device instances.

        endif
    else
        GLOBAL_internalVoltage = **UserInput** (Enter the open circuit voltage of the internal bus power supply, value [V])
        GLOBAL_internalCurrent = **UserInput** (Enter the specified maximum current of the internal bus power supply, value [mA])
        GLOBAL_VbusHigh = *GLOBAL_internalVoltage*
        GLOBAL_Ibus = 250 - *GLOBAL_internalCurrent*
    endif
endif
UserInput (Set power supply such to have *GLOBAL_VbusHigh* V bus high and *GLOBAL_Ibus* mA and connect the DUT, *OK*)

if (*GLOBAL_MultiMasterControlDevice* == No)
    SingleMasterApplicationControllerPING ()
    halt 2 No further test procedures are applicable to a single master application controller.
else

    answer = QUERY DEVICE CAPABILITIES

    if (*answer* == NO)

        halt 2 DUT not found

    endif

endif

// Test factory default values - this part is optional
operatingMode = -1
factoryNewDevice = **UserInput** (Is DUT factory new ?, *YesNo*)
if (*factoryNewDevice* == Yes)
    (*operatingMode*) = **CheckFactoryDefault103** ()
else
    report 1 The check for factory default variables will be skipped for this DUT since the device is not factory new.
    operatingMode = QUERY OPERATING MODE

```
endif

// Ask user which operating mode to use for further testing
if (operatingMode != 0)
    keepOperatingMode = UserInput (Use current operating mode ('No' forces operating mode 0)?, YesNo)
    if (keepOperatingMode == Yes)
        keepOperatingMode = UserInput (Are all instructions defined in this standard implemented in all manufacturer specific modes and is the memory bank 0 the same for all manufacturer specific modes?, YesNo)
    endif
    if (keepOperatingMode == No)
        // Force DUT to standard mode
        DTR0 (0)
        SET OPERATING MODE
    endif
endif

// Abort testing if device has 64 logical units
GLOBAL_numberOfLogicalUnits = GetNumberOfLogicalUnits ()
if (GLOBAL_numberOfLogicalUnits == 64)
    warning 1 Bus unit has 64 logical units. Short address 63 is used for testing; therefore testing of this device is aborted.
else
    // Assign a short address to each logical unit, short address shall be equal to the index of the logical unit
    GLOBAL_numberShortAddresses = AddressPreamble ()
    if (GLOBAL_numberShortAddresses == 0)
        error 1 No units found.
    else if (GLOBAL_numberShortAddresses >= 64)
        error 2 Too many units found.
    else
        if (GLOBAL_numberShortAddresses != numberOfLogicalUnits)
            error 3 Number assigned short addresses differs from number of logical units available in the bus unit. Expected: numberOfLogicalUnits. Actual: GLOBAL_numberShortAddresses.
        endif

        // Get information per logical unit and store them as global parameters
        for (i = 0; i < 97; i++)
            GLOBAL_logicalUnit[j].extendedVersions[i] = -1
        endfor
        // query 103 version (same as for instance type 0)
        GLOBAL_logicalUnit[j].extendedVersions[0] = GetVersionNumber ()
        for (j = 0; j < GLOBAL_numberShortAddresses; j++)
            GLOBAL_logicalUnit[j].numberOfInstances = QUERY NUMBER OF INSTANCES, send to device (ShortAddress (j))

            for (i = 0; i < GLOBAL_logicalUnit[j].numberOfInstances; i++)
                // query instance type
                answer = QUERY INSTANCE TYPE, send to device ShortAddress (j), send to instance InstanceNumber (i)
                GLOBAL_logicalUnit[j].instanceTypes[i] = answer
                // query instance type version
                if (GLOBAL_logicalUnit[j].extendedVersions[answer] == -1)
                    DTR0 (answer)
                    GLOBAL_logicalUnit[j].extendedVersions[answer] = QUERY EXTENDED VERSION NUMBER, send to device ShortAddress (j)
                endif
                // query feature type
                for (k = 0; k < 65; k++)
                    GLOBAL_logicalUnit[j].instance[i].featureTypes[k] = -1
```

```
            endfor
            answer = QUERY FEATURE TYPE, send to device ShortAddress (j), send
            to instance InstanceNumber (i)
            if (answer == 254)
                // no feature implemented
            else if (answer >= 32 AND answer <= 96)
                GLOBAL_logicalUnit[j]. instance[i].featureTypes[0] = answer
                if (GLOBAL_logicalUnit[j].extendedVersions[answer] == -1)
                    DTR0 (answer)
                    GLOBAL_logicalUnit[j].extendedVersions[answer] = QUERY
                    EXTENDED VERSION NUMBER, send to device ShortAddress
                    (j)
                endif
            else
                k = 0
                do
                    answer = QUERY NEXT FEATURE TYPE, send to device
                    ShortAddress (j), send to instance InstanceNumber (i)
                    if (answer == 254 OR answer == No Answer)
                        if (k < 2)
                            error 4 QUERY NEXT FEATURE returned just one or
                            even no feature type at all, but QUERY FEATURE
                            returned to be more than one feature implemented.
                        endif
                        break
                    else
                        GLOBAL_logicalUnit[j]. instance[i].featureTypes[k] = answer
                        if (GLOBAL_logicalUnit[j].extendedVersions[answer] == -1)
                            DTR0 (answer)
                            GLOBAL_logicalUnit[j].extendedVersions[answer] =
                            QUERY EXTENDED VERSION NUMBER, send to
                            device ShortAddress (j)
                        endif
                        k++
                    endif
                while (k < 65)
            endif
        endfor
    endfor

    // Test factory default values of the variables which are different per logical unit
    if (factoryNewDevice == Yes)
        for (logicalUnitAddress = 0; logicalUnitAddress <
        GLOBAL_numberOfLogicalUnits; logicalUnitAddress++)
            CheckFactoryDefault103PerLogicalUnit (logicalUnitAddress;
            operatingMode)
        endfor
    endif
    // Define a variable to be used for further testing, to know which logical unit is under
    test
    GLOBAL_currentUnderTestLogicalUnit = 0

    // If multiple logical units are available in the bus unit, ask the user if tests shall be
    run for all logical units or for specific ones
    if (GLOBAL_numberOfLogicalUnits != 1)
        answer = UserInput (Shall the tests be run for all logical units?, YesNo)
        if (answer == Yes)
            for (i = 0; i < GLOBAL_numberOfLogicalUnits; i++)
                GLOBAL_logicalUnit[i].runTests = true
            endfor
        else
            for (i = 0; i < GLOBAL_numberOfLogicalUnits; i++)
```

answer = **UserInput** (Shall the tests be run for logical unit with index *i* ?, *YesNo*)
if (*answer* == Yes)
GLOBAL_logicalUnit[*i*].*runTests* = true
else
GLOBAL_logicalUnit[*i*].*runTests* = false
endif
endfor
endif
else
// Tests shall be run for the only one logical unit available in the bus unit
GLOBAL_logicalUnit [0] .*runTests* = true
endif
endif
endif

12.2.1.1 CheckFactoryDefault103

The test subsequence checks the 103 factory default variables. As the operating mode can differ per logical unit it is checked either in this subsequence or after each logical device has a short address assigned with the test subsequence CheckFactoryDefault103PerLogicalUnit() subsequence.

Subsequence shall be run for all logical units in parallel.

Test description:

(*operatingMode*) = **CheckFactoryDefault103** ()

// Verify operating mode of DUT
operatingMode = -1
answer = QUERY OPERATING MODE **accept** Violation
if (*answer* == Violation)
report 1 Multiple logical units with different default operating modes are available in one physical device.
answer = **UserInput** (Are all instructions defined in this standard implemented in all manufacturer specific modes and is the memory bank 0 the same for all manufacturer specific modes?, *YesNo*)
if (*answer* == No)
warning 1 Default operating mode for all logical devices cannot be verified. DUT is forced to operating mode 0x00.
DTR0 (0)
SET OPERATING MODE
operatingMode = 0
else
report 2 Default operating mode needs to be tested after each logical device has a short address assigned.
endif
else
if (*answer* == 0)
report 3 DUT is in the 0x00 operating mode.
operatingMode = *answer*
else
if (0x01 <= *answer* AND *answer* <= 0x7F)
error 1 DUT is in a reserved operating mode. Actual: *answer*. Expected: 0, [0x80,0xFF]. DUT is forced to operating mode 0x00.
DTR0 (0)
SET OPERATING MODE
operatingMode = 0
else

```
            report 4 DUT is in a manufacturer specific mode, operating mode answer.
            operatingMode = answer
        endif
    endif
endif

// Check default value of the 103 device variables
for (i = 0; i <= 11; i++)
    answer = query[i], accept Violation, Value
    if (answer == Violation)
        error 3 Multiple logical units returned different default values for variable[i].
    else
        if (answer != expectedAnswer[i])
            error 4 Wrong default value for variable[i]. Answer: answer. Expected:
            expectedAnswer[i].
        endif
    endif
    if (variable[i] == randomAddress)
        randomAddress = answer
    endif
endfor

// Verify initialiseState variable
answer = QUERY SHORT ADDRESS
if (answer != NO)
    error 5 At least one logical unit has an incorrect default initialisation state. Found:
    ENABLED or WITHDRAWN. Expected: DISABLED.
endif
answer = COMPARE
if (answer != NO)
    error 6 At least one logical unit has an incorrect default initialisation state. Found:
    ENABLED. Expected: DISABLED.
endif

// Verify shortAddress variable
INITIALISE (MASK)
answer = QUERY SHORT ADDRESS, accept Violation, Value
if (answer == Violation)
    error 7 Multiple logical units returned different default values for shortAddress.
else
    if (answer != 255)
        error 8 Wrong default value for shortAddress. Answer: answer. Expected: 255.
    endif
endif

// Verify searchAddress variable
if (randomAddress == 0xFF FF FF)
    answer = COMPARE
    if (answer == NO)
        error 9 Wrong default value for searchAddress since no answer was received from
        COMPARE command.
    endif
else
    warning 2 Default value for searchAddress variable not verified.
endif
TERMINATE

return (operatingMode)
```

Table 25 – Parameters for test sequence Check Factory Default 103

Test step i	query	variable	expectedAnswer
0	**GetRandomAddress** ()	randomAddress	0xFFFFFF
1	QUERY POWER CYCLE NOTIFICATION	powerCycleNotification	NO
2	QUERY DEVICE GROUPS 0-7	deviceGroups	0x00
3	QUERY DEVICE GROUPS 8-15	deviceGroups	0x00
4	QUERY DEVICE GROUPS 16-23	deviceGroups	0x00
5	QUERY DEVICE GROUPS 24-31	deviceGroups	0x00
6	QUERY CONTENT DTR0	DTR0	0x00
7	QUERY CONTENT DTR1	DTR1	0x00
8	QUERY CONTENT DTR2	DTR2	0x00
9	QUERY QUIESCENT MODE	quiescentMode	NO
10	QUERY APPLICATION CONTROLLER ERROR	applicationControllerError	NO
11	QUERY INPUT DEVICE ERROR	inputDeviceError	NO

12.2.1.2 AddressPreamble

The test subsequence clears all short addresses, then discovers the logical units available in a bus unit and gives each of them a short address equal to their index number. The subsequence returns the number of logical units found.

Subsequence shall be run for all logical units in parallel.

Test description:

numAssignedShortAddresses = **AddressPreamble** ()

searchCompleted = false

numAssignedShortAddresses = 0

assignedAddresses[63] = false

highestAssigned = -1

// Clear all short addresses, then detect all units and assign them short addresses

DTR0(MASK)

SET SHORT ADDRESS

INITIALISE (MASK)

RANDOMISE

wait 100 ms *// after stop condition of RANDOMISE command*

while (!*searchCompleted*)

// Check if any unit is still unaddressed

SetSearchAddress (0xFFFFFF)

answer = COMPARE

```
if (answer == NO)
    searchCompleted = true
endif
if (!searchCompleted)
    if (numAssignedShortAddresses < 63)
        searchAddress = 0xFFFFFF
        for (i = 23; i >= 0; i--)
            mask = (1 << i)
            searchAddress = searchAddress & (~mask)
            SetSearchAddress (searchAddress)
            answer = COMPARE
            if (answer == NO)
                // No unit in the requested random address range => revert mask
                searchAddress = searchAddress | mask
            else
                // At least one unit is there => keep mask
            endif
        endfor
        // Last bit reached => set valid searchAddress
        SetSearchAddress (searchAddress)
        answer = COMPARE
        if (answer == YES)
            // Valid single unit found => program short address, where short address is the
            index of the logical unit
            PROGRAM SHORT ADDRESS (ShortAddress (63))
            address = GetIndexOfLogicalUnit (63)
            if (address < 63)
                if (assignedAddresses[address] == true)
                    halt 1 Unexpected duplicate index number found. Actual: address
                else
                    PROGRAM SHORT ADDRESS (ShortAddress (address))
                    WITHDRAW
                    numAssignedShortAddresses++
                    assignedAddresses[address] = true
```

~~if (*address > highestAssigned*)~~

~~*highestAssigned = address*~~

~~endif~~

~~endif~~

~~else~~

~~halt 2 Unexpected high index number found in memorybank 0. Actual: *address* Expected: <63~~

~~endif~~

~~else~~

~~halt 3 No unit found at last search address~~

~~endif~~

~~endif~~

~~endif~~

~~INITIALISE (0111 1111b)~~

~~endwhile~~

~~TERMINATE~~

~~if (*numAssignedShortAddresses -1 != highestAssigned*)~~

~~for (*i = .0; i < highestAssigned; i++*)~~

~~if (*assignedAddresses[i]* == true)~~

~~report 1 Address assigned: *i*~~

~~else~~

~~report 2 Address not assigned: *i*~~

~~endif~~

~~endfor~~

~~halt 4 Unexpected gap in assigned short addresses detected.~~

~~endif~~

~~return *numAssignedShortAddresses*~~

~~12.2.1.3 CheckFactoryDefault103PerLogicalUnit~~

~~The test subsequence checks the default values of operating mode, for each logical unit, in case these were not tested before.~~

~~Subsequence shall be run for the logical unit with short address given by address variable.~~

~~**Test description:**~~

CheckFactoryDefault103PerLogicalUnit (*address*; *operatingMode*)

if (*operatingMode* == -1)

answer = QUERY OPERATING MODE, send to device **ShortAddress** (*address*)

if (*answer* == 0)

report 1 Logical unit *address* is in the 0x00 operating mode.

else

if (0x01 <= *answer* AND *answer* <= 0x7F)

error 1 Logical unit *address*: Logical unit is in a reserved operating mode. Actual: *answer*. Expected: 0, [0x80,0xFF].

report 2 In order to proceed with testing, logical unit *address* is set to operating mode 0x00.

DTR (0)

SET OPERATING MODE, send to device **ShortAddress** (*address*)

else

report 3 LogicalUnit *address*: Logical unit is in a manufacturer specific mode, operating mode *answer*.

endif

endif

endif

// Check default value of the 103 device variables
answer = QUERY DEVICE CAPABILITIES, send to device **ShortAddress** (*address*), **accept** Value
answer2 = QUERY DEVICE STATUS, send to device **ShortAddress** (*address*), **accept** Value
if (*answer* != 0000 00XXb)
error 2 Unused bytes not set to zero. Answer: *answer*. Expected: 0000 00XXb.
endif
if (*answer* == XXXX XXX1b)
GLOBAL_logicalUnit[*address*].*applicationController* == true
report 4 Application controller present.
else
GLOBAL_logicalUnit[*address*].*applicationController* == false
report 5 No Application controller present.
endif
if (*answer2* == 0XX0 X000b)
error 3 Wrong value for device status. Answer: *answer*. Expected: 0XX0 X000b.
endif
if (*answer* == XXXX XXX0b AND *answer2* == XXXX 1XXXb)
error 4 Non existing application controller is reported active. Answer: *answer2*. Expected: XXXX 0XXXb.
endif
if (*answer* == XXXX XXX1b AND *answer2* == XXXX 0XXXb)
error 5 Wrong default value for application controller status. Answer: *answer2*. Expected: XXXX 1XXXb (enabled).
endif
// Check default value of the 103 instance variables
for (*i* = 0; *i* < *GLOBAL_logicalUnit*[*address*].*numberOfInstances*; *i*++)

for (*j* = 0; *j* <= 6; *j*++)
answer = *query*[*j*], send to device **ShortAddress** (*address*), send to instance **InstanceNumber** (*i*), **accept** Value
if (*answer* != *expectedAnswer*[*j*])
error 6 Wrong default value at instance *i* for *variable*[*j*]. Answer: *answer*. Expected: *expectedAnswer*[*j*].
endif
endfor
answer = QUERY INSTANCE TYPE, send to device **ShortAddress** (*address*), send to instance **InstanceNumber** (*i*), **accept** Value
if (*answer* >= 32)
error 7 Wrong instance type number at instance *i*. Answer: *answer*. Expected: [0, 31].
else if (*answer* == 0)
eventFilter = **GetEventFilter** (send to device **ShortAddress** (*address*), send to instance **InstanceNumber** (*i*))
if (*eventFilter* != 0xFFFFFF)
error 8 Wrong event filter at instance *i*. Answer: *answer*. Expected: 0xFFFFFF.
endif
endif
answer = QUERY RESOLUTION, send to device **ShortAddress** (*address*), send to instance **InstanceNumber** (*i*), **accept** Value
if (*answer* == 0)
error 9 Wrong resolution at instance *i*. Answer: *answer*. Expected: [1, 255].
endif
endfor
return

Table 26 – Parameters for test sequence CheckFactoryDefault103PerLogicalUnit

Test step i	query	variable	expectedAnswer
0	QUERY PRIMARY INSTANCE GROUP	*"instanceGroup0"*	MASK
1	QUERY INSTANCE GROUP 1	*"instanceGroup1"*	MASK
2	QUERY INSTANCE GROUP 2	*"instanceGroup2"*	MASK
3	QUERY INSTANCE ENABLED	*"instanceActive"*	YES
4	QUERY EVENT SCHEME	*"eventScheme"*	0
5	QUERY EVENT PRIORITY	*"eventPriority"*	4
6	QUERY INSTANCE ERROR	*"instanceError"*	NO

12.2.1.4 SingleMasterApplicationControllerPING

For installation troubleshooting single master application controllers are meant to send out a cyclic PING message. This subsequence checks for the first PING being sent between 5 min to 10 min after power on and a cyclic repetition after 10 min with 10% tolerance.

Also the test checks on the single master transmitter bit timings on all of the repetitive PING messages.

Test sequence shall be run for all logical units in parallel.

Test description:

SingleMasterApplicationControllerPING ()

// Wait for first PING after power up occurs between 5 min and 10 min
PowerCycleAndWaitForDecoder (5)
StartBusRecording (*record*)

```
start_timer (timer)
do
    pingFound = FindFrame (record, PING)
    timestamp = get_timer (timer) // time in seconds
while (pingFound == 0 AND timestamp < 600)
if (pingFound == 0)
    error 1 Single master application controller did not sent PING command within 10 min after power up.
else
    if (timestamp < 300)
        error 2 Single master application controller sent PING before 5 min after power up. Actual: timestamp s. Expected: >= 300s.
    else
        report 1 Single master application controller sent PING timestamp s after power up.
    endif
    for (k = 0; k < 2; k++)
        // Wait for next PING occurs between 10 min +/-10%
        StartBusRecording (record)
        start_timer (timer)
        do
            pingFound = FindFrame (record, PING)
            timestamp = get_timer (timer) // time in seconds
        while (pingFound == 0 AND timestamp < 660)
        if (pingFound == 0)
            error 3 Single master application controller did not repeated PING command within 10 min +10%.
        else
            if (timestamp < 540)
                error 4 Single master application controller repeated PING before 10 min -10%. Actual: timestamp s. Expected: >= 540s.
            else
                report 2 Single master application controller repeated PING after timestamp s.
            endif
            for (i = 0; i < 6; i++)

                timeTe = Measure (Time of period[i] of PING frame at 8 V in µs)

                if (TeNo[i] == 1)

                    if (timeTe < 336 OR timeTe > 467)

                        error 5 Incorrect half bit timing at period[i] in PING. Actual: timeTe µs. Expected: 336 µs <= half bit time <= 467 µs.

                    else

                        report 3 Half bit timing at period[i] in PING. Actual: timeTe µs.

                    endif

                endif

                if (TeNo[i] == 2)

                    if (timeTe < 733 OR timeTe > 934)

                        error 6 Incorrect double half bit timing at period[i] in PING. Actual: timeTe µs. Expected: 733 µs <= double half bit time <= 934 µs.

                    else

                        report 4 Double half bit timing at period[i] in PING. Actual: timeTe µs.
```

endif

endif

endfor

for (*j* = 0; *j* < 2; *j*++)

fallTimeRelative = **Measure** (Time between 10% and 90% of the signal voltage swing for *edge*[*i*] falling edge in PING frame in µs)

riseTimeRelative = **Measure** (Time between 10% and 90% of the signal voltage swing for *edge*[*i*] rising edge in PING frame in µs)

if (*fallTimeRelative* < 3)

error 7 Wrong fall time at *GLOBAL_VbusHigh* V and 250 mA in *edge*[*i*] falling edge in PING frame. Actual: *fallTimeRelative* µs. Expected: >= 3 µs.

endif

if (*riseTimeRelative* < 3)

error 8 Wrong rise time at *GLOBAL_VbusHigh* V and 250 mA in *edge*[*i*] rising edge in PING frame. Actual: *riseTimeRelative* µs. Expected: >= 3 µs.

endif

endfor

for (*j* = 0; *j* < 2; *j*++)

voltage = **Measure** (Last high voltage of signal before *edge*[*i*] edge in V)

if (*voltage* < 12)

fallTimeAbsolute = **Measure** (Time between (*GLOBAL_VbusHigh* - 0,5) V and 4,5 V of *edge*[*i*] falling edge in backward frame in µs)

riseTimeAbsolute = **Measure** (Time between 4,5 V and (*GLOBAL_VbusHigh* - 0,5) V of *edge*[*i*] rising edge in backward frame in µs)

else

fallTimeAbsolute = **Measure** (Time between 11,5 V and 4,5 V of *edge*[*i*] falling edge in backward frame in µs)

riseTimeAbsolute = **Measure** (Time between 4,5 V and 11,5 V of *edge*[*i*] rising edge in backward frame in µs)

endif

if (*fallTimeAbsolute* > 25)

error 9 Wrong fall time at *GLOBAL_VbusHigh* V and 250 mA in *edge*[*i*] falling edge in PING frame. Actual: *fallTimeAbsolute* µs. Expected: <= 25 µs.

endif

if (*riseTimeAbsolute* > 25)

error 10 Wrong rise time at *GLOBAL_VbusHigh* V and 250 mA in *edge*[*i*] rising edge in PING frame. Actual: *riseTimeAbsolute* µs. Expected: <= 25 µs.

endif

endfor

endif

endfor

endif

StopBusRecording (*record*)

Table 27 – Parameters for test sequence Transmitter bit timing

Test step i	**period**	**TeNo**
0	first low period	1
1	first high period	1
2	second low period	1
3	second high period	2
4	last low period	1
5	last high period	1

Test step j	**edge**
0	first
1	last

12.3 Physical operational parameters

12.3.1 Polarity test

Test sequence checks if DUT is polarity insensitive with regard to bus interface connections.
Test sequence applies for DUTs without or with an inactive integrated bus power supply.

Test sequence shall be run for all logical units in parallel.

Test description:

if (*GLOBAL_internalBPS* == No)

answer = QUERY DEVICE CAPABILITIES, **accept** No Answer

if (*answer* != NO)

report 1 Communication possible at current polarity.

else

error 1 No communication at current polarity.

endif

Change (Swap data wires at DUT bus interface)

wait 100ms

answer = QUERY DEVICE CAPABILITIES, **accept** No Answer

if (*answer* != NO)

report 2 Communication possible at inverted polarity.

else

error 2 No communication at inverted polarity.

Change (Swap data wires at DUT bus interface)

wait 100ms

endif

else

report 3 Polarity test not executed due to presence of internal power supply.

endif

12.3.2 Maximum and minimum system voltage

Test sequence checks if the interface is able to withstand the maximum and minimum voltage ratings.

Test sequence shall be run for all logical units in parallel.

Test description:

if (*GLOBAL_Ibus* == 0)

report 1 Test not executed since device under test does not allow for additional external power supplies.

else

Apply (Current of *GLOBAL_Ibus* mA + 10 mA on bus interface)

for (*i* = 0; *i* < 2; *i*++)

Vbus = voltage[*i*]

Apply (Disconnect interface)

Apply (Voltage of *Vbus* V on bus interface)

Apply (Reconnect interface)

//The voltage may change

wait 1 min

Apply (Voltage of *GLOBAL_VbusHigh* V on bus interface)

DTR0 (13)

for (*j* = 0; *j* < 12; *j*++)

DTR0 (*value*[*j*])

answer = QUERY CONTENT DTR0

if (*answer* != *value*[*j*])

error 1 No successful operation after applying rating of *Vbus* V at bus interface for 1 min. Actual: *answer*. Expected: *value*[*j*].

endif

endfor

endfor

endif

Table 28 – Parameters for test sequence Maximum and minimum system voltage

Test step i	0	1
voltage [V]	22,5	-6,5

Test step j	0	1	2	3	4	5	6	7	8	9	10	11
value	0	1	2	4	8	16	32	64	85	128	170	255

12.3.3 Overvoltage protection test

Check over-voltage protection of the interface for the maximum rated external voltage of the system.

Test sequence shall be run for all logical units in parallel.

Test description:

overvoltageProtection = **UserInput** (Is overvoltage protection supported by DUT?, *YesNo*)

if (*overvoltageProtection* == Yes)

maximumVoltage = **UserInput** (Enter the maximum rated external voltage supported by DUT, *value* [V])

maximumFrequency = **UserInput** (Enter the maximum rated external frequency supported by DUT, *value* [Hz])

answer = QUERY DEVICE CAPABILITIES, **accept** No Answer

if (*answer* == NO)

error 1 No communication possible.

else

if (*GLOBAL_busPowered* == Yes)

Disconnect (Bus interface of DUT from the tester)

else

Switch_off (external power)

Disconnect (External power and bus interface of DUT from the tester)

endif

Apply (Overvoltage of *maximumVoltage* V with a frequency of *maximumFrequency* Hz on bus interface)

wait 1 min

Remove (Overvoltage from bus interface)

if (*GLOBAL_busPowered* == Yes)

Connect (Bus interface of DUT to the tester)

else

Connect (External power and bus interface of DUT to the tester)

Switch_on (external power)

endif

start_timer (*timer*)

do

answer = QUERY DEVICE CAPABILITIES, **accept** No Answer

timestamp = **get_timer** (*timer*)

if (*answer* != NO)

report 1 Overvoltage protection supported by DUT.

break

endif

while (*timestamp* <= 1 min)

if (*answer* == NO)

error 2 No communication after 1 min after applying *maximumVoltage* V / *maximumFrequency* Hz on bus interface.

endif

endif

else

report 2 Overvoltage protection is not supported by DUT.

endif

It is recommended that this test be repeated at the maximum and minimum operating temperature.

12.3.4 Current rating test

Test sequences checks current consumption while the bus is in idle state.

Test sequence shall be run for all logical units in parallel.

Test description:

if (*GLOBAL_internalBPS* == Yes)

report 1 Test is not applicable.

else

currentLimit = 2

if (*GLOBAL_busPowered*)

currentLimit = **UserInput** (Enter the current consumption shown on the label or stated in the literature, *value* [mA])

endif

Apply (Linear voltage change from 0 V to 22,5 V within 10 s to bus terminals as illustrated in Phase 1 of Figure 2)

QUERY CONTENT DTR0

// Voltage drop shall be applied directly after valid stop condition of forward frame to discharge internal capacitor of the receiver

Apply (Immediate voltage drop from 22,5 V to 0 V - see Figure 2)

Apply (Voltage of 0 V during 20 s - see Phase 2 in Figure 2)

Apply (Immediate voltage change from 0 V to 22,5 V at end of Phase 2 of Figure 2)

current1 = **Measure** (Maximum current consumption in mA at bus terminals during Phase 1)

current2 = **Measure** (Current consumption in mA at bus terminals during the immediate voltage change at end of Phase 2)

if (*current1 <= currentLimit*)

report 2 Maximum current consumption measured during Phase1 is *current1* mA. Expected: <= *currentLimit* mA.

else

error 1 Wrong maximum current consumption during Phase1. Actual: *current1* mA. Expected: <= *currentLimit* mA.

endif

if (*current2 <= currentLimit*)

report 3 Maximum current consumption measured at end of Phase 2 is *current2* mA. Expected: <= *currentLimit* mA.

else

error 2 Wrong maximum current consumption at end of Phase 2. Actual: *current2* mA. Expected: <= *currentLimit* mA.

endif

endif

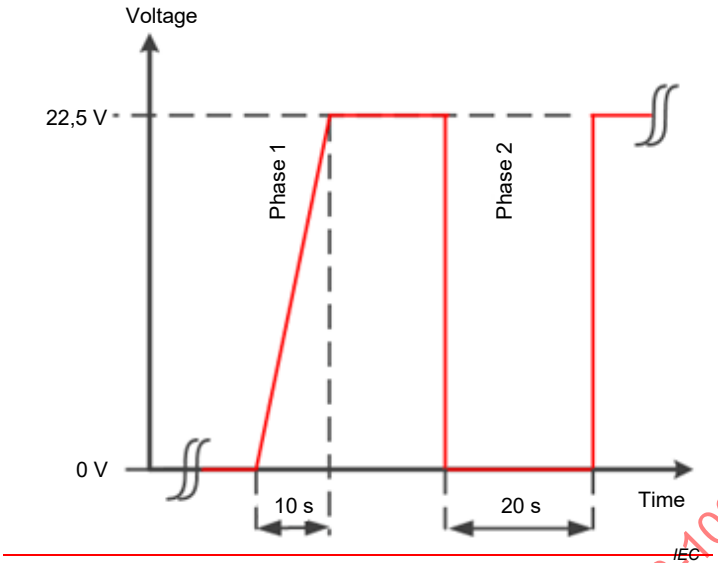


Figure 2 – Current rating test

It is recommended that this test be repeated at the maximum and minimum operating temperature.

12.3.5 Transmitter voltages

Test sequence checks

- if device responds for different voltage and current settings;
- low level voltage during active state of transmitter;
- high level voltage within backwards frame.

Test sequence shall be run for all logical units in parallel.

Test description:

// 250 mA if allowed, internal current in case of single internal BPS. Maximum current allowed is provided.

Apply (Current of *GLOBAL_Ibus* mA on bus interface)

// Test at minimum voltage and maximum current

if (*GLOBAL_internalBPS*)

Vbus = 12

Apply (Clamp bus voltage to *Vbus* V on bus interface)

else

Vbus = 10

Apply (Voltage of *Vbus* V on bus interface)

endif

CheckTxVoltages (*Vbus*; *GLOBAL_Ibus*)

if (*GLOBAL_Ibus* > 0)

// Test at 20,5 V and maximum current. GLOBAL_Ibus = 0 in case of single internal BPS

Apply (Voltage of 20,5 V on bus interface)

CheckTxVoltages (20,5; *GLOBAL_Ibus*)

endif

if (*GLOBAL_internalBPS*)

// Test at internal bus power supply voltage and maximum current if allowed

Apply (Voltage of *GLOBAL_internalVoltage* V on bus interface)

CheckTxVoltages (*GLOBAL_internalVoltage*; *GLOBAL_Ibus*)

// Test using only internal bus power supply

if (*GLOBAL_Ibus* > 0)

// Switch off test power supply, minimum current

Apply (Current of 0 mA on bus interface)

CheckTxVoltages (*GLOBAL_internalVoltage*; 0)

endif

else

// Test for current of 8 mA and different voltages

Apply (Current of 8 mA on bus interface)

Apply (Voltage of 10 V on bus interface)

CheckTxVoltages (10; 8)

Apply (Voltage of 20,5 V on bus interface)

CheckTxVoltages (20,5; 8)

endif

It is recommended that this test be repeated at the maximum and minimum operating temperature.

12.3.5.1 CheckTxVoltages

This subsequence checks the voltage of a signal sent by transmitter.

Test description:

CheckTxVoltages (*Vbus*; *Ibus*)

for (*i* = 0; *i* < 4; *i*++)

DTR0 (*value*[*i*])

current = Ibus + GLOBAL_internalCurrent

answer = QUERY CONTENT DTR0, **accept** No Answer

if (*answer* == NO)

error 1 No reply received at *Vbus* V and *current* mA for QUERY CONTENT DTR0.

else

// Once the bus voltage has crossed 4,5 V for a low level or 10 V for a high level, this level shall be crossed once in the opposite direction at the end of the high or low period

levelOkLow = **UserInput** (Are active low periods of answer within interval [-4,5 V; 4,5 V]?, *YesNo*)

levelOkHigh = **UserInput** (Is voltage high level of answer within interval [10 V; 22,5 V]?, *YesNo*)

if (*levelOkLow* == No)

error 2 Active low voltage period outside -4,5 V < Vlow < 4.5 V at *Vbus* V and *current* mA in backward frame *value*[*i*].

endif

if (*levelOkHigh* == No)

error 3 High level voltage period outside 10 V < Vhigh < 22,5 V at *Vbus* V and *current* mA in backward frame *value*[*i*].

endif

endif

endfor

return

Table 29 – Parameters for test sequence Transmitter voltages

Test step i	value
0	255
1	170
2	85
3	0

12.3.6 Transmitter rising and falling edges

Test sequence evaluates correctness of first and last falling and rising edges within a backward frame at different voltage and current settings.

Test sequence shall be run for all logical units in parallel.

Test description:

// Test at 12 V and maximum current

Apply (Current of *GLOBAL_Ibus* mA on bus interface)

Vbus = 12

if (*GLOBAL_internalBPS*)

Apply (Clamp bus voltage to *Vbus* V on bus interface)

else

Apply (Voltage of *Vbus* V on bus interface)

endif

CheckMaximumTxRiseFallTimes (12; *GLOBAL_Ibus*)

// Test at 10 V and 250 mA if possible

if (!*GLOBAL_internalBPS*)

Apply (Voltage of 10 V on bus interface)

CheckMaximumTxRiseFallTimes (10; *GLOBAL_Ibus*)

endif

// Test using maximum voltage if possible

if (*GLOBAL_Ibus* > 0)

Apply (Voltage of 20,5 V on bus interface)

CheckMaximumTxRiseFallTimes (20,5; *GLOBAL_Ibus*)

CheckMinimumTxRiseFallTimes (20,5; *GLOBAL_Ibus*)

endif

if (*GLOBAL_internalBPS*)

// Test at internal bus power supply voltage and maximum current

Apply (Voltage of *GLOBAL_internalVoltage* V on bus interface)

CheckMaximumTxRiseFallTimes (*GLOBAL_internalVoltage*; *GLOBAL_Ibus*])

// Test using only internal bus power supply if not covered by previous step

if (*GLOBAL_Ibus* > 0)

// Switch off test power supply

Apply (Current of 0 mA on bus interface)

CheckMaximumTxRiseFallTimes (*GLOBAL_internalVoltage*; 0)

else

CheckMinimumTxRiseFallTimes (*GLOBAL_internalVoltage*; 0)

endif

else

// Test for current of 8 mA and different voltages

Apply (Current of 8 mA on bus interface)

Apply (Voltage of 10 V on bus interface)

CheckMaximumTxRiseFallTimes (10; 8)

Apply (Voltage of 12 V on bus interface)

CheckMaximumTxRiseFallTimes (12; 8)

Apply (Voltage of 20,5 V on bus interface)

CheckMaximumTxRiseFallTimes (20,5; 8)

endif

Table 30 – Parameters for test sequence Transmitter rising and falling edges

Test step i	edge
0	first
1	last

It is recommended that this test be repeated at the maximum and minimum operating temperature.

12.3.6.1 CheckMinimumTxRiseFallTimes

This subsequence checks the minimum rise and fall times of a signal sent by transmitter.

Test description:

CheckMinimumTxRiseFallTimes (*Vbus*; *Ibus*)

for (*i* = 0; *i* < 2; *i*++)

DTR0 (95)

current = *Ibus* + *GLOBAL_internalCurrent*

answer = QUERY CONTENT DTR0, **accept** No Answer

if (*answer* == NO)

error 1 No reply received at *Vbus* V and *current* mA for QUERY CONTENT DTR0.

else

fallTimeRelative = **Measure** (Time between 10% and 90% of the signal voltage swing for *edge*[*i*] falling edge in backward frame in µs)

riseTimeRelative = **Measure** (Time between 10% and 90% of the signal voltage swing for *edge*[*i*] rising edge in backward frame in µs)

if (*fallTimeRelative* < 3)

error 2 Wrong fall time at *Vbus* V and *current* mA in *edge*[*i*] falling edge in backward frame. Actual: *fallTimeRelative* µs. Expected: >= 3 µs.

endif

if (*riseTimeRelative* < 3)

error 3 Wrong rise time at *Vbus* V and *current* mA in *edge*[*i*] rising edge in backward frame. Actual: *riseTimeRelative* µs. Expected: >= 3 µs.

endif

endif

endfor

return

12.3.6.2 CheckMaximumTxRiseFallTimes

This subsequence checks the maximum rise and fall times of a signal sent by transmitter.

Test description:

CheckMaximumTxRiseFallTimes (*Vbus*; *Ibus*)

for (*i* = 0; *i* < 2; *i*++)

DTR0 (95)

current = *Ibus* + *GLOBAL_internalCurrent*

answer = QUERY CONTENT DTR0, **accept** No Answer

if (*answer* == NO)

error 4 No reply received at *Vbus* V and *current* mA for QUERY CONTENT DTR0.

else

voltage = **Measure** (Last high voltage of signal before *edge*[*i*] edge in V)

if (*voltage* < 12)

fallTimeAbsolute = **Measure** (Time between (*Vbus* - 0,5) V and 4,5 V of *edge*[*i*] falling edge in backward frame in µs)

riseTimeAbsolute = **Measure** (Time between 4,5 V and (*Vbus* - 0,5) V of *edge*[*i*] rising edge in backward frame in µs)

else

fallTimeAbsolute = **Measure** (Time between 11,5 V and 4,5 V of *edge*[*i*] falling edge in backward frame in µs)

riseTimeAbsolute = **Measure** (Time between 4,5 V and 11,5 V of *edge*[*i*] rising edge in backward frame in µs)

endif

if (*fallTimeAbsolute* > 15)

error 5 Wrong fall time at *Vbus* V and *current* mA in *edge*[*i*] falling edge in backward frame. Actual: *fallTimeAbsolute* µs. Expected: <= 15 µs.

endif

if (*riseTimeAbsolute* > 15)

error 6 Wrong rise time at *Vbus* V and *current* mA in *edge[i]* rising edge in backward frame. Actual: *riseTimeAbsolute* µs. Expected: <= 15 µs.

endif

endif

endfor

return

12.3.7 Transmitter bit timing

This test sequence checks transmitter half bit time and double half bit timing being in limits.

Test sequence shall be run for all logical units in parallel.

Test description:

Apply (Current of *GLOBAL_Ibus* mA on bus interface)

if (*GLOBAL_internalBPS*)

Vbus = 12

Apply (Clamp bus voltage to *Vbus* V on bus interface)

else

Vbus = 10

Apply (Voltage of *Vbus* V on bus interface)

endif

// Test for maximum current and minimum voltage

CheckTxBitTiming (*Vbus*; *GLOBAL_Ibus*)

if (*GLOBAL_Ibus* > 0)

// Test for 20,5 V and maximum current if possible

Apply (Voltage of 20,5 V on bus interface)

CheckTxBitTiming (20,5; *GLOBAL_Ibus*)

endif

if (*GLOBAL_internalBPS*)

// Test at internal bus power supply voltage and maximum current if applicable

Apply (Voltage of *GLOBAL_internalVoltage* V on bus interface)

CheckTxBitTiming (*GLOBAL_internalVoltage*; *GLOBAL_Ibus*)

// Test using only internal bus power supply if not covered by previous step

if (*GLOBAL_Ibus* > 0)

// Switch off test power supply

Apply (Current of 0 mA on bus interface)

CheckTxBitTiming (*GLOBAL_internalVoltage*; 0)

endif

else

// Test for current equal to 8 mA and different voltages

Apply (Current of 8 mA on bus interface)

Apply (Voltage of 10 V on bus interface)

CheckTxBitTiming (10; 8)

Apply (Voltage of 20,5 V on bus interface)

CheckTxBitTiming (20,5; 8)

endif

It is recommended that this test be repeated at the maximum and minimum operating temperature.

12.3.7.1 CheckTxBitTiming

This subsequence checks the bit timings of a signal sent by transmitter.

Test description:

CheckTxBitTiming (*Vbus*; *Ibus*)

for (*i* = 0; *i* < 24; *i*++)

DTR0 (*value*[*i*])

current = Ibus + GLOBAL_internalCurrent

answer = QUERY CONTENT DTR0, **accept** No Answer

if (*answer* =! NO)

error 1 No reply received at *Vbus* V and *current* mA for QUERY CONTENT DTR0.

else

// Note: high level measurements apply only after the start bit and before the stop condition

timeTe = **Measure** (Time of *period*[*i*] of backward frame at 8 V in µs)

if (*TeNo*[*i*] == 1)

if (*timeTe* < 400 OR *timeTe* > 434)

error 2 Incorrect half bit timing at *period*[*i*] in *value*[*i*]. Actual: *timeTe* µs. Expected: 400 µs <= half bit time <= 434 µs.

else

report 1 Half bit timing at *period*[*i*] in *value*[*i*]. Actual: *timeTe* µs.

```
            endif
        endif
        if (TeNo[i] == 2)
            if (timeTe < 800 OR timeTe > 867)
                error 3 Incorrect double half bit timing at period[i] in value[i]. Actual: timeTe µs.
                Expected: 800 µs <= double half bit time <= 867 µs.
            else
                report 2 Double half bit timing at period[i] in value[i]. Actual: timeTe µs.
            endif
        endif
    endif
endfor
return
```

Table 31 – Parameters for test sequence Transmitter bit timing

Test step i	period	value	TeNo
0	first low period	0	1
1	first high period	0	2
2	second low period	0	1
3	second high period	0	1
4	last low period	0	1
5	last high period	0	1
6	first low period	85	1
7	first high period	85	1
8	second low period	85	1
9	second high period	85	2
10	last low period	85	2
11	last high period	85	2
12	first low period	170	1
13	first high period	170	1
14	second low period	170	1
15	second high period	170	2
16	last low period	170	1
17	last high period	170	2
18	first low period	255	1
19	first high period	255	1
20	second low period	255	1
21	second high period	255	1
22	last low period	255	1
23	last high period	255	1

12.3.8 Transmitter frame timing

Test sequence checks answer times being inside limits.

Test sequence shall be run for all logical units in parallel.

Test description:

minTime = 100

maxTime = 0

for (*i* = 0; *i* < 12; *i*++)

DTR0 (*value*[*i*])

for (*j* = 0; *j* < 10; *j*++)

answer = QUERY CONTENT DTR0

answerTime = **Measure** (Settling time between forward frame and backward frame of QUERY CONTENT DTR0 in ms)

// Test transmitter forward backward frame settling time according to Table 17 IEC62386-101 Ed2.0

if (*answerTime* < 5,5 OR *answerTime* > 10,5)

error 1 Incorrect answer time at test step (*i*,*j*) = (*i*,*j*). Actual: *answerTime* ms. Expected: 5,5 ms <= settling time <= 10,5 ms.

endif

if (*answerTime* < *minTime*)

minTime = *answerTime*

endif

if (*answerTime* > *maxTime*)

maxTime = *answerTime*

endif

endfor

endfor

report 1 Minimum measured settling time is *minTime* ms.

report 2 Maximum measured settling time is *maxTime* ms.

Table 32 – Parameters for test sequence Receiver frame timing

Test step i	0	1	2	3	4	5	6	7	8	9	10	11
value	0	1	2	4	8	16	32	64	85	128	170	255

It is recommended that this test be repeated at the maximum and minimum operating temperature.

12.3.9 Receiver start-up behavior

Test sequences tests if

- 40 ms bus power interruptions are ignored by control gear; tested four times;
- start-up behavior after a external power cycle is correct; tested four times. In case of no internal bus power supply four different times are used;
- start-up behavior after a bus power failure is correct.

Test sequence shall be run for all logical units in parallel.

Test description:

for (*i* = 0; *i* < 4; *i*++)

// 40 ms bus power interruption

wait 7 s

Apply (Voltage of 0 V on bus interface)

wait 40 ms

if (*GLOBAL_internalBPS*)

Apply (Clamp voltage to 12 V on bus interface)

else

Apply (Voltage of 10 V on bus interface)

endif

wait 2,4 ms *// stop condition*

answer = QUERY DEVICE CAPABILITIES, **accept** No Answer

if (*answer* == NO)

error 1 No communication after 40 ms bus power supply interruption at test step i = *i*.

endif

// External power cycle start-up

if (!*GLOBAL_internalBPS*)

Apply (Voltage of 0 V on bus interface)

endif

base = **PowerCycleAndWaitForBusPower** (60)

start_timer (*timer*)

if (*GLOBAL_internalBPS*)

Apply (Clamp voltage to 12 V on bus interface)

else

wait *delayTime*[*i*] ms

Apply (Voltage of 10 V on bus interface and a current supply of 8 mA)

endif

value = base + **get_timer** (*timer*) *// Get time in ms between power cycle and bus power supply restored*

if (*GLOBAL_busPowered*)

waitTime = 1200 // Maximum boot time

else

// Check for footnote e, Table 6, IEC 62386-101:2014

if (*value* < 350)

waitTime = 450 – value

else

waitTime = 100

endif

endif

wait *waitTime* ms *// Device should be ready now, all circumstances checked*

answer = QUERY DEVICE CAPABILITIES, **accept** No Answer

if (*answer* == NO)

error 2 No communication after *waitTime* ms after bus power supply available after external power cycle at test step i = *i*.

endif

// Bus power failure start-up

wait 1 200 ms

Apply (Voltage of 0 V on bus interface)

wait *busPowerDown*[*i*] ms

if (*GLOBAL_internalBPS*)

Apply (Clamp voltage to 12 V on bus interface)

else

Apply (Voltage of 10 V on bus interface)

endif

if (*GLOBAL_busPowered*)

waitTime = 1200

else

waitTime = 100

endif

wait *waitTime* ms

answer = QUERY DEVICE CAPABILITIES, **accept** No Answer

if (*answer* == NO)

error 3 No communication after *waitTime* ms after bus power down period of *busPowerDown*[*i*] ms at test step i = *i*.

endif

endfor

Table 33 – Parameters for test sequence Receiver start-up behavior

Test step i	0	1	2	3
delayTime [ms]	50	250	340	500
busPowerDown [ms]	100	500	1000	2000

It is recommended that this test be repeated at the maximum and minimum operating temperature.

12.3.10 Receiver threshold

The test sequence checks if the receiver threshold is in the expected range.

Test sequence shall be run for all logical units in parallel.

Test description:

for (*Vbus* = 9,5; *Vbus* <= 10; *Vbus* = *Vbus* + 0,5)

for (*i* = 0; *i* < 20; *i*++)

DTR0 (55)

Apply (Voltage of *Vbus* V on bus interface)

Apply (DTR0 (255) with low voltage of 6,5 V)

Apply (Voltage of *GLOBAL_VbusHigh* V on bus interface)

answer = QUERY CONTENT DTR0, **accept** No Answer

if (*answer* != 255)

if (*Vbus* < 10)

warning 1 DTR0 (255) not executed correctly for a bus high voltage of *Vbus* V and a bus low voltage of 6,5 V. Loop: *i* Actual: *answer*. Expected: 255.

else

error 1 DTR0 (255) not executed correctly for a bus high voltage of *Vbus* V and a bus low voltage of 6,5 V. Loop: *i* Actual: *answer*. Expected: 255.

endif

endif

endfor

endfor

It is recommended that this test be repeated at the maximum and minimum operating temperature.

12.3.11 Receiver bit timing

The test sequence checks receiver decoder bit timing compliance:

- for different half bit timings;
- for half bit and double half bit timing violations;
- for different high and low bus voltages.

Test sequence shall be run for all logical units in parallel.

Test description:

Vbus = GLOBAL_VbusHigh

for (*i* = 0; *i* < 4; *i*++)

// Command byte send with nominal timing; boundary combinations in 2nd byte.

for (*m* = 0; *m* < 5; *m*++)

lowTime = TeLowTime[*m*]

for (*n* = 0; *n* < 5; *n*++)

highTime = TeHighTime[*n*]

for (*j* = 0; *j* < 10; *j*++)

DTR0 (0)

// All other commands shall be executed with nominal timing

Apply (*command*[*i*] with bit timings given in Table)

answer = QUERY CONTENT DTR0

if (*answer* != *expectation*[*i*])

error 1 *command*[*i*] not correctly executed at half bit high time *highTime* µs half bit low time *lowTime* µs. Actual: *answer*. Expected: *expectation*[*i*].

endif

endfor

endfor

endfor

for (*i* = 4; *i* < 11; *i*++)

// step 4: no violation

// step 5: 750 µs half bit violation high bit 5

// step 6: 750 µs half bit violation low bit 2

// step 7: 1250 µs double half bit violation low bit 4 and 3

// step 8: 1250 µs double half bit violation low bit 8 and 7

// step 9: 1200 µs double half bit violation low bit 4 and 3

// step 10:1200 µs double half bit violation low bit 8 and 7

for (*l* = 0; *l* < 2; *l*++)

if (*GLOBAL_internalBPS*)

if (*l* == 1)

Vbus = 12

endif

else

Vbus = *busVoltage*[*l*]

endif

for (*j* = 0; *j* < 10; *j*++)

DTR0 (0)

Apply (*command*[*i*] with bit timings and voltages given in Table)

answer = QUERY CONTENT DTR0

if (*answer* != *expectation*[*i*])

error 2 *command*[*i*] not correctly processed for bus voltage of *Vbus* V at step *i*. Actual: *answer*. Expected: *expectation*[*i*].

endif

endfor

endfor

endfor

Table 34 – Parameters for test sequence Receiver bit timing

Test step m	0	1	2	3	4
TeLowTime [µs]	334	375	416	458	500

Test step n	0	1	2	3	4
TeHighTime [µs]	334	375	416	458	500

Test step l	0	1
busVoltage [V]	10	20,5

Test step i	0		1		2		3		4		5		6		7		8		9		10	
command	DTR0 (240)		DTR0 (15)		DTR0 (85)		DTR0 (255)		DTR0 (15)		DTR0 (15)		DTR0 (15)		DTR0 (15)		DTR0 (15)		DTR0 (15)		DTR0 (15)	
expectation	240		15		85		255		15		0		0		0		0		0		0	
voltage / time	bus voltage [V]	time [µs] at 8V	bus voltage [V]	time [µs] at 8V	bus voltage [V]	time [µs] at 8V	bus voltage [V]	time [µs] at 8V	bus voltage [V]	time [µs] at 8V	bus voltage [V]	time [µs] at 8V	bus voltage [V]	time [µs] at 8V	bus voltage [V]	time [µs] at 8V	bus voltage [V]	time [µs] at 8V	bus voltage [V]	time [µs] at 8V	bus voltage [V]	time [µs] at 8V
start bit	0	416	0	416	0	416	0	416	0	334	0	334	0	334	0	334	0	334	0	334	0	334
	VBus	416	VBus	416	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
bit 15	0	416	0	416	0	416	0	416	0	334	0	334	0	334	0	334	0	334	0	334	0	334
	VBus	416	VBus	416	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
bit 14	VBus	416	VBus	416	VBus	416	VBus	416	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
	0	416	0	416	0	416	0	416	0	500	0	500	0	500	0	500	0	500	0	500	0	500
bit 13	0	416	0	416	0	416	0	416	0	500	0	500	0	500	0	500	0	500	0	500	0	500
	VBus	416	VBus	416	VBus	416	VBus	416	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
bit 12	VBus	416	VBus	416	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
	0	416	0	416	0	416	0	416	0	334	0	334	0	334	0	334	0	334	0	334	0	334
bit 11	VBus	416	VBus	416	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
	0	416	0	416	0	416	0	416	0	334	0	334	0	334	0	334	0	334	0	334	0	334
bit 10	VBus	416	VBus	416	VBus	416	VBus	416	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
	0	416	0	416	0	416	0	416	0	334	0	334	0	334	0	334	0	334	0	334	0	334
bit 9	0	416	0	416	0	416	0	416	0	500	0	500	0	500	0	500	0	500	0	500	0	500
	VBus	416	VBus	416	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
bit 8	0	416	0	416	0	416	0	416	0	500	0	500	0	500	0	500	0	500	0	500	0	500
	VBus	416	VBus	416	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	600
bit 7	0	lowTime	VBus	highTime	VBus	highTime	0	highTime	VBus	334	VBus	334	VBus	334	VBus	334	VBus	750	VBus	334	VBus	600
	VBus	highTime	0	lowTime	0	lowTime	VBus	lowTime	0	500	0	500	0	500	0	500	0	500	0	500	0	500
bit 6	0	lowTime	VBus	highTime	0	highTime	0	highTime	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
	VBus	highTime	0	lowTime	VBus	lowTime	VBus	lowTime	0	500	0	500	0	334	0	500	0	500	0	500	0	500
bit 5	0	lowTime	VBus	highTime	VBus	highTime	0	highTime	VBus	334	VBus	750	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334

Test step i	0		1		2		3		4		5		6		7		8		9		10	
	VBus	highTime	0	lowTime	0	lowTime	VBus	lowTime	0	334	0	334	0	334	0	334	0	500	0	500	0	334
bit 4	0	lowTime	VBus	highTime	0	highTime	0	highTime	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
	VBus	highTime	0	lowTime	VBus	lowTime	VBus	lowTime	0	500	0	500	0	500	0	500	0	500	0	600	0	500
bit 3	VBus	highTime	0	lowTime	VBus	lowTime	0	lowTime	0	500	0	500	0	500	0	750	0	500	0	600	0	500
	0	lowTime	VBus	highTime	0	highTime	VBus	highTime	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
bit 2	VBus	highTime	0	lowTime	0	lowTime	0	lowTime	0	500	0	500	0	750	0	500	0	500	0	500	0	500
	0	lowTime	VBus	highTime	VBus	highTime	VBus	highTime	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
bit 1	VBus	highTime	0	lowTime	VBus	lowTime	0	lowTime	0	334	0	334	0	334	0	334	0	334	0	334	0	334
	0	lowTime	VBus	highTime	0	highTime	VBus	highTime	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
bit 0	VBus	highTime	0	lowTime	0	lowTime	0	lowTime	0	334	0	334	0	334	0	334	0	334	0	334	0	334
	0	lowTime	VBus	highTime	VBus	highTime	VBus	highTime	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500

It is recommended that this test be repeated at the maximum and minimum operating temperature.

12.3.12 Extended receiver bit timing

All phase lengths (half bits and double half bits) are set to the same value. One phase is set to special test value, resulting in a still valid waveform or causing a bit timing violation. The test is repeated for different idle bus voltages.

Test description:

```
waveForm[28]  = {417, 417, 417, 833, 833, 833, 417, 417, 417, 417,
                 833, 417, 417, 833, 417, 417, 417, 417, 417, 417,
                 833, 417, 417, 417, 417, 417, 417, 417}
                 // nominal timing for command DTR0(15)
for (i = 0; i <= 1; i++)
    for (j = 0; j <= 8; j++)
        for (k = 0; k <= 27; k ++) // k selects phase position to modify
            // assemble the test frame
            expected = expect[j]
            for (x = 0; x <= 27; x++)
                if (x == k)
                    // insert the modified phase length at the selected phase position
                    if (phase[x] == H)
                        // if a half bit starts in the middle of a logical bit it shall not be
                        extended as it would result in an valid double half bit and not in a
                        bit timing violation.
                        if (expect[j] == accept OR bitstart[x] == Y)
                            waveForm[x] = modHalf[j]
                        else
                            waveForm[x] = half[j]
                            expected = accept
                        endif
                    else
                        waveForm[x] = modDouble[j]
                    endif
                else
                    // use a valid phase length at all other phase positions
                    if (phase[x] == H)
                        waveForm[x] = half[j]
                    else
                        waveForm[x] = double[j]
                    endif
                endif
            endfor
            // send the test frame
            for (x = 0; x < 10; x++)
                DTR0(0)
                // All other commands shall be executed with nominal timing
                if (i == 0)

                    // minimum voltage
                    if (GLOBAL_internalBPS)

                        Apply (Clamp voltage to 12 V on bus interface)

                        busVoltage = 12

                    else

                        Apply (Voltage of 10 V on bus interface)

                        busVoltage = 10
```

endif

else

// maximum voltage
if (*GLOBAL_Ibus* == 0)

Apply (Voltage of *GLOBAL_VbusHigh* V on bus interface)

busVoltage = GLOBAL_VbusHigh

else

Apply (Voltage of 20,5 V on bus interface)

busVoltage = 20,5

endif

endif

Apply (*waveForm*[])
Apply (Voltage of *GLOBAL_VbusHigh* on bus interface)
answer = QUERY CONTENT DTR0
if (*expected* == accept)
if (*answer* != 15)
error 1 Test command DTR0 (15) not correctly executed with a half bit time *half*[*j*] µs and double half bit time *double*[*j*] µs and a modified half bit time *modHalf*[*j*] µs respectively double half bit time *modDouble*[*j*] µs at signal phase *k*. Bus Voltage: *busVoltage*. Actual: *answer*. Expected: 15.
endif
else
if (*answer* != 0)
error 2 Test command DTR0 (15) not ignored or executed falsely with a half bit time *half*[*j*] µs and double half bit time *double*[*j*] µs and a modified half bit time *modHalf*[*j*] µs respectively double half bit time *modDouble*[*j*] µs at signal phase *k*. Bus Voltage: *busVoltage*. Actual: *answer*. Expected: 0.
endif
endif
endfor
endfor
endfor
endfor

Table 35 – Parameters for test sequence extended receiver bit timing

Test step j	half	double	modHalf	modDouble	expect
0	417µs	833µs	500µs	1000µs	accept
1	417µs	833µs	334µs	667µs	accept
2	334µs	667µs	500µs	1000µs	accept
3	500µs	1000µs	334µs	667µs	accept
4	334µs	1000µs	500µs	667µs	accept
5	500µs	667µs	334µs	1000µs	accept
6	417µs	833µs	750µs	1200µs	ignore
7	334µs	667µs	750µs	1200µs	ignore

Test step j	half	double	modHalf	modDouble	expect
8	500µs	1000µs	750µs	1200µs	ignore

Phase x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
phase	H	H	H	D	D	D	H	H	H	H	D	H	H	D	H	H	H	H	H	H	D	H	H	H	H	H	H	H
bitstart	Y	N	Y	N	N	N	N	Y	N	Y	N	N	Y	N	N	Y	N	Y	N	Y	N	N	Y	N	Y	N	Y	N

It is recommended that this test be repeated at the maximum and minimum operating temperature.

12.3.13 Receiver forward frame violation

The test sequences checks if the receiver is able to recover after the reception of an non-standard forward frame.

Test sequence shall be run for all logical units in parallel.

Test description:

for (*i* = 0; *i* < 4; *i*++)

  for (*j* = 0; *j* < 10; *j*++)

    DTR0 (*value*[*i*])

    // waveform sent with directly after last rising edge of DTR0 command

    answer = waveform[*i*]

    if (*answer* != *value*[*i*])

      error 1 *text*[*i*]. Loop: *j*. Actual: *answer*. Expected: *value*[*i*].

    endif

  endfor

endfor

Table 36 – Parameters for test sequence Receiver frame violation and recovering after frame size violation

Test step i	value	waveform	text
0	7	2400 µs + 1101000011111110000b + 2400 µs + 1 11111111 11111110 00110110b	16 bit frame DTR0 (240) according Part 102 not ignored
1	10	2400 µs + 1010b + 2400 µs + 1 11111111 11111110 00110110b	Few bits (frame size violation) changed the content of DTR0
2	0	2400 µs + 1 11000001 00110000 00001111 0b + 2400 µs + 1 11111111 11111110 00110110b	25 bit frame containing DTR0 (15) in first 24 bits not ignored
3	0	2400 µs + 1 11000001 00110000 00001111 01010101b + 2400 µs + 1 11111111 11111110 00110110b	32 bit frame containing DTR0 (15) in first 24 bits not ignored

12.3.14 Receiver settling timing

Test sequence checks if

- forward-forward (FF-FF) frames with valid settling times are accepted;
- forward-forward (FF-FF) frames with invalid settling times are rejected;
- backward-forward (BF-FF) frames with valid settling times are accepted;
- backward-forward (BF-FF) frames with invalid settling times are rejected.

Test sequence shall be run for all logical units in parallel.

Test description:

// FF-FF frame tests

for (*i* = 0; *i* < 4; *i*++)

for (*j* = 0; *j* < 12; *j*++)

DTR0 (13)

DTR1 (13)

DTR0 (*value*[*j*])

wait *settlingTime*[*i*] ms *// settling time between FF FF*

DTR1 (*value*[*j*])

answer0 = QUERY CONTENT DTR0

answer1 = QUERY CONTENT DTR1

if (*i* < 2)

if (*answer0* != *value*[*j*])

error 1 Unexpected value for DTR0 for FF-FF settling time set to *settlingTime*[*i*] ms. Actual: *answer0*. Expected: *value*[*j*].

endif

if (*answer1* != *value*[*j*])

error 2 Unexpected value for DTR1 for FF-FF settling time set to *settlingTime*[*i*] ms. Actual: *answer1*. Expected: *value*[*j*].

endif

else

if (*answer0* != *13*)

error 3 Unexpected value for DTR0 for FF-FF settling time set to *settlingTime*[*i*] ms. Actual: *answer0*. Expected: 13.

endif

if (*answer1* != *13*)

error 4 Unexpected value for DTR1 for FF-FF settling time set to *settlingTime*[*i*] ms. Actual: *answer1*. Expected: 13.

~~endif~~

~~endif~~

~~endfor~~

~~endfor~~

~~*// BF-FF frame tests*~~

~~**for** (*i* = 0; *i* < 4; *i*++)~~

~~**for** (*j* = 0; *j* < 12; *j*++)~~

~~DTR1 (13)~~

~~*answer0* = QUERY CONTENT DTR0~~

~~**wait** *settlingTime*[*i*] ms *// settling time between BF-FF*~~

~~DTR1 (*value*[*j*])~~

~~*answer1* = QUERY CONTENT DTR1~~

~~**if** (*i* < 2)~~

~~**if** (*answer1* != *value*[*j*])~~

~~**error 5** DTR1 not accepted for BF-FF settling time set to *settlingTime*[*i*] ms. Actual: *answer1*. Expected: *value*[*j*].~~

~~**endif**~~

~~**else**~~

~~**if** (*answer1* != 13)~~

~~**error 6** DTR1 not ignored for BF-FF settling time set to *settlingTime*[*i*] ms. Actual: *answer1*. Expected: 13.~~

~~**endif**~~

~~**endif**~~

~~**endfor**~~

~~**endfor**~~

~~**Table 37 – Parameters for test sequence Receiver frame timing**~~

~~**Test step i**~~	~~**0**~~	~~**1**~~	~~**2**~~	~~**3**~~
~~**settlingTime [ms]**~~	~~3~~	~~2,4~~	~~1,4~~	~~1,2~~

~~**Test step j**~~	~~**0**~~	~~**1**~~	~~**2**~~	~~**3**~~	~~**4**~~	~~**5**~~	~~**6**~~	~~**7**~~	~~**8**~~	~~**9**~~	~~**10**~~	~~**11**~~
~~**value**~~	~~0~~	~~1~~	~~2~~	~~4~~	~~8~~	~~16~~	~~32~~	~~64~~	~~85~~	~~128~~	~~170~~	~~255~~

~~It is recommended that this test be repeated at the maximum and minimum operating temperature.~~

12.3.15 Receiver frame timing FF-FF send twice

Test sequence checks if

- send twice frames with maximum send twice settling time between the forward frames are correctly received;
- if send twice commands with one command in-between for minimum settling times are ignored.

Test sequence shall be run for all logical units in parallel.

Test description:

DTR2 (0)

for (*value* = 0; *value* < 16; *value*++)

// Correct reception for maximum send twice settling time

DTR1 (*value*)

ADD TO DEVICE GROUPS 0-15

DTR1 (*value* + 1)

ADD TO DEVICE GROUPS 0-15, send once

wait 94 ms *// settling time*

ADD TO DEVICE GROUPS 0-15, send once

answer = QUERY DEVICE GROUPS 0-7

if (*answer* != *value* + 1)

error 1 Send twice ADD TO DEVICE GROUPS 0-15 within 94 ms settling time between forward frames not accepted. Actual: *answer*. Expected: *value* + 1.

endif

// Ignore send twice for too long send twice settling time

DTR1 (*value*)

ADD TO DEVICE GROUPS 0-15

DTR1 (*value* + 1)

ADD TO DEVICE GROUPS 0-15, send once

wait 105 ms *// settling time*

ADD TO DEVICE GROUPS 0-15, send once

answer = QUERY DEVICE GROUPS 0-7

if (*answer* != *value*)

error 2 Send twice ADD TO DEVICE GROUPS 0-15 within 105 ms settling time between forward frames not ignored. Actual: *answer*. Expected: *value*.

endif

endfor

// Ignore send twice command for command in-between with minimum settling times

for (*value* = 0; *value* < 16; *value*++)

DTR1 (*value*)

DTR0 (255)

ADD TO DEVICE GROUPS 0-15

DTR1 (*value* + 1)

ADD TO DEVICE GROUPS 0-15, send once

wait 13,5 ms *// settling time*

DTR0 (*value*)

wait 13,5 ms *// settling time*

ADD TO DEVICE GROUPS 0-15, send once

answer = QUERY DEVICE GROUPS 0-7

if (*answer* != *value*)

error 3 Send twice not ignored for command in-between at test step = *value*. Actual: *answer*. Expected: *value*.

endif

answer = QUERY CONTENT DTR0

if (*answer* != *value*)

error 4 Command in-between DTR0 (*value*) not correctly executed at test step = *value*. Actual: *answer*. Expected: *value*.

endif

endfor

// Ignore send twice command for command in-between with minimum settling times, accept 2nd send twice

for (*value* = 0; *value* < 16; *value*++)

DTR1 (*value*)

DTR0 (255)

ADD TO DEVICE GROUPS 0-15

DTR1 (*value* + 1)

ADD TO DEVICE GROUPS 0-15, send once

wait 13,5 ms *// settling time*

DTR0 (*value*)

wait 13,5 ms *// settling time*

ADD TO DEVICE GROUPS 0-15, send once

wait 80 ms *// settling time*

ADD TO DEVICE GROUPS 0-15, send once

answer = QUERY DEVICE GROUPS 0-7

if (*answer* != *value* + 1)

error 5 Second send twice ignored for command in-between at test step = *value*. Actual: *answer*. Expected: *value* + 1.

endif

answer = QUERY CONTENT DTR0

if (*answer* != *value*)

error 6 Command in-between DTR0 (*value*) not correctly executed at test step = *value*. Actual: *answer*. Expected: *value*.

endif

endfor

It is recommended that this test be repeated at the maximum and minimum operating temperature.

12.3.16 Transmitter collision avoidance by priority

This test procedure requests a test frame to be sent with a priority 2 up to 5. At each time a frame with an according priority level one level higher than the requested test frame is send before to check for collision avoidance.

Test sequence shall be run for all logical units in parallel.

Test description:

ResetDevice (false)

pulseStartDelayStep = 1

for (*i* = 0; *i* < 2; *i*++)
  for (*priority* = 1; *priority* <= 5; *priority* ++)
    counter = 0
    for (*value* = 0; *value* < 10; *value*++)
      // send waveform and receive next frame
      SetupTestFrame (*frame*[*i*])
      next_frame = **SendWaveform** (SEND_TESTFRAME (*priority, 0*), settling time *idletime*, DTR0 (*value*))
      if (*next_frame* != *frame*[*i*])
        counter ++
      endif
      answer = QUERY CONTENT DTR0
      if (*answer* != *value*)
        counter ++
      endif
    endfor
    if (*counter* > 0)
      error 1 No successful operation at priority *priority* (*idletime*) settling time of *Idletime* ms. Actual: *counter* errors. Expected: 0.
    endif
  endfor
endfor

ResetDevice (false)

Table 38 – Parameters for test sequence transmitter collision avoidance by priority

Test step i	frame
0	0x000000
1	0xFFFFFF

Priority	Idletime
1	10,5 ms
2	14,7 ms
3	16,1 ms
4	17,7 ms
5	19,3 ms

12.3.17 Transmitter collision detection for truncated idle phase

In this test sequence the idle time of a certain bit in the DUT signal is truncated by start transmitting an active signal from the tester.

The truncating low pulse from the test system ends 483 (max half bit 433µs + 50µs) after the rising edge of the truncated idle phase (or after 916 for max double half bit 866µs + 50µs).

If through truncation the resulting idle phase is:

- longer than 400 (or 800 for double half bit), the DUT shall continue its transmission;
- smaller than 356 (or 723 for double half bit), the DUT shall abort the transmission. It might also destroy the frame;
- between 356..400 (or 723..800 for double half bit), the DUT might react either way.

The test system records the bus signal during the test and checks either for aborting and possibly destruction of the DUT test frame:

- the break condition exists (active period longer than 1.2ms);
- the reaction time (falling edge of truncated idle phase is closer than xy ms to the falling edge of the break condition);
- the recovery time (rising edge of the break condition to first falling edge of the re-send frame);
- the re-send frame (complete transmission of the requested test frame),

or continued transmission:

- complete transmission of the DUT test frame.

Test sequence shall be run for all logical units in parallel.

Test description:

ResetDevice (false)

```
pulseStartDelayStep = 1

for (i = 0; i < 2; i++)
    SetupTestFrame (frame[i])

    for (pulseStartDelay = initPulseStartDelay[i]; pulseStartDelay < maxPulsStartDelay[i];
    pulseStartDelay += pulseStartDelayStep)
        pulseLength = pulseStopReference[i] - pulseStartDelay

        SetPulseTrigger (first rising edge after first frame, pulseStartDelay, pulseLength)

        StartBusRecordingTrigger (after first frame, record)

        SEND TESTFRAME (2, 0)

        // wait here until two complete 24 bit frames are received: The first one is the "SEND
        TESTFRAME" of the test system and the second one is the test frame itself if no
        collision was detected or the repeated test frame after the first test frame was
        invalidated by a break condition
        wait until two 24 bit frames are received

        StopBusRecording (record)

        testFrameFound = FindFrame (record, frame[i])
        testFrameStart = FindFrameStart (reference point is first falling edge, record, frame[i])
        stopConditionStart = FindStopConditionStart (reference point is second rising edge,
        record)

        bitDuration = TimeDifference (first rising edge, second falling edge, record)
        breakConditionFound = FindBreakCondition (record)
        breakConditionStart = FindBreakConditionStart (reference point is first rising edge,
        record)

        if (testFrameFound == 0)
            error 1 Testframe not received
        else
            if (bitDuration > bitOkDuration[i])
                // here the test frame should be ok, the DUT can react in two ways:
                // a) continue with transmission
                // b) retreat the transmission

                // check if break condition which is not allowed in both cases
                if (breakConditionFound)
                    error 2 Break condition detected at bit duration: bitDuration µs
                endif

                if (stopConditionStart == 0)
                    // case b)
                    // DUT has retreated because stop condition occurred directly after
                    releasing the bus by the test system
                else
                    // case a)
                    // DUT has continued transmission because stop condition // occurred
                    later than the rising edge which releases the bus => check if test
                    frame was sent correctly from the beginning, which means that test
                    frame started at first falling edge otherwise the test frame is a
                    repeated one
                    if (testFrameStart != 0)
                        // test frame is a repeated one
                        error 3 Test frame is a repeated one and the first test frame was
                        not continued at bit duration: bitDuration µs
```

```
                endif
            endif

        else if (bitDuration < bitNotOkDuration[i])
            // here the test frame should be a repeated one, so that before the DUT has:
            // a) retreated or
            // b) destroyed the first test frame
            if (!breakConditionFound)
                // case a)
                // here the DUT retreated so that the stop condition is expected directly after the pulse otherwise the reaction is not correct
                if (stopConditionStart != 0)
                    error 4 DUT has not retreated the transmission nor send a break condition at bit duration: bitDuration µs
                endif
            else
                // case b)
                // here the DUT has send a break condition to destroy the first test frame => check if break condition timing is ok
                if (breakConditionStart > maximumBreakConditionDelay[i])
                    error 5 The break condition was set to late at bit duration: bitDuration µs
                endif
            endif
        else
            // only informative because of grey zone
            // case a) continue the transmission
            // case b) retreat the transmission
            // case c) destroy the transmission
            if (breakConditionFound)
                // case c)
                // here the DUT has send a break condition to destroy the first test frame => check if break condition timing is ok
                if (breakConditionStart > maximumBreakConditionDelay[i])
                    error 6 The break condition was set to late @ bit duration: bitDuration µs
                endif
            else
                if (stopConditionStart == 0)
                    // case b)
                    // DUT has retreated because stop condition occurred directly after releasing the bus by the test system
                else
                    // case a)
                    // DUT has continued transmission because stop condition occurred later than the rising edge which releases the bus => check if test frame was sent correctly from the beginning which means that test frame started at first falling edge otherwise the test frame is a repeated one
                    if (testFrameStart != 0)
                        // test frame is a repeated one
                        error Test frame is a repeated one and the first test frame was not continued at bit duration: bitDuration µs
                    endif
                endif
            endif
        endif
    endif
  endfor
endfor
```

ResetDevice (false)

Table 39 – Parameters for test sequence transmitter collision detection for truncated idle phase

Test step i	*frame*	*initPulseStartDelay*	*maxPulseStartDelay*	*pulseStopReference*	*bitOkDuration*	*bitNotOkDuration*	*Maximum Break Condition Delay*
0	0xFFFFFF	300	450	483	400	356	476+416
1	0x000000	670	850	916	800	723	943+416

12.3.18 Transmitter collision detection for extended active phase

In this test sequence the active time of a certain bit in the DUT signal is extended by start transmitting an active signal from the tester.

The low pulse from the test system to extend the pulse from the DUT starts 50µs after the falling edge of the active phase of the DUT is about to be extended.

If through elongation the resulting active phase is:

- smaller than 433 (or 866 for double half bit) the DUT shall continue its transmission;
- longer than 476 (or 943 for double half bit) the DUT shall abort the transmission. It might also destroy the frame;
- between 433..476 (or 866..943 for double half bit) the DUT might react either way.

The test system records the bus signal during the test and checks either for both aborting and destruction of the DUT test frame:

- the break condition exists (active period longer than 1,2ms);
- the reaction time (falling edge of truncated idle phase is closer than xy ms to the falling edge of the break condition);
- the recovery time (rising edge of the break condition to first falling edge of the re-send frame);
- the re-send frame (complete transmission of the requested test frame),

or continued transmission:

- completely continued transmission of the DUT test frame.

Test sequence shall be run for all logical units in parallel.

Test description:

ResetDevice (false)

pulseLengthStep = 1

for (*i* = 0; *i* < 2; *i*++)
    SetupTestFrame (*frame*[*i*])

for (*pulseLength = initPulseLength*[*i*]; *pulseLength < maxPulsLength*[*i*]; *pulseLength += pulseLengthStep*)
SetPulseTrigger (second falling edge after first frame, *pulseStartDelay*[*i*], *pulseLength*)

StartBusRecordingTrigger (after first frame, *record*)

SEND TESTFRAME (2, 0)

// wait here until two complete 24 bit frames are received: The first one is the "SEND TESTFRAME" of the test system and the second one is the test frame itself if no collision was detected or the repeated test frame after the first test frame was invalidated by a break condition
wait until two 24 bit frames are received

StopBusRecording (*record*)

testFrameFound = **FindFrame** (*record*, *frame*[*i*])
testFrameStart = **FindFrameStart** (reference point is second falling edge, *record*, *frame*[*i*])
stopConditionStart = **FindStopConditionStart** (reference point is second rising edge, *record*)

bitDuration = **TimeDifference** (second falling edge, second rising edge, *record*)
breakConditionFound = **FindBreakCondition** (*record*)
breakConditionStart = **FindBreakConditionStart** (reference point is second falling edge, *record*)

if (*testFrameFound* == 0)
error 1 Testframe not received
else
if (*bitDuration < bitOkDuration*[*i*])
// here the test frame should be ok, the DUT can react in two ways:
// a) continue with transmission
// b) retreat the transmission

// check if break condition which is not allowed in both cases
if (*breakConditionFound*)
error 2 Break condition detected at bit duration: *bitDuration* µs
endif

if (*stopConditionStart* == 0)
// case b)
// DUT has retreated because stop condition occurred directly after releasing the bus by the test system
else
// case a)
// DUT has continued transmission because stop condition occurred later than the rising edge which releases the bus => check if test frame was sent correctly from the beginning which means that test frame started at first falling edge otherwise the test frame is a repeated one
if (*testFrameStart* != 0)
// test frame is a repeated one
error 3 Test frame is a repeated one and the first test frame was not continued at bit duration: *bitDuration* µs
endif
endif
else if (*bitDuration > bitNotOkDuration*[*i*])
// here the test frame should be a repeated one, so that before the DUT has:

```
                    // a) retreated or
                    // b) destroyed the first test frame
                    if (!breakConditionFound)
                        // case a)
                        // here the DUT retreated so that the stop condition is expected
                        directly after the pulse otherwise the reaction is not correct
                        if (stopConditionStart != 0)
                            error 4 DUT has not retreated the transmission nor send a break
                            condition at bit duration: bitDuration µs
                        endif
                    else
                        // case b)
                        // here the DUT has send a break condition to destroy the first test
                        frame => check if break condition timing is ok
                        if (breakConditionStart > maximumBreakConditionDelay[i])
                            error 5 The break condition was set to late at bit duration:
                            bitDuration µs
                        endif
                    endif
                else
                    // only informative because of grey zone
                    // case a) continue the transmission
                    // case b) retreat the transmission
                    // case c) destroy the transmission
                    if (breakConditionFound)
                        // case c)
                        // here the DUT has send a break condition to destroy the first test
                        frame => check if break condition timing is ok
                        if (breakConditionStart > maximumBreakConditionDelay[i])
                            error 6 The break condition was set to late at bit duration:
                            bitDuration µs
                        endif
                    else
                        if (stopConditionStart == 0)
                            // case b)
                            // DUT has retreated because stop condition occurred directly
                            after releasing the bus by the test system
                        else
                            // case a)
                            // DUT has continued transmission because stop condition
                            occurred later than the rising edge which releases the bus =>
                            check if test frame was sent correctly from the beginning which
                            means that test frame started at first falling edge otherwise the
                            test frame is a repeated one
                            if (testFrameStart != 0)
                                // test frame is a repeated one
                                Error 7 Test frame is a repeated one and the first test frame
                                was not continued @ bit duration: bitDuration µs
                            endif
                        endif
                    endif
                endif
            endif
        endfor
    endfor

    ResetDevice (false)
```

Table 40 – Parameters for test sequence transmitter collision detection for extended active phase

Test step i	*initPulseLength*	*initPulseLength*	*maxPulseLength*	*pulseStartDelay*	*bitOkDuration*	*bitNotOkDuration*	*Maximum Break Condition Delay*
0	0xFFFFFF	330	480	50	433	476	476+416
1	0x7FFFFF	760	950	50	866	943	943+416

12.4 Device configuration instructions

12.4.1 RESET deviceGroups

In this test sequence the deviceGroups are set to non reset values. After sending a RESET command or after setting the deviceGroups back to their reset values, the deviceGroups shall be checked for their reset values. The resetState and the status of DUT are checked also after each change of the deviceGroups.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice ()

answer = QUERY DEVICE STATUS

if (*answer* != X1XX XXXXb)

error 1 Wrong answer at QUERY DEVICE STATUS after RESET command. Actual: *answer*. Expected: X1XX XXXXb.

endif

for (*i* = 0; *i* < 32; *i*++)

for (*j* = 0; *j* < 2; *j*++)

mask = (0x00000001 << *i*)

AddDeviceGroups (*mask*)

answer = QUERY RESET STATE

if (*answer* != NO)

error 2 Wrong answer at QUERY RESET STATE at test step (i,j) = (*i*,*j*). Actual: *answer*. Expected: NO.

endif

answer = QUERY STATUS

if (*answer* != X0XX XXXXb)

error 3 Wrong answer at QUERY STATUS at test step (i,j) = (*i*,*j*). Actual: *answer*. Expected: X0XX XXXXb.

endif

if (*j* == 0)

ResetDevice ()

else

RemoveDeviceGroups (*mask*)

endif

answer = **GetDeviceGroups** ()

if (*answer* != 0x00000000)

error 4 No RESET of deviceGroups at test step (i,j) = (*i,j*). Actual: *answer*. Expected: 0x00000000.

endif

answer = QUERY RESET STATE

if (*answer* != YES)

error 5 Wrong answer at QUERY RESET STATE at test step (i,j) = (*i,j*). Actual: *answer*. Expected: YES.

endif

answer = QUERY DEVICE STATUS

if (*answer* != X1XX XXXXb)

error 6 Wrong answer at QUERY STATUS at test step (i,j) = (*i,j*). Actual: *answer*. Expected: X1XX XXXXb.

endif

endfor

endfor

12.4.2 RESET quiescentMode

In this test sequence the quiescentMode is set to non-reset values. After sending a RESET command or after setting the quiescentMode back to its reset value, the quiescentMode shall be checked for its reset value. The resetState and the status of DUT are checked also after each change of the quiescentMode.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice ()

answer = QUERY DEVICE STATUS

if (*answer* != X1XX XXXXb)

error 1 Wrong answer at QUERY DEVICE STATUS after RESET command. Actual: *answer*. Expected: X1XX XXXXb.

```
endif

for (j = 0; j < 2; j++)

    START QUIESCENT MODE

    answer = QUERY RESET STATE

    if (answer != NO)

        error 2 Wrong answer at QUERY RESET STATE at test step (j) = (j). Actual: answer. Expected: NO.

    endif

    answer = QUERY STATUS

    if (answer != X0XX XXXXb)

        error 3 Wrong answer at QUERY STATUS at test step (j) = (j). Actual: answer. Expected: X0XX XXXXb.

    endif

    if (j == 0)

        ResetDevice ()

    else

        STOP QUIESCENT MODE

    endif

    answer = QUERY QUIESCENT MODE

    if (answer != NO)

        error 4 No RESET of quiescentMode at test step (j) = (j). Actual: answer. Expected: NO.

    endif

    answer = QUERY RESET STATE

    if (answer != YES)

        error 5 Wrong answer at QUERY RESET STATE at test step (j) = (j). Actual: answer. Expected: YES.

    endif

    answer = QUERY DEVICE STATUS

    if (answer != X1XX XXXXb)

        error 6 Wrong answer at QUERY STATUS at test step (j) = (j). Actual: answer. Expected: X1XX XXXXb.

    endif

endfor
```

12.4.3 RESET instance groups

In this test sequence the instance Groups are set to non-reset values. After sending a RESET command or after setting the instance Groups back to their reset values, the instance Groups shall be checked for their reset values. The resetState and the status of DUT are checked also after each change of the instance Groups.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice ()

answer = QUERY DEVICE STATUS

if (*answer* != X1XX XXXXb)

error 1 Wrong answer at QUERY DEVICE STATUS after RESET command. Actual: *answer*. Expected: X1XX XXXXb.

endif

numberOfInstances = **GetNumberOfInstances** ()
if (*numberOfInstances* == 0)
report 1 No instances available
else
for (*i* = 0; *i* < *numberOfInstances*; *i*++)
for (*c* = 0; *c* < 3; *c*++)

for (*j* = 0; *j* < 2; *j*++)

for (*k* = 0; *k* < 32; *k* = *k* + 8)

DTR0 (*k*)

setCommand[*c*]

answer = QUERY RESET STATE

if (*answer* != NO)

error 2 Wrong answer at QUERY RESET STATE at test step (i, j, k, c) = (*i, j, k, c*). Actual: *answer*. Expected: NO.

endif

answer = QUERY STATUS

if (*answer* != X0XX XXXXb)

error 3 Wrong answer at QUERY STATUS at test step (i, j, k, c) = (*i, j, k, c*). Actual: *answer*. Expected: X0XX XXXXb.

endif

if (*j* == 0)

ResetDevice ()

else

DTR0 (MASK)

setCommand[*c*]

endif

answer = queryCommand[*c*]

if (*answer* != MASK)

error 4 No RESET of instanceGroups at test step (i, j, k, c) = (*i, j, k, c*). Actual: *answer*. Expected: MASK.

endif

answer = QUERY RESET STATE

if (*answer* != YES)

error 5 Wrong answer at QUERY RESET STATE at test step (i, j, k, c) = (*i, j, k, c*). Actual: *answer*. Expected: YES.

endif

answer = QUERY DEVICE STATUS

if (*answer* != X1XX XXXXb)

error 6 Wrong answer at QUERY STATUS at test step (i, j, k, c) = (*i, j, k, c*). Actual: *answer*. Expected: X1XX XXXXb.

endif

endfor

endfor

endfor

endfor

endif

Table 41 – Parameters for test sequence RESET instance groups

Test step c	setCommand	queryCommand
0	SET PRIMARY INSTANCE GROUP	QUERY PRIMARY INSTANCE GROUP
1	SET INSTANCE GROUP 1	QUERY INSTANCE GROUP 1
2	SET INSTANCE GROUP 2	QUERY INSTANCE GROUP 2

12.4.4 RESET event filter

In this test sequence the instance event filter is set to non-reset values. After sending a RESET command or after setting the instance event filter back to its reset values, the instance event filter shall be checked for its reset values. The resetState and the status of DUT are checked also after each change of the instance event filter.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice ()

answer = QUERY DEVICE STATUS

if (*answer* != X1XX XXXXb)

error 1 Wrong answer at QUERY DEVICE STATUS after RESET command. Actual: *answer*. Expected: X1XX XXXXb.

endif

numberOfInstances = **GetNumberOfInstances** ()
if (*numberOfInstances* == 0)
report 1 No instances available
else
for (*i* = 0; *i* < *numberOfInstances*; *i*++)
answer = QUERY INSTANCE TYPE, send to instance **InstanceNumber** (*i*)
if (*answer* == 0)
for (*j* = 0; *j* < 2; *j*++)

for (*k* = 0; *k* < 24; *k*++)

setEventFilter (~(0x000001 << *k*))

answer = QUERY RESET STATE

if (*answer* != NO)

error 2 Wrong answer at QUERY RESET STATE at test step (i, j, k) = (*i, j, k*). Actual: *answer*. Expected: NO.

endif

answer = QUERY STATUS

if (*answer* != X0XX XXXXb)

error 3 Wrong answer at QUERY STATUS at test step (i, j, k) = (*i, j, k*). Actual: *answer*. Expected: X0XX XXXXb.

endif

if (*j* == 0)

ResetDevice ()

else

setEventFilter (0xFFFFFF)

endif

answer = **getEventFilter** ()

if (*answer* != 0xFFFFFF)

error 4 No RESET of eventFilter at test step (i, j, k) = (*i, j, k*). Actual: *answer*. Expected: MASK.

endif

answer = QUERY RESET STATE

if (*answer* != YES)

error 5 Wrong answer at QUERY RESET STATE at test step (i, j, k) = (*i, j, k*). Actual: *answer*. Expected: YES.

endif

answer = QUERY DEVICE STATUS

if (*answer* != X1XX XXXXb)

error 6 Wrong answer at QUERY STATUS at test step (i, j, k) = (*i, j, k*). Actual: *answer*. Expected: X1XX XXXXb.

endif

endfor

endfor

endif

endfor

endif

12.4.5 RESET event scheme

In this test sequence the instance event scheme is set to non-reset values. After sending a RESET command or after setting the instance event scheme back to its reset values, the instance event scheme shall be checked for its reset values. The resetState and the status of DUT are checked also after each change of the instance event scheme.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice ()

answer = QUERY DEVICE STATUS

if (*answer* != X1XX XXXXb)

error 1 Wrong answer at QUERY DEVICE STATUS after RESET command. Actual: *answer*. Expected: X1XX XXXXb.

endif

numberOfInstances = **GetNumberOfInstances** ()
if (*numberOfInstances* == 0)
report 1 No instances available
else
for (*i* = 0; *i* < *numberOfInstances*; *i*++)
for (*j* = 0; *j* < 2; *j*++)

for (*k* = 1; *k* < 5; *k*++)

DTR0 (*k*)

SET EVENT SCHEME

answer = QUERY RESET STATE

if (*answer* != NO)

error 2 Wrong answer at QUERY RESET STATE at test step (i, j, k) = (*i, j, k*). Actual: *answer*. Expected: NO.

endif

answer = QUERY STATUS

if (*answer* != X0XX XXXXb)

error 3 Wrong answer at QUERY STATUS at test step (i, j, k) = (*i, j, k*). Actual: *answer*. Expected: X0XX XXXXb.

endif

if (*j* == 0)

ResetDevice ()

else

DTR0 (0)

SET EVENT SCHEME

endif

answer = QUERY EVENT SCHEME

if (*answer* != MASK)

error 4 No RESET of quiescentMode at test step (i, j, k) = (*i, j, k*). Actual: *answer*. Expected: MASK.

endif

answer = QUERY RESET STATE

if (*answer* != YES)

error 5 Wrong answer at QUERY RESET STATE at test step (i, j, k) = (*i, j, k*). Actual: *answer*. Expected: YES.

endif

answer = QUERY DEVICE STATUS

if (*answer* != X1XX XXXXb)

error 6 Wrong answer at QUERY STATUS at test step (i, j, k) = (*i, j, k*). Actual: *answer*. Expected: X1XX XXXXb.

endif

endfor

endfor

endfor

endif

12.4.6 RESET: timeout / command in-between

The command RESET shall be executed only if it is received twice.

This test sequence checks the behaviour of DUT in the following conditions:

- one single RESET command is sent instead of two identical commands;

- RESET command is sent twice with a settling time of 105 ms which is longer than the defined settling time;
- RESET command is sent with a frame in-between, frame which consists of few bits, but not a command;
- RESET command is sent with a command in-between, command which is broadcast sent;
- RESET command is sent with a command in-between, command which is sent to a certain group address;
- RESET command is sent with a command in-between, command which is sent to a certain short address;
- RESET command is sent with a command in-between, and RESET command is sent again once.

In the first six cases, the RESET command should not be executed. In the last case RESET command should be executed. Where given, the command in-between should be accepted.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice ()

// Test send RESET once

START QUIESCENT MODE

RESET, send once

wait 400 ms *//100 ms for "virtual" send-twice + 300 ms needed for RESET*

answer = QUERY QUIESCENT MODE

if (*answer* != YES)

    error 1 RESET sent once executed. Actual: *answer*. Expected: YES.

endif

// Test send RESET with timeout

START QUIESCENT MODE

RESET, send once

wait 105 ms *// settling time*

RESET, send once

wait 300 ms

answer = QUERY QUIESCENT MODE

if (*answer* != YES)

    error 2 RESET with timeout executed. Actual: *answer*. Expected: YES.

endif

// Test send RESET with a frame in-between

START QUIESCENT MODE

// The following 3 steps must be sent within 75 ms, counted from the last rise bit of first "RESET, send once" command until first fall bit of second "RESET, send once" command

RESET, send once

idle 13 ms + 110010 + idle 13 ms *// settling time: idle 13 ms followed by a frame, followed by 13 ms*

RESET, send once

wait 300 ms

answer = QUERY QUIESCENT MODE

if (*answer* != YES)

error 3 RESET with few bits in between executed. Actual: *answer*. Expected: YES.

endif

// Test send RESET with broadcast command in-between

START QUIESCENT MODE

DTR0 (0)

// The following 3 steps must be sent within 75 ms, counted from the last rise bit of first "RESET, send once" command until first fall bit of second "RESET, send once" command

RESET, send once

DTR0 (1)

RESET, send once

wait 300 ms

answer = QUERY QUIESCENT MODE

if (*answer* != YES)

error 4 RESET with command in-between executed. Actual: *answer*. Expected: YES.

endif

answer = QUERY CONTENT DTR0

if (*answer* != 1)

error 5 Command in-between RESET not executed. Actual: *answer*. Expected: 1.

endif

// Test send RESET with group command in-between

START QUIESCENT MODE

// The following 3 steps must be sent within 75 ms, counted from the last rise bit of first "RESET, send once" command until first fall bit of second "RESET, send once" command

RESET, send once

answerCmdInBetween = QUERY DEVICE CAPABILITIES, send to device **GroupAddress** (0), **accept** No Answer

RESET, send once

wait 300 ms

answer = QUERY QUIESCENT MODE

if (*answer* != YES)

error 6 RESET with command in-between executed. Actual: *answer*. Expected: YES.

endif

if (*answerCmdInBetween* != NO)

error 7 Command in-between RESET executed. Actual: *answerCmdInBetween*. Expected: NO.

endif

// Test send RESET with short command in-between

START QUIESCENT MODE

// The following 3 steps must be sent within 75 ms, counted from the last rise bit of first "RESET, send once" command until first fall bit of second "RESET, send once" command

RESET, broadcast, send once

answerCmdInBetween = QUERY DEVICE CAPABILITIES, send to device **ShortAddress** (63)

RESET, broadcast, send once

wait 300 ms

answer = QUERY QUIESCENT MODE

if (*answer* != YES)

error 8 RESET with command in-between executed. Actual: *answer*. Expected: YES.

endif

if (*answerCmdInBetween* != NO)

error 9 Command in-between RESET executed. Actual: *answerCmdInBetween*. Expected: NO.

endif

// Test send RESET with broadcast command in-between, and again a new RESET command sent once

START QUIESCENT MODE

DTR0 (0)

// The following 4 steps must be sent within 75 ms, counted from the last rise bit of first "RESET, send once" command until first fall bit of third "RESET, send once" command

RESET, send once

DTR0 (1)

RESET, send once

RESET, send once

wait 300 ms

answer = QUERY QUIESCENT MODE

if (*answer* != NO)

error 10 RESET command not executed. Actual: *answer*. Expected: NO.

endif

answer = QUERY CONTENT DTR0

if (*answer* != 1)

error 11 Command in-between RESET not executed. Actual: *answer*. Expected: 1.

endif

12.4.7 Send twice timeout (device)

Any configuration instruction shall be executed only if it is received twice.

In this test sequence, all user programmable parameters of the DUT are attempted to be changed using configuration instructions sent as follows:

- one single command is sent instead of two identical commands, therefore the parameter should not change;
- command is sent twice with a settling time of 105 ms which is longer than the defined settling time, therefore the parameter should not change;
- command is sent three times with a settling time between the first two commands of 105 ms, and a settling time between the next two commands of 50 ms. Therefore, the first command should be ignored, and the next two should be interpreted as a send-twice command. As a consequence the parameter should change.

Test sequence shall be run for each selected logical unit.

Test description:

oldAddress = GLOBAL_currentUnderTestLogicalUnit

for (*i* = 0; *i* < 13; *i*++)

for (*j* = 0; *j* < 3; *j*++)

ResetDevice ()

DTR0 (0)

DTR1 (1)

command1[*i*]

if (*j* == 0) *// Test send command once*

command2[*i*], send once

else if (*j* == 1) *// Test send command with timeout*

command2[*i*], send once

wait 105 ms *// settling time*

command2[*i*], send once

else *// Test send command with timeout followed by a new command*

command2[i], send once

wait 105 ms *// settling time*

command2[i], send once

wait 50 ms *// settling time*

command2[i], send once

endif

if (*j* < 2)

answer = query[i]

if (*answer* != *value1*[*i*])

error 1 Wrong setting of *errorText[i]* at test step (i,j) = (*i*,*j*). Actual: *answer*. Expected: *value*[*i*].

endif

else

if (*i* < 12)

answer = query[i]

else

answer = query[i], send to device **ShortAddress** (63), **accept** No Answer

endif

if (*answer* != *value2*[*i*])

error 2 Wrong setting of *errorText[i]* at test step (i,j) = (*i*,*j*). Actual: *answer*. Expected: *value*[*i*].

endif

endif

if (*i* == 10 OR *i* == 11)

TERMINATE

endif

endfor

endfor

SetShortAddress (63; *oldAddress*)

Table 42 – Parameters for test sequence Send twice timeout (device)

Test step i	command1	command2	query	value1	value2	errorText
0	-	ADD TO DEVICE GROUPS 0-15	QUERY DEVICE GROUPS 8-15	0x00	0x01	gearGroups0-15
1	ADD TO DEVICE GROUPS 0-15	REMOVE FROM DEVICE GROUPS 0-15	QUERY DEVICE GROUPS 8-15	0x01	0x00	gearGroups0-15
2	-	ADD TO DEVICE GROUPS 16-31	QUERY DEVICE GROUPS 24-31	0x00	0x01	gearGroups16-31
3	ADD TO DEVICE GROUPS 16-31	REMOVE FROM DEVICE GROUPS 16-31	QUERY DEVICE GROUPS 24-31	0x01	0x00	gearGroups16-31
4	DISABLE APPLICATION CONTROLLER	ENABLE APPLICATION CONTROLLER	QUERY APPLICATION CONTROLLER ENABLED	NO	YES	applicationActive
5	ENABLE APPLICATION CONTROLLER	DISABLE APPLICATION CONTROLLER	QUERY APPLICATION CONTROLLER ENABLED	YES	NO	applicationActive
6	START QUIESCENT MODE	STOP QUIESCENT MODE	QUERY QUIESCENT MODE	YES	NO	quiescentMode
7	STOP QUIESCENT MODE	START QUIESCENT MODE	QUERY APPLICATION CONTROLLER ENABLED	NO	YES	quiescentMode
8	DISABLE POWER CYCLE NOTIFICATION	ENABLE POWER CYCLE NOTIFICATION	QUERY POWER CYCLE NOTIFICATION	NO	YES	powerCycleNotification
9	ENABLE POWER CYCLE NOTIFICATION	DISABLE POWER CYCLE NOTIFICATION	QUERY POWER CYCLE NOTIFICATION	YES	NO	powerCycleNotification
10	-	INITIALISE (*oldAddress*)	QUERY SHORT ADDRESS	NO	*oldAddress*	initialisationState
11	INITIALISE (*oldAddress*)	RANDOMISE	**GetRandomAddress** ()	0xFF FF FF	! 0xFF FF FF	randomAddress
12	DTR0 (63)	SET SHORT ADDRESS	QUERY DEVICE CAPABILITIES	NO	! NO	shortAddress

12.4.8 Send twice timeout (instance)

Any configuration instruction shall be executed only if it is received twice.

In this test sequence, all user programmable parameters of the DUT are attempted to be changed using configuration instructions sent as follows:

- one single command is sent instead of two identical commands, therefore the parameter should not change;
- command is sent twice with a settling time of 105 ms which is longer than the defined settling time, therefore the parameter should not change;
- command is sent three times with a settling time between the first two commands of 105 ms, and a settling time between the next two commands of 50 ms. Therefore, the first command should be ignored, and the next two should be interpreted as a send twice command. As a consequence the parameter should change.

Test sequence shall be run for each selected logical unit.

Test description:

oldAddress = GLOBAL_currentUnderTestLogicalUnit

numberOfInstances = **GetNumberOfInstances** ()
if (*numberOfInstances* == 0)
    report 1 No instances available
else

    for (*k* = 0; *k* < *numberOfInstances*; *k*++)

        for (*j* = 0; *j* < 8; *j*++)

            for (*j* = 0; *j* < 3; *j*++)

                ResetDevice ()

                DTR0 (4)

                command1[*i*] , send to instance **InstanceNumber** (*k*)

                if (*j* == 0) *// Test send command once*

                    command2[*i*], send once, send to instance **InstanceNumber** (*k*)

                else if (*j* == 1) *// Test send command with timeout*

                    command2[*i*], send once, send to instance **InstanceNumber** (*k*)

                    wait 105 ms *// settling time*

                    command2[*i*], send once, send to instance **InstanceNumber** (*k*)

                else *// Test send command with timeout followed by a new command*

                    command2[*i*], send once, send to instance **InstanceNumber** (*k*)

                    wait 105 ms *// settling time*

                    command2[*i*], send once, send to instance **InstanceNumber** (*k*)

                    wait 50 ms *// settling time*

                    command2[*i*], send once, send to instance **InstanceNumber** (*k*)

                endif

~~*answer = query[i]* , send to instance **InstanceNumber** (*k*)~~

~~**if** (*j* < 2)~~

~~**if** (*answer* != *value1*[*i*])~~

~~**error 1** Wrong setting of *errorText*[*i*] at test step (i,j,k) = (*i,j,k*). Actual: *answer*. Expected: *value*[*i*].~~

~~**endif**~~

~~**else**~~

~~**if** (*answer* != *value2*[*i*])~~

~~**error 2** Wrong setting of *errorText*[*i*] at test step (i,j,k) = (*i,j,k*). Actual: *answer*. Expected: *value*[*i*].~~

~~**endif**~~

~~**endif**~~

~~**endfor**~~

~~**endfor**~~

~~**endfor**~~

~~**endif**~~

Table 43 – Parameters for test sequence Send twice timeout (instance)

Test step i	command1	command2	query	value1	value2	errorText
0	SET EVENT PRIORITY, DTR0 (2)	SET EVENT PRIORITY	QUERY EVENT PRIORITY	4	2	eventPriority
1	DTR0 (2), SET EVENT PRIORITY, DTR0 (4)	SET EVENT PRIORITY	QUERY EVENT PRIORITY	2	4	eventPriority
2	DISABLE INSTANCE	ENABLE INSTANCE	QUERY INSTANCE ENABLED	NO	YES	instanceActive
3	ENABLE INSTANCE	DISABLE INSTANCE	QUERY INSTANCE ENABLED	YES	NO	instanceActive
4	-	SET PRIMARY INSTANCE GROUP	QUERY PRIMARY INSTANCE GROUP	MASK	4	instanceGroup0
5	-	SET INSTANCE GROUP 1	QUERY INSTANCE GROUP 1	MASK	4	instanceGroup1
6	-	SET INSTANCE GROUP 2	QUERY INSTANCE GROUP 2	MASK	4	instanceGroup2
7	-	SET EVENT SCHEME	QUERY EVENT SCHEME	0	4	eventScheme

12.4.9 Commands in-between (device)

Any device configuration instruction shall be executed only if it is received twice.

In this test sequence, all user programmable device parameters of the DUT are attempted to be changed using device configuration instructions send as follows:

- device configuration instruction is sent with a frame in-between, frame which consists of few bits, but not a command;
- device configuration instruction is sent with a command in-between, command which is broadcast sent;
- device configuration instruction is sent with a command in-between, command which is sent to a certain group address;
- device configuration instruction is sent with a command in-between, command which is sent to a certain short address;
- device configuration instruction is sent with a command in-between, and configuration instruction is sent again once.

In the first four cases, the device configuration command should not be executed. In the last case the device configuration instruction should be accepted. Where given, the command in-between should be accepted.

Test sequence shall be run for each selected logical unit.

Test description:

oldAddress = GLOBAL_currentUnderTestLogicalUnit

for (*i* = 0; *i* < 12; *i*++)

if (*i* == 4 AND *GLOBAL_logicalUnit*[*oldAddress*].*applicationController* == false))

i = i + 2 // Skip the next two test steps if no application controller is present

endif

// Test the rejection of the configuration instruction

for (*j* = 0; *j* < 5; *j*++)

ResetDevice ()

DTR0 (0)

DTR1 (1)

command1[*i*]

// The following steps must be sent within 75 ms, counted from the last rise bit of first "command2[i], send once" command until first fall bit of the last "command2[i], send once" command

command2[*i*], send once

if (*j* == 0)

idle 13 ms + 110010 + idle 13 ms *// Idle 13 ms followed by a frame, followed by 13 ms - Test send command with a frame in-between*

else if (*j* == 1)

answerCmdInBetween = QUERY DEVICE CAPABILITIES, send to device **Broadcast** (), **accept** No Answer

else if (*j* == 2)

answerCmdInBetween = QUERY DEVICE CAPABILITIES, send to device **GroupAddress** (0), **accept** No Answer

else if (*j* == 3)

answerCmdInBetween = QUERY DEVICE CAPABILITIES, send to device **ShortAddress** (*oldAddress*), **accept** No Answer

else

answerCmdInBetween = QUERY DEVICE CAPABILITIES send to device **ShortAddress** (63), **accept** No Answer

endif

command2[*i*], send once

answer = query[*i*]

if (*answer* != *value1*[*i*])

error 1 Wrong setting of *errorText*[*i*] at step (i,j) = (*i,j*). Actual: *answer*. Expected: *value1*[*i*].

endif

if (*j* == 1 OR *j* == 3)

if (*answerCmdInBetween* == NO)

error 2 Command in-between not executed at step (i,j) = (*i,j*). Actual: *answerCmdInBetween*. Expected: not NO.

endif

else

if (*answerCmdInBetween* != NO)

error 3 Command in-between executed at step (i,j) = (*i,j*). Actual: *answerCmdInBetween*. Expected: NO.

endif

endif

endfor

// Test the acceptance of the configuration instruction

ResetDevice ()

DTR0 (0)

DTR1 (1)

DTR2 (0)

command1[*i*]

// The following four steps must be sent within 75 ms, counted from the last rise bit of first "command2[i], send once" command until first fall bit of the last "command2[i], send once" command

command2[i], send once

DTR2 (1)

command2[i], send once

command2[i], send once

wait 100 ms

answer = query[i]

if (*answer* != *value2[i]*)

error 4 Wrong setting of *errorText[i]* at step i = *i*. Actual: *answer*. Expected: *value2[i]*.

endif

answer = QUERY CONTENT DTR2

if (*answer* != 1)

error 5 Wrong value of DTR2 at step i = *i*. Actual: *answer*. Expected: 1.

endif

if (*i* == 10 OR *i* == 11)

TERMINATE

endif

endfor

Table 44 – Parameters for test sequence Commands in-between (device)

Test step i	command1	command2	query	value1	value2	errorText
0		ADD TO DEVICE GROUPS 0-15	QUERY DEVICE GROUPS 8-15	0x00	0x01	gearGroups0-15
1	ADD TO DEVICE GROUPS 0-15	REMOVE FROM DEVICE GROUPS 0-15	QUERY DEVICE GROUPS 8-15	0x01	0x00	gearGroups0-15
2		ADD TO DEVICE GROUPS 16-31	QUERY DEVICE GROUPS 24-31	0x00	0x01	gearGroups16-31
3	ADD TO DEVICE GROUPS 16-31	REMOVE FROM DEVICE GROUPS 16-31	QUERY DEVICE GROUPS 24-31	0x01	0x00	gearGroups16-31
4	DISABLE APPLICATION CONTROLLER	ENABLE APPLICATION CONTROLLER	QUERY APPLICATION CONTROLLER ENABLED	NO	YES	applicationActive
5	ENABLE APPLICATION CONTROLLER	DISABLE APPLICATION CONTROLLER	QUERY APPLICATION CONTROLLER ENABLED	YES	NO	applicationActive
6	START QUIESCENT MODE	STOP QUIESCENT MODE	QUERY QUIESCENT MODE	YES	NO	quiescentMode
7	STOP QUIESCENT MODE	START QUIESCENT MODE	QUERY APPLICATION CONTROLLER ENABLED	NO	YES	quiescentMode
8	DISABLE POWER CYCLE NOTIFICATION	ENABLE POWER CYCLE NOTIFICATION	QUERY POWER CYCLE NOTIFICATION	NO	YES	powerCycleNotification
9	ENABLE POWER CYCLE NOTIFICATION	DISABLE POWER CYCLE NOTIFICATION	QUERY POWER CYCLE NOTIFICATION	YES	NO	minLevel
10	-	INITIALISE (*oldAddress*)	QUERY SHORT ADDRESS	NO	*oldAddress*	initialisationState
11	INITIALISE (*oldAddress*)	RANDOMISE	**GetRandomAddress** ()	0xFF FF FF	! 0xFF FF FF	randomAddress

12.4.10 Commands in-between (instance)

Any instance configuration instruction shall be executed only if it is received twice.

In this test sequence, all user programmable instance parameters of the DUT are attempted to be changed using instance configuration instructions sent as follows:

- instance configuration instruction is sent with a frame in between, frame which consists of few bits, but not a command;
- instance configuration instruction is sent with a command in-between, command which is broadcast sent;
- instance configuration instruction is sent with a command in-between, command which is sent to a certain group address;
- instance configuration instruction is sent with a command in-between, command which is sent to a certain short address;
- instance configuration instruction is sent with a command in-between, and configuration instruction is sent again once.

In the first four cases, the instance configuration command should not be executed. In the last case the instance configuration instruction should be accepted. Where given, the command in-between should be accepted.

Test sequence shall be run for each selected logical unit.

Test description:

oldAddress = GLOBAL_currentUnderTestLogicalUnit

numberOfInstances = **GetNumberOfInstances** ()
if (*numberOfInstances* == 0)
    report 1 No instances available
else

    for (*k* = 0; *k* < *numberOfInstances*; *k*++)

        for (*i* = 0; *i* < 8; *i*++)

            // Test the rejection of the configuration instruction

            for (*j* = 0; *j* < 5; *j*++)

                ResetDevice ()

                DTR0 (4)

                command1[*i*], send to instance **InstanceNumber** (*k*)

                // The following steps must be sent within 75 ms, counted from the last rise bit of first "command2[i], send once" command until first fall bit of the last "command2[i], send once" command

                command2[*i*], send-once, send to instance **InstanceNumber** (*k*)

                if (*j* == 0)

                    idle 13 ms + 110010 + idle 13 ms *// Idle 13 ms followed by a frame, followed by 13 ms - Test send command with a frame in between*

                else if (*j* == 1)

answerCmdInBetween = QUERY DEVICE CAPABILITIES, send to device **Broadcast** (), **accept** No Answer

else if (*j* == 2)

answerCmdInBetween = QUERY DEVICE CAPABILITIES, send to device **GroupAddress** (0), **accept** No Answer

else if (*j* == 3)

answerCmdInBetween = QUERY DEVICE CAPABILITIES, send to device **ShortAddress** (*oldAddress*), **accept** No Answer

else

answerCmdInBetween = QUERY DEVICE CAPABILITIES send to device **ShortAddress** (63), **accept** No Answer

endif

command2[*i*], send once, send to instance **InstanceNumber** (*k*)

answer = *query*[*i*] , send to instance **InstanceNumber** (*k*)

if (*answer* != *value1*[*i*])

error 1 Wrong setting of *errorText*[*i*] at step (i,j,k) = (*i,j,k*). Actual: *answer*. Expected: *value1*[*i*].

endif

if (*j* == 1 OR *j* == 3)

if (*answerCmdInBetween* == NO)

error 2 Command in-between not executed. Actual: *answerCmdInBetween*. Expected: not NO.

endif

else

if (*answerCmdInBetween* != NO)

error 3 Command in-between executed. Actual: *answerCmdInBetween*. Expected: NO.

endif

endif

endfor

// Test the acceptance of the configuration instruction

ResetDevice ()

DTR2 (0)

command1[*i*] , send to instance **InstanceNumber** (*k*)

// The following four steps must be sent within 75 ms, counted from the last rise bit of first "command2[i], send once" command until first fall bit of the last "command2[i], send once" command

command2[i], send once, send to instance **InstanceNumber** (*k*)

DTR2 (1)

command2[i], send once, send to instance **InstanceNumber** (*k*)

command2[i], send once, send to instance **InstanceNumber** (*k*)

wait 100 ms

answer = *query[i]* , send to instance **InstanceNumber** (*k*)

if (*answer* != *value2[i]*)

error 4 Wrong setting of *errorText[i]* at step (i,k) = *i,k*. Actual: *answer*. Expected: *value2[i]*.

endif

answer = QUERY CONTENT DTR2

if (*answer* != 1)

error 5 Wrong value of DTR2 at step (i,k) = *i,k*. Actual: *answer*. Expected: 1.

endif

endfor

endfor

endif

Table 45 – Parameters for test sequence Commands in-between

Test step i	command1	command2	query	value1	value2	errorText
0	SET EVENT PRIORITY, DTR0 (2)	SET EVENT PRIORITY	QUERY EVENT PRIORITY	4	2	eventPriority
1	DTR0 (2), SET EVENT PRIORITY, DTR0 (4)	SET EVENT PRIORITY	QUERY EVENT PRIORITY	2	4	eventPriority
2	DISABLE INSTANCE	ENABLE INSTANCE	QUERY INSTANCE ENABLED	NO	YES	instanceActive
3	ENABLE INSTANCE	DISABLE INSTANCE	QUERY INSTANCE ENABLED	YES	NO	instanceActive
4	-	SET PRIMARY INSTANCE GROUP	QUERY PRIMARY INSTANCE GROUP	MASK	4	instanceGroup0
5	-	SET INSTANCE GROUP 1	QUERY INSTANCE GROUP 1	MASK	4	instanceGroup1
6	-	SET INSTANCE GROUP 2	QUERY INSTANCE GROUP 2	MASK	4	instanceGroup2
7	-	SET EVENT SCHEME	QUERY EVENT SCHEME	0	4	eventScheme

12.4.11 SAVE PERSISTENT VARIABLES

Manufacturer is recommended to check the correct behaviour of the SAVE PERSISTENT VARIABLES command.

12.4.12 SET OPERATING MODE

The test sequence checks if reserved modes (0x01 to 0x7F) are reserved by trying to set the DUT in one of those modes, and checks if there are any manufacturer specific modes (0x80 to 0xFF) and if so, the DUT should keep reacting according to specification.

Test sequence shall be run for each selected logical unit.

Test description:

initialOperatingMode = QUERY OPERATING MODE

for (*i* = 1; *i* < 0x80; *i*++)

DTR0 (0)

SET OPERATING MODE

DTR0 (*i*)

SET OPERATING MODE

answer = QUERY OPERATING MODE

if (*answer* != 0)

error 1 SET OPERATING MODE executed with DTR0 set to the reserved operating mode *i*. Actual: *answer*. Expected: 0.

endif

answer = QUERY MANUFACTURER SPECIFIC MODE

if (*answer* != NO)

error 2 QUERY MANUFACTURER SPECIFIC MODE answered when operating mode set to mode 0, and DTR0 set to *i*. Actual: *answer*. Expected: NO.

endif

endfor

for (*i* = 0x80; *i* <= 0xFF; *i*++)

DTR0 (0)

SET OPERATING MODE

DTR0 (*i*)

SET OPERATING MODE

answer = QUERY OPERATING MODE

if (*answer* == 0)

answer = QUERY MANUFACTURER SPECIFIC MODE

if (*answer* != NO)

error 3 QUERY MANUFACTURER SPECIFIC MODE answered when operating mode set to mode 0, and DTR0 set to *i*. Actual: *answer*. Expected: NO.

endif

else if (*answer == i*)

report 1 Manufacturer specific mode *i* implemented in DUT.

answer = QUERY MANUFACTURER SPECIFIC MODE

if (*answer* != YES)

error 4 QUERY MANUFACTURER SPECIFIC MODE did not answer when DUT is in the manufacturer specific mode *i*. Actual: *answer*. Expected: YES.

endif

else *//different value than 0 and i received*

error 5 Operating mode set to a completely different operating mode than allowed. Actual: *answer*. Expected: 0 or *i*.

endif

endfor

DTR0 (*initialOperatingMode*)

SET OPERATING MODE

12.4.13 Device Disable/Enable Application Controller

This sequence first tests if an application controller is present (Bit 0 of QUERY DEVICE CAPABILITIES).

If an application controller is present, it will be disabled (send twice DISABLE APPLICATION CONTROLLER) and its resulting status will be checked (QUERY DEVICE STATUS Bit 3 and QUERY APPLICATION CONTROL ENABLED).

Afterwards the application controller will be enabled (send twice ENABLE APPLICATION CONTROLLER) and its status will be checked, also after power cycle (combined bus and mains).

In the next step the application controller will be disabled and its status will be checked, also after power cycle (combined bus and mains).

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice (true)

appControllerExist = **HasApplicationController**()

if (!*appControllerExist)*
report 1 No application controller is present

// enable application controller and check state
ENABLE APPLICATION CONTROLLER

```
    enabled = QUERY APPLICATION CONTROLLER ENABLED
    if (enabled)
        error 1 Application controller is enabled but device features implies no one is present
    endif
    status = QUERY DEVICE STATUS
    if (status == XXXX 1XXXb)
        error 2 Application controller is enabled in device status but device features implies no one is present
    endif

    // perform power cycle and check state
    PowerCycleAndWaitForDecoder(5)
    enabled = QUERY APPLICATION CONTROLLER ENABLED
    if (enabled)
        error 3 Application controller is enabled but device features implies no one is present
    endif
    status = QUERY DEVICE STATUS
    if (status == XXXX 1XXXb)
        error 4 Application controller is enabled in device status but device features implies no one is present
    endif
else
    // disable application controller and check state
    DISABLE APPLICATION CONTROLLER
    enabled = QUERY APPLICATION CONTROLLER ENABLED
    if (enabled)
        error 5 Application controller is enabled
    endif
    status = QUERY DEVICE STATUS
    if (status == XXXX 1XXXb)
        error 6 Application controller is enabled in device status
    endif

    // perform power cycle and check state
    PowerCycleAndWaitForDecoder(5)
    enabled = QUERY APPLICATION CONTROLLER ENABLED
    if (enabled)
        error 7 Application controller is enabled after power up
    endif
    status = QUERY DEVICE STATUS
    if (status == XXXX 1XXXb)
        error 8 Application controller is enabled in device status after power up
    endif

    // enable application controller and check state
    ENABLE APPLICATION CONTROLLER
    enabled = QUERY APPLICATION CONTROLLER ENABLED
    if (!enabled)
        error 9 Application controller is not enabled
    endif
    status = QUERY DEVICE STATUS
    if (status == XXXX 0XXXb)
        error 10 Application controller is not enabled in device status
    endif

    // perform power cycle and check state
    PowerCycleAndWaitForDecoder(5)
    enabled = QUERY APPLICATION CONTROLLER ENABLED
    if (!enabled)
        error 11 Application controller is not enabled after power up
```

endif
status = QUERY DEVICE STATUS
if (*status* == XXXX 0XXXb)
error 12 Application controller is not enabled in device status after power up
endif
endif

ResetDevice (false)

12.4.14 Multi Master Control Device PING

For installation troubleshooting only single master application controllers are meant to send out a cyclic PING message. This test sequence checks for multi master control devices if no PING is sent within 10 min +10% after power up.

Test sequence shall be run for all logical units in parallel.

Test description:

PowerCycleAndWaitForDecoder (5)
// Wait for no PING occurs within 10 min +10%
StartBusRecording (*record*)
start_timer (*timer*)
do
pingFound = **FindFrame** (*record*, PING)
timestamp = **get_timer** (*timer*) *// time in seconds*
while (*pingFound* == 0 AND *timestamp* < 660)
if (*pingFound* > 0)
error 1 PING command sent by a multi master control device.
endif
StopBusRecording (*record*)

12.4.15 Quiescent Mode

In order to check the quiescent mode the sequence first checks for an application controller and instances of an input device. If found, the application controller (send twice ENABLE APPLICATION CONTROLLER) and/or instances (send twice ENABLE INSTANCE) will be enabled.

The sequence checks for the quiescent mode being disabled after a power cycle. The quiescent mode is set (send twice START QUIESCENT MODE) and queried through QUERY QUIESCENT MODE and QUERY DEVICE STATUS Bit 1. After 5 min the quiescent mode will be retriggered (send twice START QUIESCENT MODE) and checked, if the timeout of 15 min (10% tolerance) is met based on the time of the retrigger command. While this time the bus is checked for no forward frame other than the query commands of the test system.

After this the Quiescent mode will be set and checked immediately again. After this the quiescent mode will be terminated (send twice STOP QUIESCENT MODE) and checked.

Test sequence shall be run for all logical units in parallel.

Test description:

// reset device and enable it
ResetDevice(true)
// Perform power cycle
PerformPowerCycle()
// start quiescent mode and bus recording
START QUIESCENT MODE

StartBusRecording(*record***)**
// check if quiescent mode is enabled
quiescentModeEnabled **= QUERY QUIESCENT MODE**
if (*quiescentModeEnabled* **!= YES)**
    error 1 Quiescent mode not enabled.
endif
*status***= QUERY DEVICE STATUS**
if (*status* **!= XXXX XX1Xb)**
    error 2 Quiescent mode not enabled in device status
endif
// wait 5 min and retrigger quiescent mode
wait 5min
START QUIESCENT MODE
// wait 15 min – 10%, query quiescent mode and stop bus recording
wait 13,5 min
quiescentModeEnabled **= QUERY QUIESCENT MODE**
if (*quiescentModeEnabled* **!= YES)**
    error 3 Quiescent mode not enabled.
endif
*status***= QUERY DEVICE STATUS**
if (*status* **!= XXXX XX1Xb)**
    error 4 Quiescent mode not enabled in device status.
endif
StopBusRecording(*record***)**
// wait until 15 min + 10%
wait 3min
quiescentModeEnabled **= QUERY QUIESCENT MODE**
if (*quiescentModeEnabled* **!= NO)**
    error 5 Quiescent mode still enabled.
endif
*status***= QUERY DEVICE STATUS**
if (*status* **!= XXXX XX0Xb)**
    error 6 Quiescent mode still enabled in device status.
endif
queryMode **= FindFrame** (*record*, **QUERY QUESCIENT MODE**)
queryStatus **= FindFrame** (*record*, **QUERY DEVICE STATUS**)
startMode **= FindFrame** (*record*, **START QUIESCENT MODE**)
others **= FindFrame** (*record*, **any other forward frame**)
if (!((*queryMode* **== 2) AND (***queryStatus* **== 2) AND (***startMode* **== 1) AND (***others* **== 0)))**
    error 7 Unexpected commands received while quiescent mode was enabled.
endif
// reset device and disable it
ResetDevice(false)

12.4.16 Device power cycle notification

This sequence first enables the power cycle notification (ENABLE POWER CYCLE NOTIFICATION) and then generates a power cycle. After this the bus is checked for the power cycle notification event message being sent out by the DUT earliest 1,3 s after power cycle completion and maximum after 5 s + power on time for the DUT.

In the second step the power cycle notification is disabled (DISABLE POWER CYCLE NOTIFICATION) and a power cycle generated. The bus is checked for 5 s + power on time for the DUT, not containing any power cycle notification event message.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice(false)
// enable power cycle notification and check state

```
ENABLE_POWER_CYCLE_NOTIFICATION
enabled = QUERY_POWER_CYCLE_NOTIFICATION
if (enabled == NO)
    error 1 Power cycle notification was not enabled. Actual: NO. Expected: YES.
endif
// Check for POWER_NOTIFICATION sent between 1,3 s and 5 s after power cycle
PowerCycleAndWaitForDecoder (5)
StartBusRecording (record)
start_timer (timer)
do
    powerNotificationFound = FindFrame (record, POWER_NOTIFICATION)
    timestamp = get_timer (timer) // time in ms
while (powerNotificationFound == 0 AND timestamp < 10000)
if (powerNotificationFound > 0)
    if (timestamp < 1 300)
        error 2 POWER_NOTIFICATION sent earlier than 1.3s after power cycle. Actual:
        timestamp ms. Expected: >= 1300ms.
    else if (timestamp > 5 000)
        error 3 POWER_NOTIFICATION sent later than 5s after power cycle. Actual:
        timestamp ms. Expected: <= 5000ms.
    else
        report 1 NOTIFICATION was sent after timestamp ms.
    endif
else
    error 4 POWER_NOTIFICATION was not sent within 10 s after power cycle.
endif
// check state after power cycle
enabled = QUERY_POWER_CYCLE_NOTIFICATION
if (enabled == NO)
    error 5 Power cycle notification is not enabled after power cycle.
endif
// disable power cycle notification and check state
DISABLE_POWER_CYCLE_NOTIFICATION
enabled = QUERY_POWER_CYCLE_NOTIFICATION
if (enabled != NO)
    error 6 Power cycle notification is enabled.
endif
// Check for no POWER_NOTIFICATION within 10 s after power cycle.
PowerCycleAndWaitForDecoder (5)
StartBusRecording (record)
start_timer (timer)
do
    powerNotificationFound = FindFrame (record, POWER_NOTIFICATION)
    timestamp = get_timer (timer) // time in ms
while (powerNotificationFound == 0 AND timestamp < 10 000)
if (powerNotificationFound > 0)
    error 7 POWER_NOTIFICATION was sent timestamp ms after power cycle with power
    cycle notification disabled.
endif
// check state after power cycle
enabled = QUERY_POWER_CYCLE_NOTIFICATION
if (enabled != NO)
    error 8 Power cycle notification is enabled after power up.
endif
StopBusRecording (record)
ResetDevice(false)
```

12.4.17 SET SHORT ADDRESS

The test sequence checks if storage of the short address is correctly programmed using the short address programmed in the step before. QUERY MISSING SHORT ADDRESS and the short address bit of the QUERY STATUS answer are also tested.

Test sequence shall be run for each selected logical unit.

Test description:

oldAddress = GLOBAL_currentUnderTestLogicalUnit

lastAssignedAddress = oldAddress

if (*GLOBAL_numberShortAddresses* < 62)

    numberNotAssignedAddresses = 62 - *GLOBAL_numberShortAddresses* + 1

    intermediateAddress = *GLOBAL_numberShortAddresses* + (*oldAddress* % *numberNotAssignedAddresses*)

    iEnd = 7

else

    iEnd = 6

endif

for (*i* = 0; *i* < *iEnd*; *i*++)

    DTR0 (*value*[*i*])

    SET SHORT ADDRESS, send to device *address1*[*i*]

    answer = QUERY MISSING SHORT ADDRESS, send to device *address2*[*i*]

    if (*answer* != *test1*[*i*])

        error 1 Wrong answer at QUERY MISSING SHORT ADDRESS at test step i = *i*. Actual: *answer*. Expected: *test1*[*i*].

    endif

    answer = QUERY STATUS, send to *address2*[*i*]

    if (*answer* != *test2*[*i*])

        error 2 Wrong answer at QUERY STATUS at test step i = *i*. Actual: *answer*. Expected: *test2*[*i*].

    endif

    lastAssignedAddress = *address2*[*i*]

endfor

SetShortAddress (*lastAssignedAddress*; *oldAddress*)

Table 46 – Parameters for test sequence SET SHORT ADDRESS

Test step i	value	description	address1	address2	test1	test2
0	63	short address 63	**ShortAddress** (*oldAddress*)	**ShortAddress** (63)	NO	XXXXX0XXb
1	MASK	delete short address	**ShortAddress** (63)	**Broadcast Unaddressed**	YES	XXXXX1XXb
2	*oldAddress*	*oldAddress*	**Broadcast Unaddressed**	**ShortAddress** (*oldAddress*)	NO	XXXXX0XXb
3	64	no change	**ShortAddress** (*oldAddress*)	**ShortAddress** (*oldAddress*)	NO	XXXXX0XXb
4	128	no change	**ShortAddress** (*oldAddress*)	**ShortAddress** (*oldAddress*)	NO	XXXXX0XXb
5	254	no change	**ShortAddress** (*oldAddress*)	**ShortAddress** (*oldAddress*)	NO	XXXXX0XXb
6	*Intermediate Address*	a short address	**ShortAddress** (*oldAddress*)	**ShortAddress** (*intermediateAddress*)	NO	XXXXX0XXb

12.4.18 Reset/Power-on values (device)

The test sequence checks the reset and the power-on values of the 103 variables. The reset and power-on values of the 3xx variables are assumed to be tested in IEC 62386-3xx standard.

Test sequence shall be run for each selected logical unit.

Test description:

operatingMode = QUERY OPERATING MODE

capabilities = QUERY DEVICE CAPABILITIES

numberInstances = QUERY NUMBER OF INSTANCES

shortAddress = *GLOBAL_currentUnderTestLogicalUnit*

// Change value of 103 variables

INITIALISE (*shortAddress*)

RANDOMISE

wait 100 ms

TERMINATE

randomAddress = **GetRandomAddress** ()

for (*i* = 6; *i* < 12; *i*++)

    command[*i*]

endfor

// Perform a RESET

RESETDEVICE ()

// Check ROM variables after reset

```
for (i = 01; i < 97; i++)
    if (GLOBAL_logicalUnit[shortAddress].extendedVersions[i] != -1)
        version = QUERY EXTENDED VERSION NUMBER
        if (version != GLOBAL_logicalUnit[shortAddress].extendedVersions[i])

            error 1 LogicalUnit shortAddress: Wrong extendedVersionNumber for part 3i after
            RESET. Actual: version. Expected: GLOBAL_logicalUnit[shortAddress].
            extendedVersions [i].

        endif

    endif

endfor

// Check reset value of 103 variables

for (i = 0; i < 12; i++)

    answer = query[i]

    if (answer != reset[i])

        error 2 Wrong reset value for variable[i]. Answer: answer. Expected: reset[i].

    endif

endfor

// Change value of 103 variables

INITIALISE (shortAddress)

RANDOMISE

wait 100 ms

TERMINATE

randomAddress = GetRandomAddress ()

for (i = 6; i < 12; i++)

    command[i]

endfor

// Perform a power cycle

SAVE PERSISTENT VARIABLES

wait 1 s

PowerCycleAndWaitForDecoder (60)

// Check ROM variables after power cycle

for (i = 01; i < 97; i++)
    if (GLOBAL_logicalUnit[shortAddress].extendedVersions[i] != -1)
        version = QUERY EXTENDED VERSION NUMBER
        if (version != GLOBAL_logicalUnit[shortAddress].extendedVersions[i])

            error 1 LogicalUnit shortAddress: Wrong extendedVersionNumber for part 3i after
            RESET. Actual: version. Expected: GLOBAL_logicalUnit[shortAddress].
            extendedVersions [i].
```

endif

endif

endfor

// Check power on value of 103 variables

for (*i* = 0; *i* < 12; *i*++)

answer = query[i]

if (*answer* != *powerOn[i]*)

error 4 Wrong power on value for *variable[i]*. Answer: *answer*. Expected: *powerOn[i]*.

endif

endfor

Table 47 – Parameters for test sequence Reset/Power-on values (device)

Test step i	command	query	variable	reset	powerOn
0	-	QUERY OPERATING MODE	operatingMode	*operatingMode*	*operatingMode*
1	-	QUERY DEVICE CAPABILITIES	ApplicationControllerPresent, numberOfInstances	*capabilities*	*capabilities*
2	-	**GetRandomAddress** ()	randomAddress	0xFF FF FF	*randomAddress*
3	-	**GetVersionNumber** ()	versionNumber	2.0	2.0
4	-	QUERY NUMBER OF INSTANCES	numberOfInstances	*numberInstances*	*numberInstances*
5	-	QUERY POWER CYCLE SEEN	powerCycleSeen	NO	YES
6	**AddDeviceGroups** (0xFFFF FFFF)	**GetDeviceGroups** ()	deviceGroups	0x0000 0000	0xFFFF FFFF
7	DTR0 (0xAA)	QUERY CONTENT DTR0	DTR0	0xAA	0x00
8	DTR1 (0xAB)	QUERY CONTENT DTR1	DTR1	0xAB	0x00
9	DTR2 (0xAC)	QUERY CONTENT DTR2	DTR2	0xAC	0x00
10	START QUIESCENT MODE	QUERY QUIESCENT MODE	quiescentMode	NO	NO
11	ENABLE POWER CYCLE NOTIFICATION	QUERY POWER CYCLE NOTIFICATION	powerCycle Notification	YES	YES

12.4.19 Reset/Power-on values (instance)

The test sequence checks the reset and the power-on values of the 103 instance variables. The reset and power-on values of the 3xx instance variables are assumed to be tested in IEC 62386-3xx standard.

Test sequence shall be run for each selected logical unit.

Test description:

```
numberInstances = QUERY NUMBER OF INSTANCES

shortAddress = GLOBAL_currentUnderTestLogicalUnit

if (numberOfInstances == 0)
    report 1 No instances available
else

    for (k = 0; k < numberOfInstances; k++)

        loopEnd = 6

        if (GLOBAL_logicalUnit[shortAddress].instanceTypes[k] == 0)

            loopEnd++

        endif

        // Change value of 103 instance variables

        DTR0 (4)

        for (i = 0; i < loopEnd; i++)

            command[i], sent to instance InstanceNumber (k)

        endfor

        // Perform a RESET

        RESETDEVICE ()

        // Check reset value of 103 instance variables

        for (i = 0; i < loopEnd; i++)

            answer = query[i], sent to instance InstanceNumber (k)

            if (answer != reset[i])

                error 1 Wrong reset value for variable[i] at instance [k]. Answer: answer.
                Expected: reset[i].

            endif

        endfor

        //Change value of 103 instance variables

        DTR0 (4)

        for (i = 0; i < loopEnd; i++)

            command[i], sent to instance InstanceNumber (k)

        endfor

        // Perform a power cycle

        SAVE PERSISTENT VARIABLES

        wait 1 s

        PowerCycleAndWaitForDecoder (60)

        // Check power on value of 103 instance variables
```

for (*i* = 0; *i* < *loopEnd*; *i*++)

answer = *query*[*i*], sent to instance **InstanceNumber** (*k*)

if (*answer* != *powerOn*[*i*])

error 2 Wrong power on value for *variable*[*i*] at instance [*k*]. Answer: *answer*. Expected: *powerOn*[*i*].

endif

endfor

endfor

endif

Table 48 – Parameters for test sequence Reset/Power-on values (instance)

Test step i	command	query	variable	reset	powerOn
0	SET PRIMARY INSTANCE GROUP	QUERY PRIMARY INSTANCE GROUP	instanceGroup0	MASK	4
1	SET INSTANCE GROUP 1	QUERY INSTANCE GROUP 1	instanceGroup1	MASK	4
2	SET INSTANCE GROUP 2	QUERY INSTANCE GROUP 2	instanceGroup2	MASK	4
3	DISABLE INSTANCE	QUERY INSTANCE ENABLED	instanceActive	NO	NO
4	SET EVENT SCHEME	QUERY EVENT SCHEME	eventScheme	0	4
5	SET EVENT PRIORITY	QUERY EVENT PRIORITY	eventPriority	4	4
6	**SetEventFilter** (0x010203)	**GetEventFilter** ()	eventFilter	0xFF FFFF	0x010203

12.4.20 DTR0 / DTR1 / DTR2

The test sequence checks the correct function of the DTR registers, by reading from and writing to them. They need to be correct before proceeding with the next tests.

Test sequence shall be run for all logical units in parallel.

Test description:

for (*i* = 0; *i* < 9; *i*++)

DTR0 (*data0*[*i*])

DTR1 (*data1*[*i*])

DTR2 (*data2*[*i*])

answer = QUERY CONTENT DTR0

if (*answer* != *data0*[*i*])

error 1 Wrong value of DTR0 stored at test step i = *i*. Actual: *answer*. Expected: *data0*[*i*].

endif

answer = QUERY CONTENT DTR1

if (*answer* != *data1*[*i*])

error 2 Wrong value of DTR1 stored at test step i = *i*. Actual: *answer*. Expected: *data1*[*i*].

endif

answer = QUERY CONTENT DTR2

if (*answer* != *data2*[*i*])

error 3 Wrong value of DTR2 stored at test step i = *i*. Actual: *answer*. Expected: *data2*[*i*].

endif

endfor

Table 49 – Parameters for test sequence DTR0 / DTR1 / DTR2

Test step i	data0	data1	data2
0	00000001b	11111111b	10000000b
1	00000010b	00000001b	11111111b
2	00000100b	00000010b	00000001b
3	00001000b	00000100b	00000010b
4	00010000b	00001000b	00000100b
5	00100000b	00010000b	00001000b
6	01000000b	00100000b	00010000b
7	10000000b	01000000b	00100000b
8	11111111b	10000000b	01000000b

12.4.21 DTR1:DTR0 and DTR2:DTR1

This test procedure is checking for correct writing of two bytes to two DTR within one command. Several different pairs of test values are written and queried again, to verify correct operation of these DTR related commands.

Test description:

ResetDevice()
// clear DTRs before testing DTR1:DTR0 command
DTR0 (0)
DTR1 (0)
DTR2 (0)
for (*i*=0; *i* < 9; *i*++)
// check if DTR1:DTR0 is set correctly and is not changing DTR2 content
DTR2 (*data2*[*i*])
DTR1:DTR0 (*data1*[*i*], *data0*[*i*])

answer = QUERY CONTENT DTR0
if (*answer* != *data0*[*i*])
error 1 Wrong value of DTR0 stored at test step i = *i*. Received: *answer*. Expected: *data0*[*i*].
endif
answer = QUERY CONTENT DTR1
if (*answer* != *data1*[*i*])
error 2 Wrong value of DTR1 stored at test step i = *i*. Received: *answer*. Expected: *data1*[*i*].
endif

answer = QUERY CONTENT DTR2
if (*answer* != *data2*[*i*])
error 3 Wrong value of DTR2 stored at test step i = *i*. Received: *answer*. Expected: *data2*[*i*].
endif
endfor
// clear DTRs before testing DTR2:DTR1 command
DTR0 (0)
DTR1 (0)
DTR2 (0)
for (*i*=0; *i* < 9; *i*++)
// check if DTR2:DTR1 is set correctly and is not changing DTR0 content
DTR0 (*data0*[*i*])
DTR2:DTR1 (*data2*[*i*], *data1*[*i*])
answer = QUERY CONTENT DTR0
if (*answer* != *data0*[*i*])
error 4 Wrong value of DTR0 stored at test step i = *i*. Received: *answer*. Expected: *data0*[*i*].
endif
answer = QUERY CONTENT DTR1
if (*answer* != *data1*[*i*])
error 5 Wrong value of DTR1 stored at test step i = *i*. Received: *answer*. Expected: *data1*[*i*].
endif
answer = QUERY CONTENT DTR2
if (*answer* != *data2*[*i*])
error 6 Wrong value of DTR2 stored at test step i = *i*. Received: *answer*. Expected: *data2*[*i*].
endif
endfor
ResetDevice()

Table 50 – Parameters for test sequence DTR1:DTR0 and DTR2:DTR1

Test step i	data0	data1	data2
0	00000001b	11111111b	10000000b
1	00000010b	00000001b	11111111b
2	00000100b	00000010b	00000001b
3	00001000b	00000100b	00000010b
4	00010000b	00001000b	00000100b
5	00100000b	00010000b	00001000b
6	01000000b	00100000b	00010000b
7	10000000b	01000000b	00100000b
8	11111111b	10000000b	01000000b

12.4.22 Device Groups

This sequence checks setting and device group memberships by reading the device group assignments and also the reaction on device group addressing using QUERY DEVICE CAPABILITIES.

Test description:

ResetDevice()
// clear group assignment
ClearAllDeviceGroups()
// check if no group is added

AddDeviceGroups(0x0000 0000)
CheckGroupAssignment(0x0000 0000)
// check if single group is added
AddDeviceGroups(0x0000 0100)
CheckGroupAssignment(0x0000 0100)
// check if multiple groups are added
AddDeviceGroups(0x0100 0007)
CheckGroupAssignment(0x0100 0107)
// check if multiple groups are added, including which are already added
AddDeviceGroups(0x0007 0107)
CheckGroupAssignment(0x0107 0107)
// check if single groups are removed
RemoveDeviceGroups(0x0100 0000)
CheckGroupAssignment(0x0007 0107)
// check if multiple groups are removed
RemoveDeviceGroups(0x0007 0100)
CheckGroupAssignment(0x0000 0007)
// check if multiple groups are removed, including which are already removed
RemoveDeviceGroups(0x0007 0007)
CheckGroupAssignment(0x0000 0000)
ResetDevice()

12.5 Device queries

12.5.1 Device query capabilities

This sequence reads the device features (QUERY DEVICE CAPABILITIES) and checks for the unused bits (2 to 7) being zero.

If bit 1 ("numberOfInstances" is greater than 0?) is set, the test procedure checks also the number of instances (QUERY NUMBER OF INSTANCES) being greater than zero.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice ()

capabilities = QUERY DEVICE CAPABILITIES
numberOfInstances = **GetNumberOfInstances** ()

//check reserved bits
if (*capabilities* != 0000 00XX)
    error 1 Reserved bits of device features are not 0
endif

// check application controller
if (*capabilities* == XXXX XXX1b)
    report 1 Application controller is present
else
    report 2 No application controller is present
endif

// check number of instances
if (*capabilities* == XXXX XX1Xb AND *numberOfInstances* > 0)
    report 3 *numberOfInstances* Instance(s) is/are present
else
    if (*features* == XXXX XX0Xb AND *numberOfInstances* == 0)
        report 4 No Instances are present

else

error 2 Device features bit 2 and numberOfInstances are not consistent

endif

endif

ResetDevice ()

12.5.2 QUERY VERSION NUMBER

The test sequence checks the version number of the standard, implemented by DUT.

Test sequence shall be run for all logical units in parallel.

Test description:

// Get version using QUERY VERSION NUMBER

answer = **GetVersionNumber** ()

if (*answer* == 2.0)

report 1 Version number is *answer*.

else if (*answer* == 2)

error 1 Version number is *answer*, but it does not have the right format. Expected: 2.0.

else

error 2 Incorrect version number. Actual: *answer*. Expected: 2.0.

endif

// Compare version number given in memory bank 0 with the answer to Query Version Number

DTR0 (0x17)

DTR1 (0)

answerMB = READ MEMORY LOCATION

answerQVN = QUERY VERSION NUMBER, **accept** Value

if (*answerMB* != *answerQVN*)

error 3 Version number given in the memory bank 0 does not match to the answer of QUERY VERSION NUMBER. Version number given in memory bank 0: *answerMB*. Answer to Query Version Number: *answerQVN*.

endif

12.5.3 Device power cycle seen

This sequence is testing for the bit “powerCycleSeen”. The bit is first cleared by the test sequence and checked through QUERY DEVICE STATUS (Bit5). After a power cycle is done the status of “powerCycleSeen” is checked again to be set. After another RESET POWER CYCLE SEEN command the flag is checked again for being cleared.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice()
// clear flag first and check
RESET POWER CYCLE SEEN
status = QUERY DEVICE STATUS
if (*status* **!= XX0X XXXXb)**
    error 1 PowerCycleSeen not cleared after RESET POWER CYCLE SEEN. Actual: *status*. Expected: XX0X XXXXb.
endif
// check if flag is set after power cycle
PowerCycleAndWaitForDecoder (5)
status = QUERY DEVICE STATUS
if (*status* **!= XX1X XXXXb)**
    error 2 PowerCycleSeen not set after power cycle. Actual: *status*. Expected: XX1X XXXXb.
endif
// reset powerCycleSeen and check for powerCycleSeen cleared
RESET POWER CYCLE SEEN
status = QUERY DEVICE STATUS
if (*status* **== XX1X XXXXb)**
    error 3 PowerCycleSeen not cleared after RESET POWER CYCLE SEEN. Actual: *status*. Expected: XX0X XXXXb.
endif
ResetDevice()

12.5.4 Input device error

This test sequence is checking for the application error information is the same from QUERY INPUT DEVICE ERROR and Bit0 of QUERY DEVICE STATUS. If there is no error reported in the status byte (QUERY DEVICE STATUS Bit0 equals zero), also the query input device error shall show no answer. If there is an error reported in the status byte, the query input device error shall show up with some value (manufacturer specific).

There is a warning, if an error state is detected, as it is strongly recommended to resolve the error before conducting the test.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice (true)
numberOfInstances = **GetNumberOfInstances** ()
if (*numberOfInstances* == 0)
    report 1 No instances available
    //check if non-existent input device has errors
    error = QUERY INPUT DEVICE ERROR
    if (*error*)
        error 1 Wrong answer without existent input device
    endif
    status = QUERY DEVICE STATUS
    if (*status* == XXXX XXX1)
        error 2 Wrong answer without existent input device
    endif
else
    //check if input device has errors
    error = QUERY INPUT DEVICE ERROR
    status = QUERY DEVICE STATUS
    if (*status* == XXXX XXX1 AND *error* != NO)

error 3 Input device has an error code *error* and report is consistent. Please resolve error before conducting this test.

else if ((*status* == XXXX XXX0 AND *error* != NO) OR (*status* == XXXX XXX1 AND *error* == NO))

error 4 Input device has an error code *error* but report is inconsistent

endif

endif

ResetDevice (false)

12.6 Device Memory banks

12.6.1 READ MEMORY LOCATION on Memory Bank 0

The test sequence checks the correct function of the READ MEMORY LOCATION command and whether memory bank 0 is implemented according to specification.

Test sequence shall be run for all logical units in parallel.

Test description:

DTR0 (0)

DTR1 (0)

answer = READ MEMORY LOCATION

if (*answer* == NO)

error 1 No answer received when reading the size of memory bank 0. Actual: NO. Expected: not NO.

else *// Read memory bank 0*

// Read memory bank location 0x00, which may differ per logical unit

DTR1 (0)

DTR2 (128)

for (*address = 0; address < GLOBAL_numberShortAddresses; address++*)

DTR0 (0)

laml[*address*] = READ MEMORY LOCATION, send to device **ShortAddress** (*address*)

if (*answer* == NO)

error 2 LogicalUnit *address*: No answer received when reading memory bank location 0. Actual: NO. Expected: not NO.

else

if (*laml*[*address*] < 0x1A) *// Check that all mandatory memory bank locations are implemented*

error 3 LogicalUnit *address*: Not all mandatory memory locations implemented. Actual: *laml*[*address*]. Expected: answer >= 0x1A.

endif

if (*laml*[*address*] >= 0x1B AND *laml*[*address*] <= 0x7F) *// Check that none of the reserved memory bank locations [0x1B,0x7F] is given as last accessible memory location*

error 4 LogicalUnit *address*: Reserved memory bank location *laml*[*address*] returned as last memory bank location. Actual: *laml*[*address*]. Expected: 0x1A or [0x80,0xFE].

endif

if (*laml*[*address*] == 0xFF) *// Check that reserved memory bank location 0xFF is not given as last accessible memory location*

error 5 LogicalUnit *address*: Reserved 0xFF memory bank location returned as last memory bank location. Actual: *laml*[*address*]. Expected: 0x1A or [0x80,0xFE].

endif

endif

answer = QUERY CONTENT DTR0, send to device **ShortAddress** (*address*) *// Check that DTR0 incremented after reading a valid memory bank location*

if (*answer* != 1)

error 6 LogicalUnit *address*: DTR0 not incremented after reading a valid memory bank location. Actual: *answer*. Expected: 1.

endif

answer = QUERY CONTENT DTR1, send to device **ShortAddress** (*address*) *// Check that DTR1 not changed after reading memory bank location*

if (*answer* != 0)

error 7 LogicalUnit *address*: DTR1 modified after reading a valid memory bank location. Actual: *answer*. Expected: 0.

DTR1 (0)

endif

answer = QUERY CONTENT DTR2, send to device **ShortAddress** (*address*) *// Check that DTR2 not changed after reading memory bank location*

if (*answer* != 128)

error 8 LogicalUnit *address*: DTR2 modified after reading a valid memory bank location. Actual: *answer*. Expected: 128.

DTR2 (128)

endif

endfor

// Read memory bank location 0x01, which shall not be implemented on the logical units

DTR0 (1)

DTR2 (64)

answer = READ MEMORY LOCATION *// Check that reserved memory bank location 0x01 is not implemented*

if (*answer* != NO)

error 9 Answer received when reading reserved - not implemented memory bank location 0x01. Actual: *answer*. Expected: NO.

endif

answer = QUERY CONTENT DTR0 *// Check that DTR0 incremented after reading a not implemented memory bank location*

if (*answer* != 2)

error 10 DTR0 not incremented after reading a not implemented memory bank location. Actual: *answer*. Expected: 2.

endif

answer = QUERY CONTENT DTR1 *// Check that DTR1 not changed after reading memory bank location*

if (*answer* != 0)

error 11 DTR1 modified after reading a not implemented memory bank location. Actual: *answer*. Expected: 0.

DTR1 (0)

endif

answer = QUERY CONTENT DTR2 *// Check that DTR2 not changed after reading memory bank location*

if (*answer* != 64)

error 12 DTR2 modified after reading a not implemented memory bank location. Actual: *answer*. Expected: 64.

endif

// Read the content of must be implemented memory bank locations

// Read memory bank location 0x02, which may differ per logical unit

for (*address* = 0; *address* < *GLOBAL_numberShortAddresses*; *address*++)

DTR0 (*address*)

answer = READ MEMORY LOCATION, send to device **ShortAddress** (*address*)

if (*answer* == NO)

error 13 LogicalUnit *address*: An error occurred when reading valid memory bank location 0x02.

else

if (*answer* <= 199)

report 1 LogicalUnit *address*: Last accessible memory bank = *answer*.

else

error 14 LogicalUnit *address*: Wrong last accessible memory bank reported. Actual: *answer*. Expected: [0, 199].

```
            endif
        endif
        answer = QUERY CONTENT DTR0, send to device ShortAddress (address) // Check that DTR0 incremented after reading a valid memory bank location
        if (answer != 3)
            error 15 LogicalUnit address: DTR0 not incremented after reading valid memory bank location. Actual: answer. Expected: 3.
        endif
    endfor
    // Read memory bank locations 0x03 - 0x19, which shall be common for all logical units
    loc0x18 = 0
    DTR0 (3)
    for (i = 0; i < 11; i++)
        multibyte = ReadMemBankMultibyteLocation (nrBytes[i])
        if (multibyte != -1)
            // multibyte shall be displayed as a decimal value
            report 2 text[i] = multibyte
            // Check allowed range for each memory bank location
            if (address[i] == 0x18)
                loc0x18 = multibyte
            endif
            if (address[i] == 0x15)
                if (multibyte == 0xFF)
                    error 16 For this DUT, the 101 standard must be implemented.
                else if (multibyte < 00001000b)
                    error 17 text[i] must be at least 2.0 (00001000b).
                endif
            else if (address[i] == 0x16)
                if (multibyte == 0xFF)
                    report 3 For this DUT, the 102 standard is not implemented.
                else if (multibyte != 00001000b)
                    error 18 text[i] must be 2.0 (00001000b).
                endif
            else if (address[i] == 0x17)
```

if (*multibyte* == 0xFF)

error 19 For this DUT, the 103 standard must be implemented.

else if (*multibyte* < 00001000b)

error 20 *text*[*i*] must be at least 2.0 (00001000b).

endif

else if (*address*[*i*] == 0x18 AND (*multibyte* < 1 OR *multibyte* > 64))

error 21 *text*[*i*] must be in the range [1,64].

else if (*address*[*i*] == 0x19 AND *multibyte* > 64)

error 22 *text*[*i*] must be in the range [0,64].

endif

answer = QUERY CONTENT DTR0 *// Check that DTR0 incremented after reading a valid memory bank location*

if (*answer* != *dtrValue*[*i*])

error 23 DTR0 not incremented after reading valid memory bank location. Actual: *answer*. Expected: *dtrValue*[*i*].

DTR0 (*dtrValue*[*i*])

endif

else

error 24 An error occurred when reading valid memory bank location *address*[*i*].

DTR0 (*dtrValue*[*i*])

endif

endfor

// Read memory bank location 0x1A, which should differ per logical unit

if (*loc0x18* == 0)

error 25 Location 0x1A cannot be verified since an error occurred when reading location 0x18.

else

for (*address* = 0; *address* < *GLOBAL_numberShortAddresses*; *address*++)

DTR0 (0x1A)

answer = READ MEMORY LOCATION, send to device **ShortAddress** (*address*)

if (*answer* != NO)

if (*answer* > (*loc0x18* - 1))

error 26 LogicalUnit *address*: Index number of this logical control gear unit must be in the range [0, *loc0x18* - 1].

else

report 4 LogicalUnit *address*: Index number of this logical control gear unit = *answer*.

endif

else

error 27 LogicalUnit *address*: An error occurred when reading valid memory bank location 0x1A.

endif

answer = QUERY CONTENT DTR0, send to device **ShortAddress** (*address*) *// Check that DTR0 incremented after reading a valid memory bank location*

if (*answer* != 0x1B)

error 28 LogicalUnit *address*: DTR0 not incremented after reading memory bank location 0x1A. Actual: *answer*. Expected: 0x1B.

endif

endfor

endif

// Check that reserved memory bank locations 0x1B-0x7F are not implemented in none of the logical units

DTR0 (0x1B)

for (*i* = 0x1B; *i* <= 0x7F; *i*++)

answer = READ MEMORY LOCATION

if (*answer* != NO)

error 29 Answer received when reading the not implemented memory bank location *i*. Actual: *answer*. Expected: NO.

endif

answer = QUERY CONTENT DTR0 *// Check that DTR0 incremented after reading a not implemented memory bank location*

if (*answer* != *i* + 1)

error 30 DTR0 not incremented after reading the not implemented memory bank location *i*. Actual: *answer*. Expected: *i* + 1.

DTR0 (*i* + 1)

endif

endfor

for (*address* = 0; *address* < *GLOBAL_numberShortAddresses*; *address*++)

// If available, read the additional control gear information, per logical unit

DTR0 (0x80)

additionalData = 0

for (*i* = 0x80; *i* <= 254; *i*++)

answer = READ MEMORY LOCATION, send to device **ShortAddress** (*address*)

if (*answer* != NO)

additionalData = *additionalData* + 1

report 5 LogicalUnit *address*: Additional control gear information at location *i* is *answer*.

if (*i* > *laml*[*address*])

error 31 LogicalUnit *address*: Additional control gear information found at location *i* . Last accessible memory location is *laml*[*address*].

endif

endif

answer = QUERY CONTENT DTR0, send to device **ShortAddress** (*address*)

// Check that DTR0 incremented after reading a valid memory bank location

if (*answer* != *i* + 1)

error 32 LogicalUnit *address*: DTR0 not incremented after reading valid memory bank location *i*. Actual: *answer*. Expected: *i* + 1.

DTR0 (*i* + 1)

endif

endfor

if (*additionalData* == 0 AND *laml*[*address*] >= 0x80)

error 33 LogicalUnit *address*: No answer received when reading additional data, however additional data expected.

endif

endfor

// Read the last memory bank location

DTR0 (0xFF)

answer = READ MEMORY LOCATION

if (*answer* != NO)

error 34 Answer received when reading memory location 0xFF. Actual: *answer*. Expected: NO.

endif

answer = QUERY CONTENT DTR0

if (*answer* != 255)

error 35 DTR0 modified after reading memory location 0xFF. Actual: *answer*. Expected: 255.

endif

endif

Table 51 – Parameters for test sequence READ MEMORY LOCATION on Memory Bank 0

Test step i	address	nrBytes	dtrValue	text
0	0x03	6	9	GTIN (decimal)
1	0x09	1	10	Firmware version (major)
2	0x0A	1	11	Firmware version (minor)
3	0x0B	8	19	Serial number (decimal)
4	0x13	1	20	HW version (major)
5	0x14	1	21	HW version (minor)
6	0x15	1	22	101 version number
7	0x16	1	23	102 version number
8	0x17	1	24	103 version number
9	0x18	1	25	Number of logical control devices units in the bus unit
10	0x19	1	26	Number of logical control gear units in the bus unit

12.6.2 READ MEMORY LOCATION on Memory Bank 1

The test sequence checks the correct function of the READ MEMORY LOCATION command and whether memory bank 1 is implemented according to specification.

Test sequence shall be run for each selected logical unit.

Test description:

DTR0 (0)

DTR1 (1)

laml = READ MEMORY LOCATION

if (*laml* != NO)

report 1 Memory bank 1 is implemented and the last accessible memory location is *laml*.

if (*laml* < 0x10) *// Check that all mandatory memory bank locations are implemented*

error 1 Not all mandatory memory locations implemented. Actual: *laml*. Expected: answer >= 0x10.

endif

if (*laml* == 0xFF) *// Check that reserved memory bank location 0xFF is not given as last accessible memory location*

error 2 Reserved 0xFF memory bank location. Actual: *laml*. Expected: [0x10,0xFE].

endif

answer = QUERY CONTENT DTR0 *// Check that DTR0 incremented after reading a valid memory bank location*

if (*answer* != 1)

error 3 DTR0 not incremented after reading a valid memory bank location. Actual: *answer*. Expected: 1.

endif

answer = QUERY CONTENT DTR1 *// Check that DTR1 not changed after reading memory bank location*

if (*answer* != 1)

error 4 DTR1 modified after reading a valid memory bank location. Actual: *answer*. Expected: 1.

endif

DTR0 (1)

DTR1 (1)

answer = READ MEMORY LOCATION *// Read the indicator byte location 0x01*

report 2 Indicator byte (memory bank 1 location 1) = *answer*

answer = QUERY CONTENT DTR0 *// Check that DTR0 incremented after reading the manufacturer specific memory bank location*

if (*answer* != 2)

error 5 DTR0 not incremented after reading the manufacturer specific memory bank location. Actual: *answer*. Expected: 2.

endif

answer = QUERY CONTENT DTR1 *// Check that DTR1 not changed after reading memory bank location*

if (*answer* != 1)

error 6 DTR1 modified after reading the manufacturer specific memory bank location. Actual: *answer*. Expected: 1.

DTR1 (1)

endif

// Read the content of must be implemented memory bank locations

DTR0 (2)

for (*i* = 0; *i* < 3; *i*++)

if (*address*[*i*] + *nrBytes*[*i*] - 1 <= *laml*)

multibyte = **ReadMemBankMultibyteLocation** (*nrBytes*[*i*])

if (*multibyte* != -1)

report 3 *text*[*i*] = *multibyte*

else

error 7 An error occurred when reading valid memory bank location (*text*[*i*]).

endif

else

error 8 Not all mandatory memory bank locations implemented.

endif

endfor

// Read above last accessible memory bank location

DTR0 (*laml* + 1)

for (*i* = *laml* + 1; *i* <= 0xFE; *i*++)

answer = READ MEMORY LOCATION

if (*answer* != NO)

error 9 Answer received when reading not implemented memory bank location *i*. Actual: *answer*. Expected: NO.

endif

answer = QUERY CONTENT DTR0

if (*answer* != *i* + 1)

error 10 DTR0 not incremented after reading above the last accessible memory location. Actual: *answer*. Expected: *i* + 1.

endif

endfor

// Read the last memory bank location

DTR0 (0xFF)

answer = READ MEMORY LOCATION

if (*answer* != NO)

error 11 Answer received when reading memory location 0xFF. Actual: *answer*. Expected: NO.

endif

answer = QUERY CONTENT DTR0

if (*answer* != 255)

error 12 DTR0 modified after reading memory location 0xFF. Actual: *answer*. Expected: 255.

endif

else

report 4 Memory bank 1 is not implemented.

// Check that indeed memory bank 1 is not implemented

for (*i* = 1; *i* <= 255; *i*++)

DTR0 (*i*)

answer = READ MEMORY LOCATION

if (*answer* != NO)

error 13 Answer received when reading the not implemented memory bank 1, location *i*. Actual: *answer*. Expected: NO.

endif

answer = QUERY CONTENT DTR0

if (*answer* != *i*)

error 14 DTR0 modified after reading the not implemented memory bank 1, location *i*. Actual: *answer*. Expected: *i*.

endif

answer = QUERY CONTENT DTR1 *// Check that DTR1 not changed after reading memory bank location*

if (*answer* != 1)

error 15 DTR1 modified after reading the not implemented memory bank 1, location *i*. Actual: *answer*. Expected: 1.

DTR1 (1)

endif

endfor

endif

Table 52 – Parameters for test sequence READ MEMORY LOCATION on Memory Bank 1

Test step i	address	nrBytes	text
0	2	1	Memory bank 1 lock byte
1	3	6	OEM GTIN (decimal)
2	9	8	OEM Serial number (decimal)

12.6.3 READ MEMORY LOCATION on other Memory Banks

The test sequence checks the correct function of the READ MEMORY LOCATION command and checks if all other memory banks besides 0 and 1 are implemented according to specification.

Test sequence shall be run for each selected logical unit.

Test description:

DTR0 (2)

DTR1 (0)

lamb = READ MEMORY LOCATION

if (*lamb* < 2)

report 1 No other memory bank besides 0 or 1 are implemented.

else

if (*lamb* <= 199)

report 2 Last accessible memory bank is *lamb*.

else

error 1 Wrong last accessible memory bank reported. Actual: *lamb*. Expected: [2, 199].

lamb = 199

endif

// Find an implemented memory bank between MB 2 and MB lamb

for (*i* = 2; *i* <= *lamb*; *i*++)

DTR0 (0)

DTR1 (*i*)

laml = READ MEMORY LOCATION

if (*laml* == NO)

report 3 Memory bank *i* is not implemented.

else

report 4 Memory bank *i* is implemented.

if (*laml* < 3 OR *laml* == 0xFF)

error 2 Wrong memory location for memory bank *i*. Actual: *laml*. Expected: [0x03,0xFE].

endif

answer = QUERY CONTENT DTR0 *// Check that DTR0 incremented after reading a valid memory bank location*

if (*answer* != 1)

error 3 DTR0 not incremented after reading location 0 of memory bank *i*. Actual: *answer*. Expected: 1.

endif

// Check location 0x02

DTR0 (2)

answer = READ MEMORY LOCATION

if (*answer* == NO)

error 4 No answer received when reading location 0x02 of memory bank *i*. Actual: NO. Expected: not NO.

endif

// Check that at least one location between 0x03 and laml is implemented.

numberOfAnswers = 0

DTR0 (3)

for (*j* = 3; *j* <= *laml*; *j*++)

answer = READ MEMORY LOCATION

if (*answer* != NO)

numberOfAnswers++

endif

answer = QUERY CONTENT DTR0

if (*answer* != *j* + 1)

error 5 DTR0 not incremented after reading location *j* of memory bank *i*. Actual: *answer*. Expected: *j* + 1.

endif

endfor

if (*numberOfAnswers* == 0)

error 6 At least one memory bank location of memory bank *i* must be implemented in the range [0x03, *laml*].

endif

// Read above last accessible memory bank location

DTR0 (*laml* + 1)

for (*j* = *laml* + 1; *j* <= 0xFE; *j*++)

answer = READ MEMORY LOCATION

if (*answer* != NO)

error 7 Answer received when reading the not implemented memory bank location *j* of memory bank *i*. Actual: *answer*. Expected: NO.

endif

answer = QUERY CONTENT DTR0

if (*answer* != *j* + 1)

error 8 DTR0 not incremented after reading above the last accessible memory location *j* of memory bank *i*. Actual: *answer*. Expected: *j* + 1.

endif

endfor

// Read the last memory bank location

DTR0 (0xFF)

answer = READ MEMORY LOCATION

if (*answer* != NO)

error 9 Answer received when reading location 0xFF of memory bank *i*. Actual: *answer*. Expected: NO.

endif

answer = QUERY CONTENT DTR0

if (*answer* != 255)

error 10 DTR0 modified after reading location 0xFF of memory bank *i*. Actual: *answer*. Expected: 255.

endif

endif

endfor

// Check that memory banks from lamb +1 untill 199 are not implemented - no reply expected when reading MB

DTR0 (0)

for (*i* = *lamb* + 1; *i* < 200; *i*++)

DTR1 (*i*)

answer = READ MEMORY LOCATION

if (*answer* != NO)

error 11 Answer received when reading above the last accessible memory bank: memory bank *i*. Actual: *answer*. Expected: NO.

endif

answer = QUERY CONTENT DTR0

if (*answer* != 0)

error 12 DTR0 modified after reading the not implemented memory bank *i*. Actual: *answer*. Expected: 0.

DTR0 (0)

endif

endfor

// Check that memory banks from 200 until 255 are reserved - no reply expected when reading MB

DTR0 (0)

for (*i* = 200; *i* < 256; *i*++)

DTR1 (*i*)

answer = READ MEMORY LOCATION

if (*answer* != NO)

error 13 Answer received when reading the reserved memory bank *i*. Actual: *answer*. Expected: NO.

endif

answer = QUERY CONTENT DTR0

if (*answer* != 0)

error 14 DTR0 modified after reading the reserved memory bank *i*. Actual: *answer*. Expected: 0.

DTR0 (0)

endif

endfor

endif

12.6.4 Memory bank writing

The test sequence checks the correct function of the WRITE MEMORY LOCATION and WRITE MEMORY LOCATION - NO REPLY commands. Before proceeding with the test, an implemented memory bank is needed.

Test sequence shall be run for each selected logical unit.

Test description:

(*memoryBankNr*; *memoryBankLoc*) = **FindImplementedMemoryBank** ()

if (*memoryBankNr* != 0)

report 1 Memory bank *memoryBankNr* is implemented and will be used for testing.

DTR0 (*memoryBankNr*)

RESET MEMORY BANK

wait 11 s

DTR0 (3)

DTR1 (*memoryBankNr*)

loc0x03 = READ MEMORY LOCATION

if (*memoryBankNr* == 1)

if (*memoryBankLoc* <= 253)

iArray = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17}

else

iArray = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14}

endif

else

if (*memoryBankLoc* <= 253)

iArray = {0,1,2,3,4,5,6,7,8,9,10,11,15,16,17}

else

iArray = {0,1,2,3,4,5,6,7,8,9,10,11}

endif

endif

foreach (*i* in *iArray*)

DTR0 (2) *// Select memory bank location*

DTR1 (*memoryBankNr*) *// Select memory bank*

if (*i* != 1 AND *i* != 3 AND *i* != 5)

ENABLE WRITE MEMORY *// ENABLE WRITE MEMORY for certain steps*

endif

command1[*i*] *// WRITE MEMORY LOCATION - NO REPLY for certain steps*

command2[*i*] *// DTR0 for certain steps*

answer = command3[*i*] *// WRITE MEMORY LOCATION with or without reply*

if (*answer* != *writeValue*[*i*])

error 1 Writing to valid memory bank location *text1*[*i*] at test step i = *i*. Actual: *answer*. Expected: *writeValue*[*i*].

endif

answer = QUERY CONTENT DTR0 *// Check if DTR0 changed after writing a valid memory bank location in different conditions*

if (*answer* != *dtrValue*[*i*])

error 2 DTR0 *text2*[*i*] at test step i = *i*. Actual: *answer*. Expected: *dtrValue*[*i*].

endif

answer = QUERY CONTENT DTR1 *// Check that DTR1 not changed after writing a memory bank location*

if (*answer* != *memoryBankNr*)

error 3 DTR1 modified at test step i = *i*. Actual: *answer*. Expected: *memoryBankNr*.

endif

DTR0 (*address*[*i*])

DTR1 (*memoryBankNr*)

answer = READ MEMORY LOCATION *// Check by reading if the location was correctly written - this will also disable the writeEnableState*

if (*answer* != *readValue*[*i*])

error 4 Wrong content of memory bank location at test step i = *i*. Actual: *answer*. Expected: *readValue*[*i*].

endif

endfor

else

report 2 No other memory bank besides memory bank 0 is implemented.

DTR1 (0) *// Select memory bank 0*

```
// Read and store all values written in memory bank 0
for (j = 0; j < 256; j++)
    DTR0 (j)
    valueOnLocation[j] = READ MEMORY LOCATION
endfor
// Try writing memory bank 0
for (j = 0; j < 2; j++)
    for (k = 0; k < 256; k++)
        ENABLE WRITE MEMORY
        DTR0 (k)
        if (valueOnLocation[k] == NO)
            value = 255
        else
            value = ~valueOnLocation[k]
        endif
        if (j == 0)
            answer = WRITE MEMORY LOCATION (value)
            if (answer != NO)
                error 5 Writing a ROM memory bank location confirmed at test step (j,k) =
                (j,k). Actual: answer. Expected: NO.
            endif
        else
            WRITE MEMORY LOCATION - NO REPLY (value)
        endif
        answer = QUERY CONTENT DTR0 // Check DTR0
        if (k == 255)
            if (answer != 255)
                error 6 DTR0 changed at test step (j,k) = (j,k). Actual: answer. Expected:
                255.
            endif
        else
            if (answer != k + 1)
                error 7 DTR0 not incremented at test step (j,k) = (j,k). Actual: answer.
                Expected: k + 1.
            endif
```

endif

answer = QUERY CONTENT DTR1 *// Check that DTR1 not changed after trying to write a memory bank location*

if (*answer* != 0)

error 8 DTR1 modified at test step (j,k) = (*j*,*k*). Actual: *answer*. Expected: 0.

DTR1 (0)

endif

DTR0 (*k*)

answer = READ MEMORY LOCATION *// Check by reading if location was written - this will also disable the writeEnableState*

if (*answer* != *valueOnLocation*[*k*])

error 9 Wrong content of memory bank location at test step (j,k) = (*j*,*k*). Actual: *answer*. Expected: *valueOnLocation*[*k*].

ENABLE WRITE MEMORY

DTR0 (*k*)

WRITE MEMORY LOCATION – NO REPLY (*valueOnLocation*[*k*])

endif

endfor

endfor

endif

Table 53 – Parameters for test sequence Memory bank writing

Test step i	command1	command2	command3	writeValue	text1	dtrValue	text2	address	readValue	test step descripton
0	-	-	WRITE MEMORY LOCATION (0x00)	0x00	not confirmed	3	not incremented	2	0x00	Check writing of a valid location when writeEnableState = enabled
1	-	-	WRITE MEMORY LOCATION (0x01)	NO	confirmed	2	incremented	2	0x00	Check writing of a valid location when writeEnableState = disabled
2	-	-	WRITE MEMORY LOCATION - NO REPLY (0x02)	NO	confirmed	3	not incremented	2	0x02	Check writing of a valid location when writeEnableState = enabled
3	-	-	WRITE MEMORY LOCATION - NO REPLY (0x03)	NO	confirmed	2	incremented	2	0x02	Check writing of a valid location when writeEnableState = disabled
4	-	-	DIRECT WRITE MEMORY (0x02, 0x04)	0x04	not confirmed	3	not incremented	2	0x04	Check writing of a valid location when writeEnableState = enabled
5	-	-	DIRECT WRITE MEMORY (0x02, 0x05)	NO	confirmed	2	incremented	2	0x04	Check writing of a valid location when writeEnableState = disabled
6	WRITE MEMORY LOCATION - NO REPLY (0x55)	DTR0 (255)	WRITE MEMORY LOCATION (0x06)	NO	confirmed	255	incremented	255	NO	Check writing when writeEnableState = enabled AND location is not implemented (loc0xFF-don't
7	WRITE MEMORY LOCATION - NO REPLY (0x55)	DTR0 (255)	WRITE MEMORY LOCATION - NO REPLY (0x07)	NO	confirmed	255	incremented	255	NO	

Test step i	command1	command2	command3	writeValue	text1	dtrValue	text2	address	readValue	test step descripton
8	WRITE MEMORY LOCATION - NO REPLY (0x55)	-	DIRECT WRITE MEMORY (0xFF, 0x08)	NO	confirmed	255	incremented	255	NO	increment DTR0)
9	WRITE MEMORY LOCATION - NO REPLY (0x55)	DTR0 (0)	WRITE MEMORY LOCATION (0x09)	NO	confirmed	1	not incremented	0	*memoryBank Loc*	Check writing when writeEnableState = enabled AND location is not writable (0x00)
10	WRITE MEMORY LOCATION - NO REPLY (0x55)	DTR0 (0)	WRITE MEMORY LOCATION - NO REPLY (0x0A)	NO	confirmed	1	not incremented	0	*memoryBank Loc*	
11	WRITE MEMORY LOCATION - NO REPLY (0x55)	-	DIRECT WRITE MEMORY (0x00, 0x0B)	NO	confirmed	1	not incremented	0	*memoryBank Loc*	
12	WRITE MEMORY LOCATION - NO REPLY (0x00)	DTR0 (3)	WRITE MEMORY LOCATION (0x0C)	NO	confirmed	4	not incremented	3	*loc0x03*	Check writing when writeEnableState = enabled AND location is lockable and MB is locked for writing
13	WRITE MEMORY LOCATION - NO REPLY (0x00)	DTR0 (3)	WRITE MEMORY LOCATION - NO REPLY (0x0D)	NO	confirmed	4	not incremented	3	*loc0x03*	
14	WRITE MEMORY LOCATION - NO REPLY (0x00)	-	DIRECT WRITE MEMORY (0x03, 0x0D)	NO	confirmed	4	not incremented	3	*loc0x03*	
15	WRITE MEMORY LOCATION - NO REPLY (0x55)	DTR0 (*memoryBankLoc*+1)	WRITE MEMORY LOCATION (0x0A)	NO	confirmed	*memoryBank Loc*+2	not incremented	*memoryBa nkLoc*+1	NO	Check writing when writeEnableState = enabled AND location is beyond the laml
16	WRITE MEMORY LOCATION - NO REPLY (0x55)	DTR0 (*memoryBankLoc*+1)	WRITE MEMORY LOCATION - NO REPLY (0x0B)	NO	confirmed	*memoryBank Loc*+2	not incremented	*memoryBa nkLoc*+1	NO	
17	WRITE MEMORY LOCATION - NO REPLY (0x55)	-	DIRECT WRITE MEMORY (*memoryBankLoc*+1, 0x0D)	NO	confirmed	*memoryBank Loc*+2	not incremented	*memoryBa nkLoc*+1	NO	

12.6.5 ENABLE WRITE MEMORY: writeEnableState

The test sequence checks the correct function of the ENABLE WRITE MEMORY command and as well as the correct implementation writeEnableState variable. Before proceeding with the test, an implemented memory bank is needed.

Test sequence shall be run for each selected logical unit.

Test description:

(*memBankNr*, *memoryBankLoc*) = **FindImplementedMemoryBank** ()

if (*memBankNr* == 0)

    report 1 No other memory bank besides memory bank 0 is implemented.

else

    report 2 Memory bank *memBankNr* is implemented and will be used for testing.

    for (*i* = 0; *i* < 15; *i*++)

        answer = QUERY DEVICE CAPABILITIES *// command should disable the writeEnableState*

        command1[*i*]

        command2[*i*]

        ENABLE WRITE MEMORY

        if (*i* >= 8)

            answer = *command3*[*i*]

            if (*answer* != *value1*[*i*])

                error 1 Wrong value at test step i = *i*. Actual: *answer*. Expected: *value1*[*i*].

            endif

        endif

        command4[*i*]

        answer = WRITE MEMORY LOCATION (*i*)

        if (*answer* != *value2*[*i*])

            error 2 Wrong value for writeEnableState at test step i = *i*. *text*[*i*].

        endif

    endfor

endif

Table 54 – Parameters for test sequence ENABLE WRITE MEMORY: writeEnableState

Test step i	command1	command2	command3	value1	command4	value2	text
0	DTR0 (2)	DTR1 (*memBankNr*)	-	-	-	0	Actual: DISABLED. Expected: ENABLED.
1	DTR0 (0)	DTR1 (*memBankNr*)	-	-	DTR0 (2)	1	Actual: DISABLED. Expected: ENABLED.
2	DTR0 (2)	DTR1 (0)	-	-	DTR1 (*memBankNr*)	2	Actual: DISABLED. Expected: ENABLED.
3	DTR0 (2)	DTR1 (*memBankNr*)	-	-	DTR2 (0)	3	Actual: DISABLED. Expected: ENABLED.
4	DTR0 (2)	DTR1 (*memBankNr*)	-	-	ENABLE WRITE MEMORY	4	Actual: DISABLED. Expected: ENABLED.
5	DTR0 (2)	DTR1 (*memBankNr*)	-	-	*answer* = QUERY DEVICE CAPABILITIES	NO	Actual: ENABLED. Expected: DISABLED.
6	DTR0 (2)	DTR1 (*memBankNr*)	-	-	RESET wait 300 ms	NO	Actual: ENABLED. Expected: DISABLED.
7	DTR0 (2)	DTR1 (*memBankNr*)	-	-	**PowerCycleAndWaitForDecoder** (5) DTR1 (*memBankNr*) DTR0 (2)	NO	Actual: ENABLED. Expected: DISABLED.
8	DTR0 (2)	DTR1 (*memBankNr*)	QUERY CONTENT DTR0	2	-	8	Actual: DISABLED. Expected: ENABLED.
9	DTR0 (2)	DTR1 (*memBankNr*)	QUERY CONTENT DTR1	*memBankNr*	-	9	Actual: DISABLED. Expected: ENABLED.
10	DTR0 (2)	DTR1 (*memBankNr*)	QUERY CONTENT DTR2	0	-	10	Actual: DISABLED. Expected: ENABLED.
11	DTR0 (2)	DTR1 (*memBankNr*)	READ MEMORY LOCATION	10	-	NO	Actual: ENABLED. Expected: DISABLED.
12	DTR0 (2)	DTR1 (*memBankNr*)	WRITE MEMORY LOCATION (80)	80	DTR0 (2)	12	Actual: DISABLED. Expected: ENABLED.
13	DTR0 (2)	DTR1 (*memBankNr*)	WRITE MEMORY LOCATION - NO REPLY (90)	NO	DTR0 (2)	13	Actual: DISABLED. Expected: ENABLED.
14	DTR0 (2)	DTR1 (*memBankNr*)	WRITE MEMORY LOCATION (100)	100	DTR0 (*memoryBankLoc* + 1)	NO	Actual: ENABLED. Expected: DISABLED.

12.6.6 ENABLE WRITE MEMORY: timeout / command in-between

The test sequence checks the correct function of the ENABLE WRITE MEMORY command. Before proceeding with the test, an implemented memory bank is needed. The command shall be executed only if it is received twice.

This test sequence checks the behaviour of DUT in the following conditions:

- one single command is sent instead of two identical commands;
- command is sent twice with a settling time of 105 ms which is longer than the defined settling time;
- command is sent with a frame in-between, frame which consists of few bits, but not a command;
- command is sent with another command in-between, command which is broadcast sent;
- command is sent with another command in-between, command which is sent to a certain group address;
- command is sent with another command in-between, command which is sent to a certain short address.

In all these cases, the command should not be executed. Where given, the command in-between should be accepted.

Test sequence shall be run for each selected logical unit.

Test description:

(*memBankNr*, *memoryBankLoc*) = **FindImplementedMemoryBank** ()

if (*memBankNr* == 0)

    report 1 No other memory bank besides memory bank 0 is implemented.

else

    report 2 Memory bank *memBankNr* is implemented and will be used for testing.

    for (*i* = 0; *i* < 3; *i*++)

        RESET

        wait 300 ms

        DTR0 (2)

        DTR1 (*memBankNr*)

        if (*i* == 0) *// Test send command once*

            ENABLE WRITE MEMORY, send once

        else if (*i* == 1) *// Test send command with timeout*

            ENABLE WRITE MEMORY, send once

            wait 105 ms *// settling time*

            ENABLE WRITE MEMORY, send once

        else *// Test send command with timeout followed by a new command*

ENABLE WRITE MEMORY, send once

wait 105 ms *// settling time*

ENABLE WRITE MEMORY, send once

wait 50 ms *// settling time*

ENABLE WRITE MEMORY, send once

endif

answer = WRITE MEMORY LOCATION (0x01)

if (*answer* != *value*[*i*])

error 1 writeEnableState *text*[*i*] at test step i = *i*. Actual: *answer*. Expected: *value*[*i*].

endif

endfor

for (*i* = 0; *i* < 5; *i*++)

answer2 == 0x55

RESET

wait 300 ms

DTR0 (2)

DTR1 (*memBankNr*)

// The following steps must be sent within 75 ms, counted from the last rise bit of first "ENABLE WRITE MEMORY, send once" command untill first fall bit of second "ENABLE WRITE MEMORY, send once" command

ENABLE WRITE MEMORY, send once

if (*i* == 0)

idle 13 ms + 110010 + idle 13 ms *// settling time: idle 13 ms and send a frame followed by 13 ms - Test send command with a frame in-between*

answer2 == NO

else if (*i* == 1)

answer2 == QUERY DEVICE GROUPS 0-7, send to device **Broadcast** (), **accept** No Answer *// Test send command with broadcast command in-between*

else if (*i* == 2)

answer2 == QUERY DEVICE GROUPS 0-7, send to device **GroupAddress** (0), **accept** No Answer *// Test send command with group command in-between - gearGroups0*

else if (*i* == 3)

answer2 == QUERY DEVICE GROUPS 0-7, send to device **ShortAddress** (*GLOBAL_currentUnderTestLogicalUnit*), **accept** No Answer *//Test send command with short command in-between*

else

answer2 == QUERY DEVICE GROUPS 0-7, send to device **ShortAddress** (63), **accept** No Answer *//Test send command with short command in-between*

endif

ENABLE WRITE MEMORY, send once

answer = WRITE MEMORY LOCATION (*i*)

if (*answer* != NO)

error 2 writeEnableState enabled at test step i = *i*. Actual: *answer*. Expected: NO.

endif

if (*i* == 1 OR *i* == 3)

if (*answer2* != 0x00)

error 3 Command in-between not executed. Actual: *answer2*. Expected: 0x00.

endif

else

if (*answer2* != NO)

error 4 Command in-between executed. Actual: *answer2*. Expected: NO.

endif

endif

endfor

endif

Table 55 – Parameters for test sequence ENABLE WRITE MEMORY: timeout / command in-between

Test step i	value	text
0	NO	enabled
1	NO	enabled
2	0x01	not enabled

12.6.7 RESET MEMORY BANK: timeout / command in-between

The test sequence checks the correct function of the RESET MEMORY BANK command. Before proceeding with the test, an implemented memory bank is needed. The command shall be executed only if it is received twice.

This test sequence checks the behaviour of DUT in the following conditions:

- one single command is sent instead of two identical commands;
- command is sent twice with a settling time of 105 ms which is longer than the defined settling time;
- command is sent with a frame in-between, frame which consists of few bits, but not a command;
- command is sent with another command in-between, command which is broadcast sent;

- command is sent with another command in-between, command which is sent to a certain group address;
- command is sent with another command in-between, command which is sent to a certain short address.

In all these cases, the command should not be executed. Where given, the command in-between should be accepted.

Test sequence shall be run for each selected logical unit.

Test description:

(*memBankNr*, *memoryBankLoc*) = **FindImplementedMemoryBank** ()

if (*memBankNr* == 0)

    report 1 No other memory bank besides memory bank 0 is implemented.

else

    report 2 Memory bank *memBankNr* is implemented and will be used for testing.

    // Check timeout behaviour

    for (*i* = 0; *i* < 3; *i*++)

        RESET

        wait 300 ms

        DTR0 (2)

        DTR1 (*memBankNr*)

        ENABLE WRITE MEMORY

        answer = WRITE MEMORY LOCATION (0x55)

        if (*answer* != 0x55)

            error 1 Wrong value written at test step i = *i*. Actual: *answer*. Expected: 0x55.

        endif

        DTR0 (*memBankNr*)

        if (*i* == 0) *// Test send command once*

            RESET MEMORY BANK, send once

        else if (*i* == 1) *// Test send command with timeout*

            RESET MEMORY BANK, send once

            wait 105 ms *// settling time*

            RESET MEMORY BANK, send once

        else *// Test send command with timeout followed by a new command*

            RESET MEMORY BANK, send once

            wait 105 ms *// settling time*

RESET MEMORY BANK, send once

wait 50 ms *// settling time*

RESET MEMORY BANK, send once

endif

wait 10,1 s

DTR0 (2)

answer = READ MEMORY LOCATION

if (*answer* != *value*[*i*])

error 2 Memory bank *memBank text*[*i*] at test step i = *i*. Actual: *answer*. Expected: *value*[*i*].

endif

endfor

// Check behaviour when a command is sent in-between the send twice

for (*i* = 0; *i* < 5; *i*++)

*answer*2 == 0x55

RESET

wait 300 ms

DTR0 (2)

DTR1 (*memBankNr*)

ENABLE WRITE MEMORY

answer = WRITE MEMORY LOCATION (*i*)

if (*answer* != *i*)

error 3 Wrong value written at test step i = *i*. Actual: *answer*. Expected: *i*.

endif

DTR0 (*memBankNr*)

// The following steps must be sent within 75 ms, counted from the last rise bit of first "RESET MEMORY BANK, send once" command untill first fall bit of second "RESET MEMORY BANK, send once" command

RESET MEMORY BANK, send once

if (*i* == 0)

idle 13 ms + 110010 + idle 13 ms *// settling time: idle 13 ms and send a frame followed by 13 ms - Test send command with a frame in-between*

*answer*2 == NO

else if (*i* == 1)

*answer*2 == QUERY DEVICE GROUPS 0-7, send to device **Broadcast** (), **accept** No Answer *// Test send command with broadcast command in-between*

else if (*i* == 2)

answer2 == QUERY DEVICE GROUPS 0-7, send to device **GroupAddress** (0), **accept** No Answer *// Test send command with group command in between - gearGroups0*

else if (*i* == 3)

answer2 == QUERY DEVICE GROUPS 0-7, send to device **ShortAddress** (*GLOBAL_currentUnderTestLogicalUnit*), **accept** No Answer *//Test send command with short command in between*

else

answer2 == QUERY DEVICE GROUPS 0-7, send to device **ShortAddress** (63), **accept** No Answer *//Test send command with short command in between*

endif

RESET MEMORY BANK, send once

wait 10,1 s

DTR0 (2)

answer = READ MEMORY LOCATION

if (*answer* != *i*)

error 4 Memory bank *memBankNr* reset at test step i = *i*. Actual: *answer*. Expected: *i*.

endif

if (*i* == 1 OR *i* == 3)

if (*answer2* != 0x00)

error 3 Command in-between not executed. Actual: *answer2*. Expected: 0x00.

endif

else

if (*answer2* != NO)

error 4 Command in-between executed. Actual: *answer2*. Expected: NO.

endif

endif

endfor

endif

Table 56 – Parameters for test sequence RESET MEMORY BANK: timeout / command in-between

Test step i	value	text
0	0x55	reset
1	0x55	reset
2	0xFF	not reset

12.6.8 RESET MEMORY BANK

The test sequence checks the correct function of the RESET MEMORY BANK. Before proceeding with the test, an implemented memory bank is needed.

Test sequence shall be run for each selected logical unit.

Test description:

(*memBankNr*[]; *memBankLoc*[]) = **FindAllImplementedMemoryBanks** ()

if (*memBankNr*[0] == 0)

report 1 No other memory bank besides memory bank 0 is implemented.

else

for (*i* = 0; *i* < 4; *i*++)

// Change lock byte of all implemented memory banks

ENABLE WRITE MEMORY

foreach (*memBank* in *memBankNr*)

DTR0 (2)

DTR1 (*memBank*)

if (*i* <= 1)

WRITE MEMORY LOCATION - NO REPLY (*i*)

else

WRITE MEMORY LOCATION - NO REPLY (0x55)

endif

endfor

// Reset memory bank

DTR0 (*dtr*[*i*])

RESET MEMORY BANK

wait 10,1 s

// Check if the reset of selected memory bank was executed

foreach (*memBank* in *memBankNr*)

DTR0 (2)

DTR1 (*memBank*)

answer = READ MEMORY LOCATION

if (*i* <= 1)

if (*answer* != *i*)

error 1 Memory bank *memBank* reset with memory bank locked for writing at test step i = *i*. Actual: *answer*. Expected: *i*.

endif

else if (*i* == 2)

if (*memBank* == *memBankNr*[0] AND *answer* != 0xFF)

error 2 Selected memory bank *memBank* not reset at test step i = *i*. Actual: *answer*. Expected: 0xFF.

else if (*memBank* != *memBankNr*[0] AND *answer* != 0x55)

error 3 Unselected memory bank *memBank* reset at test step i = *i*. Actual: *answer*. Expected: 0x55.

endif

else

if (*answer* != 0xFF)

error 4 Memory bank *memBank* not reset at test step i = *i*. Actual: *answer*. Expected: 0xFF.

endif

endif

endfor

endfor

endif

Table 57 – Parameters for test sequence RESET MEMORY BANK

Test step i	dtr	test description
0	*memBankNr*[0]	no ResetDevice (memory bank is locked)
1	0	no ResetDevice (memory bank is locked)
2	*memBankNr*[0]	reset the first memory bank; the other memory banks remain unchanged
3	0	reset all memory bank; all reset

12.7 Device Special commands

12.7.1 INITIALISE – timer

The test sequence checks the followings:

- the reset and power on values for initialisationState variable;
- initialisationState is DISABLED by RESET command;
- the correct function of the 15 min timer (starting, stopping and prolonging of the timer).

Test sequence shall be run for each selected logical unit.

Test description:

responsiveDevice = GLOBAL_currentUnderTestLogicalUnit

// Test reset value for initialisationState variable

TERMINATE

RESET

wait 300 ms

INITIALISE (*responsiveDevice*) *// initialisationState = ENABLED*

RESET

wait 300 ms

RANDOMISE

wait 100 ms

answer = **GetRandomAddress** ()

if (*answer* == 0xFF FF FF)

    error 1 initialisationState disabled by RESET command. Execution of RANDOMISE command expected after RESET command.

endif

TERMINATE

INITIALISE (*responsiveDevice*)

WITHDRAW *// initialisationState = WITHDRAWN*

RESET

wait 300 ms

RANDOMISE

wait 100 ms

answer = **GetRandomAddress** ()

if (*answer* == 0xFF FF FF)

    error 2 initialisationState disabled by RESET command. Execution of RANDOMISE command expected after RESET command.

endif

// Test power on value for initialisationState variable

TERMINATE

INITIALISE (*responsiveDevice*)

PowerCycleAndWaitForDecoder (5)

RANDOMISE

wait 100 ms

answer = **GetRandomAddress** ()

if (*answer* != 0xFF FF FF)

    error 3 Wrong power on value for initialisationState. No execution of RANDOMISE command expected after power cycle.

endif

TERMINATE

// Test that initialisationState is enabled by INITIALISE command, and that timer ends after 15 min

INITIALISE (*responsiveDevice*)

start_timer (*timer*)

answer = QUERY SHORT ADDRESS

if (*answer* != *responsiveDevice*)

    error 4 initialisationState not enabled. Actual: *answer*. Expected: *responsiveDevice*.

endif

do

    time = **get_timer** (*timer*)

    answer = QUERY SHORT ADDRESS

    if (*answer* != *responsiveDevice*)

        break

    endif

while (*time* < 17 min)

if (*time* < (15 - 1,5))

    error 5 Initialisation timer expires too early. Actual: *time* min. Expected: 13,5 min < timer < 16,5 min.

else if (*time* > (15 + 1,5))

    error 6 Initialisation timer expires too late. Actual: *time* min. Expected: 13,5 min < timer < 16,5 min.

endif

TERMINATE

// Test that timer is prolonged

INITIALISE (*responsiveDevice*)

start_timer (*timer*)

wait 5 min

INITIALISE (*responsiveDevice*)

do

time = **get_timer** (*timer*)

answer = QUERY SHORT ADDRESS

if (*answer* != *responsiveDevice*)

break

endif

while (*time* < 22 min)

if (*time* < ((15 + 5) - 1,5))

error 7 Re-triggered initialisation timer expires too early. Actual: *time* min. Expected: 18,5 min < timer < 21,5 min.

else if (*time* > ((15 + 5) + 1,5))

error 8 Re-triggered initialisation timer expires too late. Actual: *time* min. Expected: 18,5 min < timer < 21,5 min.

endif

TERMINATE

12.7.2 TERMINATE

Test sequence checks if TERMINATE command disables the initialisationState.

Test sequence shall be run for each selected logical unit.

Test description:

PowerCycleAndWaitForDecoder (5)

RESET

wait 300 ms

responsiveDevice = *GLOBAL_currentUnderTestLogicalUnit*

INITIALISE (*responsiveDevice*)

answer = QUERY SHORT ADDRESS

if (*answer* != *responsiveDevice*)

error 1 initialisationState not enabled. Actual: *answer*. Expected: *responsiveDevice*.

endif

TERMINATE

answer = QUERY SHORT ADDRESS

if (*answer* != NO)

error 2 initialisationState not disabled. Actual: *answer*. Expected: NO.

endif

12.7.3 INITIALISE - device addressing

The test sequence checks the device addressing scheme defined in the standard, in two cases: when DUT has no short address and when DUT has a short address assigned.

Test sequence shall be run for each selected logical unit.

Test description:

RESET

wait 300 ms

oldAddress = GLOBAL_currentUnderTestLogicalUnit

newAddress = 63

for (*i* = 0; *i* < 2; *i*++)

SetShortAddress (*fromAddress*[*i*]; *toAddress*[*i*])

for (*j* = 0; *j* < 5; *j*++)

INITIALISE (*device*[*j*])

answer = QUERY SHORT ADDRESS

if (*answer* != *shortAddress*[*i*,*j*])

error 1 Wrong responsive logical unit at test step (i,j) = (*i*,*j*). Actual: *answer*. Expected: *shortAddress*[*i*,*j*].

endif

TERMINATE

endfor

endfor

SetShortAddress (*newAddress*; *oldAddress*)

Table 58 – Parameters for test sequence INITIALISE - device addressing

Test step i	fromAddress	toAddress
0	*oldAddress*	MASK
1	MASK	*newAddress*

Test step j	device	*GLOBAL_numberOf LogicalUnits*	shortAddress		test step description
			i = 0 (no shortAddress)	i = 1 (shortAddress = newAddress)	
0	MASK	1	MASK	*newAddress*	All logical units react
		>1	invalid backward frame	invalid backward frame	
1	0111 1111b		MASK	NO	Only logical units without shortAddress react
2	*newAddress*		NO	*newAddress*	Only logical units with shortAddress = *newAddress* react
3	1011 1111b		NO	NO	No logical unit reacts
4	1100 0000b		NO	NO	No logical unit reacts

12.7.4 RANDOMISE

The test sequence checks the correct function of the RANDOMISE command when initialisationState has one of the following values: disabled, enabled or withdrawn.

Test sequence shall be run for each selected logical unit.

Test description:

RESET

wait 300 ms

TERMINATE

RANDOMISE

wait 100 ms

randomAddress = **GetRandomAddress** ()

if (*randomAddress* != 0xFF FF FF)

> **error 1** Command executed when initialisationState is disabled. Actual: *randomAddress*. Expected: 0xFF FF FF.

endif

RESET

wait 300 ms

responsiveDevice = GLOBAL_currentUnderTestLogicalUnit

INITIALISE (*responsiveDevice*)

RANDOMISE

wait 100 ms

randomAddress1 = **GetRandomAddress** ()

if (*randomAddress1* == 0xFFFFFF)

error 2 Command not executed when initialisationState is enabled. Generated random address is 0xFF FF FF.

endif

SetSearchAddress (*randomAddress1*)

WITHDRAW

RANDOMISE

wait 100 ms

randomAddress2 = **GetRandomAddress** ()

if (*randomAddress2* == *randomAddress1*)

error 3 Command not executed when initialisationState is withdrawn. No new random address generated.

endif

TERMINATE

12.7.5 COMPARE

The test sequence checks the correct function of the COMPARE command when initialisationState is either disabled or enabled. Test also checks whenever DUT should send an answer to COMPARE command, depending on the randomAddress and searchAddress stored in DUT.

Test sequence shall be run for each selected logical unit.

Test description:

RESET

wait 300 ms

TERMINATE

responsiveDevice = GLOBAL_currentUnderTestLogicalUnit

answer = COMPARE

if (*answer* != NO)

error 1 Command executed when initialisationState is disabled and randomAddress = searchAddress = 0xFF FF FF. Actual: *answer*. Expected: NO.

endif

INITIALISE (*responsiveDevice*)

answer = COMPARE

if (*answer* != YES)

error 2 Command not executed when initialisationState is enabled and randomAddress = searchAddress = 0xFF FF FF. Actual: *answer*. Expected: YES.

endif

randomAddress = **GetLimitedRandomAddress** (*responsiveDevice*)

if (*randomAddress* == 0xFF FF FF)

error 3 Bad random generator. For testing purpose each byte of the random address must be in [0x01, 0xFE] range.

else

INITIALISE (*responsiveDevice*)

for (*i* = 0; *i* < 7; *i*++)

SetSearchAddress (*data*[*i*])

answer = COMPARE

if (*answer* != *value*[*i*])

error 4 *errorText*[*i*] at test step i = *i*. Actual: *answer*. Expected: *value*[*i*].

endif

endfor

TERMINATE

answer = COMPARE

if (*answer* != NO)

error 5 Command executed when initialisationState is disabled and randomAddress = searchAddress and different from 0xFF FF FF. Actual: *answer*. Expected: NO.

endif

endif

Table 59 – Parameters for test sequence COMPARE

Test step i	data	value	errorText
0	*randomAddress* + 0x01 00 00	YES	Command not executed at RANDOM ADDRESS < SEARCH ADDRESS
1	*randomAddress* + 0x00 01 00	YES	Command not executed at RANDOM ADDRESS < SEARCH ADDRESS
2	*randomAddress* + 0x00 00 01	YES	Command not executed at RANDOM ADDRESS < SEARCH ADDRESS
3	*randomAddress* – 0x01 00 00	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS

Test step i	data	value	errorText
4	randomAddress = 0x00 01 00	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
5	randomAddress = 0x00 00 01	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
6	randomAddress	YES	Command not executed at RANDOM ADDRESS = SEARCH ADDRESS

12.7.6 WITHDRAW

The test sequence checks the correct function of the WITHDRAW command. Test also checks that the INITIALISE command should not restart the compare process and should not prolong the initialisation timer.

Test sequence shall be run for each selected logical unit.

Test description:

RESET

wait 300 ms

responsiveDevice = GLOBAL_currentUnderTestLogicalUnit

randomAddress = **GetLimitedRandomAddress** (*responsiveDevice*)

if (*randomAddress* == 0xFF FF FF)

    error 1 Bad random generator. For testing purpose each byte of the random address must be in [0x01, 0xFE] range.

else

    for (*i* = 0; *i* < 7; *i*++)

        TERMINATE

        INITIALISE (*responsiveDevice*)

        SetSearchAddress (*data*[*i*])

        WITHDRAW

        SetSearchAddress (*randomAddress*)

        answer = COMPARE

        if (*answer* != *value*[*i*])

            error 2 *errorText*[*i*] at test step i = *i*. Actual: *answer*. Expected: *value*[*i*].

        endif

    endfor

    INITIALISE (*responsiveDevice*)

    answer = COMPARE

    if (*answer* != NO)

        error 3 INITIALISE resets initialisationState to ENABLED. Actual: *answer*. Expected: NO.

    endif

TERMINATE

endif

// Test that the initialisationState timer is re-triggered when initialisationState = WITHDRAWN state

RESET

wait 300 ms

INITIALISE (*responsiveDevice*)

start_timer (*timer*)

wait 5 min

WITHDRAW

INITIALISE (*responsiveDevice*)

do

time = **get_timer** (*timer*)

answer = QUERY SHORT ADDRESS

if (*answer* != *responsiveDevice*)

break

endif

while (*time* < 22 min)

if (*time* < ((15 + 5) - 1,5) OR *time* > ((15 + 5) + 1,5))

error 4 Initialisation timer not re-triggering while initialisationState is withdrawn. Actual: *time* min. Expected: 18,5 min < timer < 21,5 min.

endif

TERMINATE

Table 60 – Parameters for test sequence WITHDRAW

Test step i	data	value	errorText	initialisationState after WITHDRAW
0	*randomAddress* + 0x01 00 00	YES	Command executed at RANDOM ADDRESS < SEARCH ADDRESS	ENABLED
1	*randomAddress* + 0x00 01 00	YES	Command executed at RANDOM ADDRESS < SEARCH ADDRESS	ENABLED
2	*randomAddress* + 0x00 00 01	YES	Command executed at RANDOM ADDRESS < SEARCH ADDRESS	ENABLED
3	*randomAddress* - 0x01 00 00	YES	Command executed at RANDOM ADDRESS > SEARCH ADDRESS	ENABLED
4	*randomAddress* - 0x00 01 00	YES	Command executed at RANDOM ADDRESS > SEARCH ADDRESS	ENABLED
5	*randomAddress* - 0x00 00 01	YES	Command executed at RANDOM ADDRESS > SEARCH ADDRESS	ENABLED

Test step i	data	value	errorText	initialisationState after WITHDRAW
6	*randomAddress*	NO	Command not executed at RANDOM ADDRESS − SEARCH ADDRESS	WITHDRAWN

12.7.7 SEARCHADDRH / SEARCHADDRM / SEARCHADDRL

The test sequence checks first the reset and power on values for searchAddress variable. After that, the correct function of the SEARCHADDRH, SEARCHADDRM, SEARCHADDRL commands is checked when initialisationState is disabled, enabled or withdrawn.

Test sequence shall be run for each selected logical unit.

Test description:

// Test reset value for searchAddress variable

responsiveDevice = GLOBAL_currentUnderTestLogicalUnit

RESET

wait 300 ms

INITIALISE (*responsiveDevice*)

SetSearchAddress (0x01 01 01)

answer = QUERY SHORT ADDRESS

if (*answer* != NO)

> **error 1** Wrong reset value for randomAddress. No answer expected from QUERY SHORT ADDRESS after setting the searchAddress. Actual: *answer*. Expected: NO

endif

RESET

wait 300 ms

answer = QUERY SHORT ADDRESS

if (*answer* != *responsiveDevice*)

> **error 2** Wrong reset value for searchAddress. Answer expected from QUERY SHORT ADDRESS after resetting the variables. Actual: *answer*. Expected: *responsiveDevice*

endif

TERMINATE

// Test power on value for searchAddress variable

SetSearchAddress (0x01 01 01)

PowerCycleAndWaitForDecoder (5)

INITIALISE (*responsiveDevice*)

answer = QUERY SHORT ADDRESS

if (*answer* != *responsiveDevice*)

error 3 Wrong power on value for searchAddress. Answer expected from QUERY SHORT ADDRESS after power on cycle. Actual: *answer.* Expected: *responsiveDevice*

endif

// Test whether searchAddress variable is correctly set

RESET

wait 300 ms

randomAddress = **GetLimitedRandomAddress** (*responsiveDevice*)

if (*randomAddress* == 0xFF FF FF)

error 4 Bad random generator. For testing purpose each byte of the random address must be in [0x01, 0xFE] range.

else

answer = COMPARE

if (*answer* != NO)

error 5 COMPARE executed when initialisationState was disabled. Actual: *answer.* Expected: NO.

endif

INITIALISE (*responsiveDevice*)

SetSearchAddress (*randomAddress* + 0x00 01 00)

answer = COMPARE

if (*answer* != YES)

error 6 searchAddress not set when initialisationState was enabled. Actual: *answer.* Expected: YES.

endif

SetSearchAddress (*randomAddress*)

WITHDRAW

SetSearchAddress (*randomAddress* + 0x01 00 00)

TERMINATE

INITIALISE (*responsiveDevice*)

answer = COMPARE

if (*answer* != YES)

error 7 searchAddress not set when initialisationState was withdrawn. Actual: *answer.* Expected: YES.

endif

TERMINATE

endif

12.7.8 PROGRAM SHORT ADDRESS

The test sequence checks the correct function of the PROGRAM SHORT ADDRESS command when initialisationState is disabled, enabled or withdrawn. Test also checks whenever command is accepted or not when different formats (valid and invalid) of the address to be stored are given.

Test sequence shall be run for each selected logical unit.

Test description:

RESET

wait 300 ms

TERMINATE

features = QUERY DEVICE CAPABILITIES, send to device **ShortAddress** (*newAddress*)

oldAddress = GLOBAL_currentUnderTestLogicalUnit

newAddress = 63

PROGRAM SHORT ADDRESS (*newAddress*)

answer = QUERY DEVICE CAPABILITIES, send to device **ShortAddress** (*newAddress*), **accept** No Answer

if (*answer* != NO)

    error 1 Command executed when initialisationState is disabled. Actual: *answer*. Expected: NO.

    SetShortAddress (*newAddress*; *oldAddress*)

endif

randomAddress = **GetLimitedRandomAddress** (*oldAddress*)

if (*randomAddress* == 0xFF FF FF)

    error 2 Bad random generator. For testing purpose each byte of the random address must be in [0x01, 0xFE] range.

else

    INITIALISE (*oldAddress*)

    for (*i* = 0; *i* < 7; *i*++)

        SetSearchAddress (*data*[*i*])

        PROGRAM SHORT ADDRESS (*newAddress*)

        answer = QUERY DEVICE CAPABILITIES, sent to (**ShortAddress** (*newAddress*)) , **accept** No Answer

        if (*answer* != *value*[*i*])

            error 3 *errorText1*[*i*] at test step i = *i*. Actual: *answer*. Expected: *value*[*i*].

            if (i != 6)

                SetShortAddress (*newAddress*; *oldAddress*)

            else

SetShortAddress (*oldAddress*; *newAddress*)

endif

endif

endfor

SetSearchAddress (*randomAddress*)

for (*j* = 0; *j* < 6; *j*++)

PROGRAM SHORT ADDRESS (*address[j]*)

answer = QUERY DEVICE CAPABILITIES, send to device **ShortAddres** (*queryAddress[j]*), **accept** No Answer

if (*answer* != *features*)

halt 1 PROGRAM SHORT ADDRESS command *errorText2[j]* at test step j = *j*. Actual: *answer*. Expected: YES.

endif

endfor

// Test for all available short addresses

for (*j = GLOBAL_numberShortAddresses*; *j* < 64; *j*++)

PROGRAM SHORT ADDRESS (*j*)

answer = QUERY SHORT ADDRESS, send to device **ShortAddress** (*j*)

if (*answer* != *j*)

halt 2 PROGRAM SHORT ADDRESS command at test step j = *j* failed. Actual: *answer*. Expected: *j*.

endif

endfor

WITHDRAW

PROGRAM SHORT ADDRESS (*oldAddress*)

answer = QUERY DEVICE CAPABILITIES, send to device **ShortAddress** (*oldAddress*), **accept** No Answer

if (*answer* != *features*)

error 4 Command not executed when initialisationState is withdrawn. Actual: *answer*. Expected: YES.

SetShortAddress (63; *oldAddress*)

endif

TERMINATE

endif

Table 61 – Parameters for test sequence PROGRAM SHORT ADDRESS

Test step i	data	value	errorText1
0	*randomAddress* + 0x01 00 00	NO	Command executed at RANDOM ADDRESS < SEARCH ADDRESS
1	*randomAddress* + 0x00 01 00	NO	Command executed at RANDOM ADDRESS < SEARCH ADDRESS
2	*randomAddress* + 0x00 00 01	NO	Command executed at RANDOM ADDRESS < SEARCH ADDRESS
3	*randomAddress* - 0x01 00 00	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
4	*randomAddress* - 0x00 01 00	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
5	*randomAddress* - 0x00 00 01	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
6	*randomAddress*	*features*	Command not executed at RANDOM ADDRESS = SEARCH ADDRESS

Test step j	address	description	queryAddress	errorText2
0	*oldAddress*	initial address	*oldAddress*	not executed
1	0011 1111b	short address 63	63	not executed
2	0100 0000b	no change	63	executed
3	1000 0000b	no change	63	executed
4	1111 1110b	no change	63	executed
5	MASK	delete short address	broadcast unaddressed	not executed

12.7.9 VERIFY SHORT ADDRESS

The test sequence checks the correct function of the VERIFY SHORT ADDRESS command when initialisationState is disabled, enabled or withdrawn. Test also checks whenever answer is received from DUT when different formats (valid and invalid) of the address to be verified are given.

Test sequence shall be run for each selected logical unit.

Test description:

RESET

wait 300 ms

TERMINATE

oldAddress = GLOBAL_currentUnderTestLogicalUnit

answer = VERIFY SHORT ADDRESS ((*oldAddress* << 1) + 1)

if (*answer* != NO)

    error 1 Command executed when initialisationState is disabled. Actual: *answer*. Expected: NO.

endif

for (*i* = 0; *i* < 8; *i*++)

INITIALISE (*address*[*i*])

DTR0 (*setAddress*[*i*])

SET SHORT ADDRESS, send to device **ShortAddress** (*address*[*i*])

answer = VERIFY SHORT ADDRESS (*data*[*i*])

if (*answer* != *value*[*i*])

error 2 *errorText*[*i*] when initialisationState is enabled at test step i = *i*. Actual: *answer*. Expected: *value*[*i*].

endif

WITHDRAW

answer = VERIFY SHORT ADDRESS (*data*[*i*])

if (*answer* != *value*[*i*])

error 3 *errorText*[*i*] when initialisationState is withdrawn at test step i = *i*. Actual: *answer*. Expected: *value*[*i*].

endif

TERMINATE

endfor

SetShortAddress (255; *oldAddress*)

Table 62 – Parameters for test sequence VERIFY SHORT ADDRESS

Test step i	address	setAddress	data	value	errorText
0	*oldAddress*	*oldAddress*	*oldAddress*	YES	Command not executed
1	*oldAddress*	*oldAddress*	*oldAddress* + 1	NO	Command executed
2	*oldAddress*	63	63	YES	Command not executed
3	63	63	62	NO	Command executed
4	63	63	64	NO	Command executed
5	63	63	MASK	NO	Command executed
6	63	63	11111110b	NO	Command executed
7	63	MASK	MASK	NO	Command executed

12.7.10 QUERY SHORT ADDRESS

The test sequence checks the correct function of the QUERY SHORT ADDRESS command. Initially the correct format of short address returned by command is checked, and after that the behaviour of DUT when initialisationState is disabled, enabled or withdrawn is tested.

Test sequence shall be run for each selected logical unit.

Test description:

RESET

wait 300 ms

oldAddress = GLOBAL_currentUnderTestLogicalUnit

// Check answer format of QUERY SHORT ADDRESS, expected: 11111111b or 00AAAAAAb

INITIALISE (*oldAddress*)

for (*i* = 0; *i* < 3; *i*++)

DTR0 (*validAddress*[*i*])

SET SHORT ADDRESS, send to device **ShortAddress** (*address*[*i*])

answer = QUERY SHORT ADDRESS, **accept** Value

if (*answer* != *addressFormat*[*i*])

error 1 Wrong format of returned short address at test step i = *i*. Actual: *answer*. Expected format: *addressFormat*[*i*].

endif

endfor

TERMINATE

// Check behaviour of DUT with initialisationState = disabled

answer = QUERY SHORT ADDRESS

if (*answer* != NO)

error 2 Command executed when initialisationState is disabled. Actual: *answer*. Expected: NO.

endif

randomAddress = **GetLimitedRandomAddress** (63)

if (*randomAddress* == 0xFF FF FF)

error 3 Bad random generator. For testing purpose each byte of the random address must be in [0x01, 0xFE] range.

SetShortAddress (63; *oldAddress*)

else

// Check behaviour of DUT with initialisationState = enabled and different values for randomAddress and searchAddress

INITIALISE (63)

for (*j* = 0; *j* < 7; *j*++)

SetSearchAddress (*data*[*j*])

answer = QUERY SHORT ADDRESS

if (*answer* != *value*[*j*])

error 4 *errorText*[*j*] when initialisationState is enabled at test step j = *j*. Actual: *answer*. Expected: *value*[*j*].

endif

endfor

WITHDRAW

// Check behaviour of DUT with initialisationState = withdrawn and different values for randomAddress and searchAddress

for (*j* = 0; *j* < 7; *j*++)

SetSearchAddress (*data*[*j*])

answer = QUERY SHORT ADDRESS

if (*answer* != *value*[*j*])

error 5 *errorText*[*j*] when initialisationState is withdrawn at test step j = *j*. Actual: *answer*. Expected: *value*[*j*].

endif

endfor

TERMINATE

// Check answer of QUERY SHORT ADDRESS when DUT has no short address assigned

DTR0 (255)

SET SHORT ADDRESS, send to device **ShortAddress** (63) *// Delete short address*

INITIALISE (255)

SetSearchAddress (*randomAddress*)

answer = QUERY SHORT ADDRESS

if (*answer* != 255)

error 6 Wrong answer when no short address is assigned and initialisationState is enabled. Actual: *answer*. Expected: 255.

endif

WITHDRAW

answer = QUERY SHORT ADDRESS

if (*answer* != 255)

error 7 Wrong answer when no short address is assigned and initialisationState is withdrawn. Actual: *answer*. Expected: 255.

endif

TERMINATE

SetShortAddress (255; *oldAddress*)

endif

Table 63 – Parameters for test sequence QUERY SHORT ADDRESS

Test step i	validAddress	address	addressFormat
0	*oldAddress*	*oldAddress*	*oldAddress*
1	MASK	*oldAddress*	MASK

Test step i	validAddress	address	addressFormat
2	63	broadcast unaddressed	63

Test step j	data	value	errorText
0	*randomAddress* + 0x01 00 00	NO	Command executed at RANDOM ADDRESS < SEARCH ADDRESS
1	*randomAddress* + 0x00 01 00	NO	Command executed at RANDOM ADDRESS < SEARCH ADDRESS
2	*randomAddress* + 0x00 00 01	NO	Command executed at RANDOM ADDRESS < SEARCH ADDRESS
3	*randomAddress* - 0x01 00 00	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
4	*randomAddress* - 0x00 01 00	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
5	*randomAddress* - 0x00 00 01	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
6	*randomAddress*	63	Command not executed at RANDOM ADDRESS = SEARCH ADDRESS

12.7.11 IDENTIFY DEVICE

The test sequence checks the correct function of the IDENTIFY DEVICE command. The identification procedure timer is also checked if it is started, finished, prolonged, or aborted according to the specification.

Test sequence shall be run for each selected logical unit.

Test description:

RESET

wait 300 ms

responsiveDevice = *GLOBAL_currentUnderTestLogicalUnit*

// Test if identification procedure is started and finished within 10 s ± 1 s

UserInput (After accepting this message please check if logical unit starts an identification procedure and measure how long the identification procedure takes (in s), *OK*)

IDENTIFY DEVICE

wait 12 s

started = **UserInput** (Did identification procedure start?, *YesNo*)

if (*started* != Yes)

error 1 Identification procedure not started by IDENTIFY DEVICE command.

else

stopped = **UserInput** (Did identification procedure stop?, *YesNo*)

if (*stopped* == Yes)

stoppedTime = **UserInput** (Enter the length of identification procedure, *value* [s])

if (*stoppedTime* < 9)

error 2 Identification procedure stopped earlier than 9 s.

else if (*stoppedTime* > 11)

error 3 Identification procedure not stopped after 11 s.

endif

else

error 4 Identification procedure not stopped after 11 s.

UserInput (Wait untill identification procedure stops, *OK*)

endif

endif

// Test if identification procedure timer is prolonged

UserInput (After accepting this message please check if logical unit starts an identification procedure of 15 s, *OK*)

IDENTIFY DEVICE

wait 5 s

IDENTIFY DEVICE

wait 12 s

started = **UserInput** (Did logical unit start an identification procedure of 15 s ± 1 s?, *YesNo*)

if (*started* != Yes)

error 5 Identification procedure not restart on reception of a second IDENTIFY DEVICE command.

endif

// Test if identification procedure timer is not stopped

for (*i* = 0; *i* < 4; *i*++)

UserInput (After accepting this message please check if logical unit starts an identification procedure of 10 s ± 1 s, *OK*)

IDENTIFY DEVICE

wait 5 s

if (*i* < 2)

command1[*i*]

else

answer = *command1*[*j*]

if (*answer* != *value1*[*i*])

error 6 *command1*[*i*] not executed.

```
        endif
    endif
    wait 12 s
    started = UserInput (Did identification procedure last for 10 s ± 1 s?, YesNo)
    if (started != Yes)
        error 7 Identification procedure stopped on reception of text1[i].
    endif
    if (i == 1)
        answer = query1[i]
        if (answer != value1[i])
            error 8 command1[i] not executed.
        endif
        TERMINATE
    endif
endfor
// Test if identification procedure is aborted
DTR0 (1)
for (j = 0; j < 5; j++)
    command2[j]
    UserInput (After accepting this message logical unit will start an identification procedure. Please
    check if identification procedure is stopped after 4 s, OK)
    IDENTIFY DEVICE
    wait 4 s
    command3[j]
    stopped = UserInput (Did logical unit start an identification procedure of 4 s?, YesNo)
    if (stopped == NO)
        error 9 Identification procedure not stopped by command3[j].
        wait 8 s
    endif
    if (j >= 1)
        answer = query2[j]
        if (answer != value2[j])
            error 10 command3[j] not executed.
        endif
```

if *(j == 1)*

TERMINATE

endif

endif

endfor

Table 64 – Parameters for test sequence IDENTIFY DEVICE

Test step j	command1	query1	value1
0	PING	-	-
1	INITIALISE *(responsiveDevice)*	VERIFY SHORT ADDRESS *(responsiveDevice)*	YES
2	COMPARE	-	YES
3	QUERY DEVICE STATUS	-	0XX0 X000b

Test step j	Command 2	Command3	query2	value2
0	-	TERMINATE	-	-
1	INITIALISE *(responsiveDevice)*	WITHDRAW	COMPARE	NO
2	DTR1 (1) DTR2 (0)	ADD TO DEVICE GROUPS 0-15	QUERY DEVICE GROUPS 07	1
3	-	RESET **wait** 300 ms	QUERY DEVICE GROUPS 07	0
4	-	DTR0 (2)	QUERY CONTENT DTR0	2

12.8 Logical unit cross contamination

12.8.1 DTR0

Test sequence checks that the change of the DTR0 register on one logical unit will not affect the value of the registers of the other logical units. DTR0 register shall be tested via memory banks. After each read of a memory bank location DTR0 shall be incremented.

Test sequence shall be run for all logical units in parallel.

Test description:

if (*GLOBAL_numberShortAddresses* == 1)

report 1 Only one logical device implemented

else

DTR0 (10)

DTR1 (0)

answer = READ MEMORY LOCATION

for (*address = 0; address < GLOBAL_numberShortAddresses; address++*)

for (*k = 0; k < address; k++*)

READ MEMORY LOCATION, send to device **ShortAddress** (*address*)

endfor

endfor

for (*address = 0; address < GLOBAL_numberShortAddresses; address++*)

answer = QUERY CONTENT DTR0, send to device **ShortAddress** (*address*)

expectedValue = 10 + 1 + *address*

if (*answer != expectedValue*)

error 1 LogicalUnit *address*: Wrong value of DTR0. Actual: *answer*. Expected: *expectedValue*.

endif

endfor

endif

12.8.2 NVM variables

Test sequence checks that the change of some variables on one logical unit will not affect the values of the variables of the other logical units.

Test sequence shall be run for all logical units in parallel.

Test description:

if (*GLOBAL_numberShortAddresses == 1*)

report 1 Only one logical device implemented

else

ResetDevice ()

// Change variables on logical units

for (*address = 0; address < GLOBAL_numberShortAddresses; address++*)

AddDeviceGroups ((0x00000001 << (*address* % 32)), **ShortAddress** (*address*))

endfor

// Check change of variables

for (*address = 0; address < GLOBAL_numberShortAddresses; address++*)

answer = **GetDeviceGroups** (**ShortAddress** (*address*))

expected = (0x00000001 << (*address* % 32))

if (*answer != expected*)

error 1 LogicalUnit *address*: Wrong value for deviceGroups. Actual: *answer*. Expected: *expected*.

endif

endfor

endif

ResetDevice ()

12.8.3 Random address generation

Test sequence checks that:

- all logical units generate a unique random address, when RANDOMISE command is sent using broadcast as addressing mode;
- only the addressed logical unit generates a random address when RANDOMISE command is sent using the short address of the selected logical unit.

Test sequence shall be run for all logical units in parallel.

Test description:

if (*GLOBAL_numberShortAddresses* == 1)

report 1 Only one logical device implemented
else

ResetDevice ()

// All logical units shall generate an unique random address

INITIALISE (MASK)

RANDOMISE

wait 100 ms

for (*address* = 0; *address* < *GLOBAL_numberShortAddresses*; *address*++)

randomAddress[*address*] = **GetRandomAddress** (*address*)

endfor

for (*i* = 0; *i* < *GLOBAL_numberShortAddresses*; *i*++)

if (*randomAddress*[*i*] == 0xFFFFFF)

error 1 LogicalUnit *i*: Wrong random address generated. Actual: 0xFFFFFF. Expected: not 0xFFFFFF.

endif

for (*j* = *i* + 1; *j* < *GLOBAL_numberShortAddresses*; *j*++)

if (*randomAddress*[*i*] == *randomAddress*[*j*])

error 2 LogicalUnit *i* and LogicalUnit *j* generated the same random address *randomAddress*[*i*].

endif

endfor

endfor

ResetDevice ()

TERMINATE

// Only one logical unit should generate a random address

for (*address* = 0; *address* < *GLOBAL_numberShortAddresses*; *address*++)

answer = QUERY RESET STATE, send to device **ShortAddress** (*address*)

if (*answer* != YES)

error 3 LogicalUnit *address*: is not in the reset state. Actual: N0. Expected: YES.

endif

INITIALISE (*address*)

RANDOMISE

wait 100 ms

TERMINATE

answer = QUERY RESET STATE, send to device **ShortAddress** (*address*)

if (*answer* != NO)

error 4 LogicalUnit *address*: is still in the reset state. Actual: YES. Expected: N0.

endif

endfor

endif

12.8.4 Addressing 1

Test sequence checks the answer sent by all logical units at broadcast queries. Test sets the quiescent mode of half of the logical units to disabled. The expected answer is YES for a YES-NO query, and an invalid backward frame for an 8-bit query.

Test sequence shall be run for all logical units in parallel.

Test description:

if (*GLOBAL_numberShortAddresses* == 1)

report 1 Only one logical device implemented

else

ResetDevice (false)

START QUIESCENT MODE

// Quiescent mode of all logical units is on

answer = QUERY QUIESCENT MODE

if (*answer* != YES)

error 1 Wrong answer received from all logical units at a YES-NO query. Actual: *answer*. Expected: YES.

endif

answer = QUERY DEVICE STATUS

if (*answer* != 0100 0010b)

error 2 Wrong answer received from all logical units at an 8-bit query. Actual: *answer*. Expected: 0100 0010b.

endif

// Quiescent mode of one logical units is off, the rest are on

for (*address* = 0; *address* < *GLOBAL_numberShortAddresses*; *address*++)

START QUIESCENT MODE

STOP QUIESCENT MODE, send to device **ShortAddress** (*address*)

for (*i* = 0; *i* < *GLOBAL_numberShortAddresses*; *i*++)

answer = QUERY QUIESCENT MODE, send to device **ShortAddress** (*i*)

if (*i* == *address*)

if (*answer* != NO)

error 3 Wrong answer received from logical unit *address* at a YES-NO query. Actual: *answer*. Expected: NO.

endif

else

if (*answer* != YES)

error 4 Wrong answer received from logical unit *i* at a YES-NO query. Actual: *answer*. Expected: YES.

endif

endif

endfor

answer = QUERY QUIESCENT MODE

if (*answer* != YES)

error 5 Wrong answer received from all logical units at a YES-NO query. Actual: *answer*. Expected: YES.

endif

answer = QUERY DEVICE STATUS, **accept** Violation

if (*answer* == Violation)

error 6 Wrong answer received from all logical units at an 8-bit query. Actual: *answer*. Expected: Violation.

endif

endfor

// Quiescent mode of all logical units is off

STOP QUIESCENT MODE

answer = QUERY QUIESCENT MODE

if (*answer* != NO)

error 7 Wrong answer received from all logical units at a YES-NO query. Actual: *answer*. Expected: NO.

endif

answer = QUERY STATUS, **accept** Violation

if (*answer* != 0100 0000b)

error 8 Wrong answer received from all logical units at an 8-bit query. Actual: *answer*. Expected: 1000 000b.

endif

endif

12.8.5 Addressing 2

Test sequence checks the answer sent by all logical units at queries sent using broadcast and grouping addressing. Test checks the behaviour of the logical units when all of them are added to one group, and after that when half of them are in one group and the other half in a different group. Test sets the quiescent mode of the logical units as follows:

- all enabled;
- first half of the group disabled and the rest enabled, with all logical units in one group; Group0 quiescent mode disabled and Group1 enabled, with logical units split into two groups;
- first half of the group enabled and the rest disabled, with all logical units in one group; Group0 quiescent mode enabled and Group1 disabled with logical units split into two groups;
- all disabled.

Test shall be run for all logical units in parallel.

Test description:

if (*GLOBAL_numberShortAddresses* == 1)

report 1 Only one logical device implemented

else

ResetDevice (false)

for (*i* = 0; *i* < 2; *i*++)

if (*i* == 0)

AddDeviceGroups (0x00000002)

```
else
    for (address = 0; address < (GLOBAL_numberShortAddresses >> 1); address++)
        RemoveDeviceGroups (0x00000002, ShortAddress (address))
        AddDeviceGroups (0x00000001, ShortAddress (address))
    endfor
endif
for (j = 0; j < 4; j++)
    switch (j)
        case 0: // All quiescent mode enabled
            START QUIESCENT MODE
            break
        case 1:
            if (i == 0) // First half of the group - quiescent mode disable, the rest - enable
                for (address = 0; address < (GLOBAL_numberShortAddresses >> 1);
                address++)
                    STOP QUIESCENT MODE, send to device ShortAddress
                    (address)
                endfor
            else // Group0 - quiescent mode disabled, Group1 - enabled
                STOP QUIESCENT MODE, send to device GroupAddress (0)
            endif
            break
        case 2:
            if (i == 0) // First half of the group - quiescent mode enable, the rest - disable
                for (address = 0; address < (GLOBAL_numberShortAddresses >> 1);
                address++)
                    START QUIESCENT MODE, send to device ShortAddress
                    (address)
                endfor
                for (address = (GLOBAL_numberShortAddresses >> 1); address <
                GLOBAL_numberShortAddresses; address++)
                    STOP QUIESCENT MODE, send to device ShortAddress
                    (address)
                endfor
            else // Group0 - quiescent mode enable, Group1 - disable
                START QUIESCENT MODE, send to device GroupAddress (0)
```

STOP QUIESCENT MODE, send to device **GroupAddress** (1)

endif

break

case 3: *// All units quiescent mode disable*

if (*i* == 0)

STOP QUIESCENT MODE

else

STOP QUIESCENT MODE, send to device **GroupAddress** (0)

endif

break

endswitch

answer = QUERY QUIESCENT MODE

if (*answer* != *enabledBroadcast*[*j*])

error 1 Wrong answer received from all logical units at a YES-NO query at test step (i,j) = (*i*,*j*). Actual: *answer*. Expected: *lampOnBroadcast*[*j*].

endif

answer = QUERY STATUS, **accept** Violation

if (*answer* != *statusBroadcast*[*j*])

error 2 Wrong answer received from all logical units at an 8-bit query at test step (i,j) = (*i*,*j*). Actual: *answer*. Expected: *statusBroadcast*[*j*].

endif

answer = QUERY QUIESCENT MODE, send to device **GroupAddress** (1)

if (*answer* != *enabledG1*[*i*,*j*])

error 3 Wrong answer received from all logical units in gearGroups1 at a YES-NO query. Actual: *answer*. Expected: *lampOnG1*[*i*,*j*].

endif

answer = QUERY STATUS, send to device **GroupAddress** (1)

if (*answer* != *statusG1*[*i*,*j*])

error 4 Wrong answer received from all logical units in gearGroups1 at an 8-bit query at test step (i,j) = (*i*,*j*). Actual: *answer*. Expected: *statusG1*[*i*,*j*].

endif

if (*i* == 1)

answer = QUERY QUIESCENT MODE, send to device **GroupAddress** (0)

if (*answer* != *enabledG0*[*j*])

error 5 Wrong answer received from all logical units in gearGroups0 at a YES-NO query. Actual: *answer*. Expected: *lampOnG0*[*j*].

endif

answer = QUERY STATUS, send to device **GroupAddress** (0) *//gearGroups0*

if (*answer* != *statusG0[j]*)

error 6 Wrong answer received from all logical units in gearGroups0 at an 8-bit query at test step (i,j) = (*i,j*). Actual: *answer*. Expected: *statusG0[j]*.

endif

endif

endfor

endfor

endif

Table 65 – Parameters for test sequence Addressing 2

Test step j		0	1	2	3
enabledBroadcast		YES	YES	YES	NO
statusBroadcast		0100 0010b	Violation	Violation	0100 0000b
i = 0	enabledG1	YES	YES	YES	NO
	statusG1	0100 0010b	Violation	Violation	0100 0000b
i = 1	enabledG1	YES	YES	NO	NO
	statusG1	0100 0010b	0100 0010b	0100 0000b	0100 0000b
	enabledG0	YES	NO	YES	NO
	statusG0	0100 0010b	0100 0000b	0100 0010b	0100 0000b

12.8.6 Addressing 3

The test sequence checks the behaviour of a logical unit at a broadcast unaddressed query.

Test sequence shall be run for each selected logical unit.

Test description:

if (*GLOBAL_numberShortAddresses* == 1)
  report 1 Only one logical device implemented
else
  ResetDevice ()
  oldAddress = *GLOBAL_currentUnderTestLogicalUnit*
  SetShortAddress (*oldAddress*; 255)
  INITIALISE (MASK)
  RANDOMISE
  wait 100 ms
  answer = QUERY RANDOM ADDRESS(H), send to device **BroadcastUnaddress** (),
  accept Violation
  if (*answer* == Violation)
    error 1 Multiple logical units answered at broadcast unaddressed command.
  endif
  answer = QUERY RANDOM ADDRESS(M), send to device **BroadcastUnaddress** (),
  accept Violation
  if (*answer* == Violation)

error 2 Multiple logical units answered at broadcast unaddressed command.
endif
answer = QUERY RANDOM ADDRESS(L), send to device **BroadcastUnaddress** (),
accept Violation
if (*answer* == Violation)
error 3 Multiple logical units answered at broadcast unaddressed command.
endif
SetShortAddress (255; *oldAddress*)
TERMINATE
endif

12.9 Instance addressing

12.9.1 Instance Type Addressing

This test procedure checks for addressing an instance by its instance type. Therefore the test procedure checks for the number of implemented instances (QUERY NUMBER OF INSTANCES) and is reading each instance type (QUERY INSTANCE TYPE). In a second loop the sequence is checking all instance types (0 to 31 with QUERY INSTANCE TYPE), expecting a valid answer (no bit timing error) at each instance type number, detected through the first loop being implemented. All other instance type addressed query commands shall be without an answer.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice (true)

numberOfInstances = **GetNumberOfInstances** ()
numberOfInstanceTypes = 0
instanceTypeArray []

if (*numberOfInstances* == 0)
report 1 No instances available
else
// check each instance for its type and store it in a list
for (*i* = 0; *i* < *numberOfInstances*; *i*++)
instanceType = QUERY INSTANCE TYPE, send to instance **InstanceNumber** (*i*)
instanceTypeArray[*numberOfInstanceTypes*] = *instanceType*
numberOfInstanceTypes++
endfor

// check each possible instance type and check the answer according the array
for (*type* = 0; *type* < 32; *type*++)
instanceTypeCheck = QUERY INSTANCE TYPE, send to instance **InstanceType** (*type*)
accept no answer

//check if type is reported before and is part of the array
containsType = false
foreach(*atype* in *instanceTypeArray*)
if (*atype* == *type*)
containsType = true
break
endif
endfor

if (*instanceTypeCheck* == NO)
if (*containsType*)
error 1 No reponse from instance type *type*

```
            endif
        else
            if (containsType)
                if (instanceTypeCheck != type)
                    error 2 Invalid instance type from instance type type reported
                endif
            else
                error 3 Response from non-existent instance type type received
            endif
        endif
    endfor
endif

ResetDevice (false)
```

12.9.2 Instance Primary Group

This sequence sets the primary instance group (SET PRIMARY INSTANCE GROUP (DTR0)) one number after the other and checks for the group assigned by QUERY PRIMARY INSTANCE GROUP and reaction to an instance group addressed command (QUERY INSTANCE STATUS).

It also checks for DTR values above 31 to be ignored and MASK to remove any instance group assignment.

Test sequence shall be run for each selected logical unit.

Test description:

```
ResetDevice (true)

numberOfInstances = GetNumberOfInstances ()

if (numberOfInstances == 0)
    report 1 No instances available
else
    // clear all instance groups
    DTR0 (MASK)
    SET PRIMARY INSTANCE GROUP, send to instance InstanceBroadcast ()
    SET INSTANCE GROUP 1, send to instance InstanceBroadcast ()
    SET INSTANCE GROUP 2, send to instance InstanceBroadcast ()

    for (i = 0; i < numberOfInstances; i++)
    // check for valid instance group reaction
        for (setInstanceGroup = 0; setInstanceGroup < 32; setInstanceGroup++)
            DTR0 (setInstanceGroup)
            SET PRIMARY INSTANCE GROUP, send to instance InstanceNumber (i)
            answer = QUERY PRIMARY INSTANCE GROUP, send to instance
            InstanceNumber (i)
            if (answer != setGroup)
                error 1 Primary instance group is not set correctly to setInstanceGroup
            endif

            // check for no response if primary instance group is set to setInstanceGroup
            for (checkGroup = 0; checkGroup < 32, checkGroup++)
                answer = QUERY INSTANCE STATUS, send to instance InstanceGroup
                (checkGroup), accept no answer

                if (checkGroup == setInstanceGroup)
                    if (answer == NO)
                        error 2 No response from instance group checkGroup with
                        primary instance group set to setInstanceGroup
```

```
                endif
            else
                if (answer != NO)
                    error 3 Response received from instance group checkGroup with primary instance group set to setInstanceGroup
                endif
            endif
        endfor
    endfor

    // check if MASK is set properly for primary instance group
    DTR0 (MASK)
    SET PRIMARY INSTANCE GROUP, send to instance InstanceNumber (i)
    answer = QUERY PRIMARY INSTANCE GROUP, send to instance InstanceNumber (i)
    if (answer != MASK)
        error 4 Primary instance group is not set to MASK
    endif

    // check for no response if primary instance group is set to MASK
    for (checkGroup = 0; checkGroup < 32, checkGroup++)
        answer = QUERY INSTANCE STATUS, send to instance InstanceGroup (checkGroup), accept no answer
        if (answer != NO)
            error 5 Response received with primary instance group = MASK
        endif
    endfor

    // set primary instance group to 9 for the following check of invalid values
    DTR0 (9)
    SET PRIMARY INSTANCE GROUP, send to instance InstanceNumber (i)
    answer = QUERY PRIMARY INSTANCE GROUP, send to instance InstanceNumber (i)
    if (answer != 9)
        error 6 Primary instance group is not set to 9
    endif

    // primary instance group is 9 and should not change when group is set to an invalid value
    for (setInstanceGroup = 32; setInstanceGroup < 255; setInstanceGroup++)
        DTR0 (setInstanceGroup)
        SET PRIMARY INSTANCE GROUP, send to instance InstanceNumber (i)
        answer = QUERY PRIMARY INSTANCE GROUP, send to instance InstanceNumber (i)
        if (answer != 9)
            error 7 Primary instance group is not 9
        endif
    endfor
  endfor
endif

ResetDevice (false)
```

12.9.3 Instance Group 2

This sequences sets the instance group 2 (SET INSTANCE GROUP 2 (DTR0)) one number after the other and checks for the group assigned by QUERY INSTANCE GROUP 2 and reaction to an instance group addressed command (QUERY INSTANCE STATUS).

It also checks for DTR values above 31 to be ignored and MASK to remove any instance group assignment.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice (true)

numberOfInstances = **GetNumberOfInstances** ()

if (*numberOfInstances* == 0)
    report 1 No instances available
else
    // clear all instance groups
    DTR0 (MASK)
    SET PRIMARY INSTANCE GROUP, send to instance **InstanceBroadcast** ()
    SET INSTANCE GROUP 1, send to instance **InstanceBroadcast** ()
    SET INSTANCE GROUP 2, send to instance **InstanceBroadcast** ()

    for (*i* = 0; *i* < *numberOfInstances*; *i*++)
        // check for valid instance group reaction
        for (*setInstanceGroup* = 0; *setInstanceGroup* < 32; *setInstanceGroup*++)
            DTR0 (*setInstanceGroup*)
            SET INSTANCE GROUP 2, send to instance **InstanceNumber** (*i*)
            answer = QUERY INSTANCE GROUP 2, send to instance **InstanceNumber** (*i*)
            if (*answer* != *setGroup*)
                error 1 “instanceGroup2” is not set correctly to *setInstanceGroup*
            endif

            // check for no response if instance group 2 is set to setInstanceGroup
            for (*checkGroup* = 0; *checkGroup* < 32, *checkGroup*++)
                answer = QUERY INSTANCE STATUS, send to instance **InstanceGroup** (*checkGroup*), **accept** no answer

                if (*checkGroup* == *setInstanceGroup*)
                    if (*answer* == NO)
                        error 2 No response from instance group *checkGroup* with “instanceGroup2” set to *setInstanceGroup*
                    endif
                else
                    if (*answer* != NO)
                        error 3 Response received from instance group *checkGroup* with “instanceGroup2” set to *setInstanceGroup*
                    endif
                endif
            endfor
        endfor

        // check if MASK is set properly for instance group 2
        DTR0 (MASK)
        SET INSTANCE GROUP 2, send to instance **InstanceNumber** (*i*)
        answer = QUERY INSTANCE GROUP 2, send to instance **InstanceNumber** (*i*)
        if (*answer* != MASK)
            error 4 “instanceGroup2” is not set to MASK
        endif

        // check for no response if instance group 2 is set to MASK
        for (*checkGroup* = 0; *checkGroup* < 32, *checkGroup*++)
            answer = QUERY INSTANCE STATUS, send to instance **InstanceGroup** (*checkGroup*), **accept** no answer

            if (*answer* != NO)
                error 5 Response received with “instanceGroup2” = MASK

```
            endif
        endfor

        // set instance group 2 to 9 for the following check of invalid values
        DTR0 (9)
        SET INSTANCE GROUP 2, send to instance InstanceNumber (i)
        answer = QUERY INSTANCE GROUP 2, send to instance InstanceNumber (i)
        if (answer != 9)
            error 6 “instanceGroup2” is not set to 9
        endif

        // instance group 2 is 9 and should not change when group is set to an invalid value
        for (setInstanceGroup = 32; setInstanceGroup < 255; setInstanceGroup++)
            DTR0 (setInstanceGroup)
            SET INSTANCE GROUP 2, send to instance InstanceNumber (i)
            answer = QUERY INSTANCE GROUP 2, send to instance InstanceNumber (i)
            if (answer != 9)
                error 7 “instanceGroup2” is not 9
            endif
        endfor
    endfor
endif

ResetDevice (false)
```

12.9.4 Instance Group 1

This sequences sets the instance group 1 (SET INSTANCE GROUP 1 (DTR0)) one number after the other and checks for the group assigned by QUERY INSTANCE GROUP 1 and reaction to an instance group addressed command (QUERY INSTANCE STATUS).

It also checks for DTR values above 31 to be ignored and MASK to remove any instance group assignment.

Test sequence shall be run for each selected logical unit.

Test description:

```
ResetDevice (true)

numberOfInstances = GetNumberOfInstances ()

if (numberOfInstances == 0)
    report 1 No instances available
else
    // clear all instance groups
    DTR0 (MASK)
    SET PRIMARY INSTANCE GROUP, send to instance InstanceBroadcast ()
    SET INSTANCE GROUP 1, send to instance InstanceBroadcast ()
    SET INSTANCE GROUP 2, send to instance InstanceBroadcast ()

    for (i = 0; i < numberOfInstances; i++)
    // check for valid instance group reaction
        for (setInstanceGroup = 0; setInstanceGroup < 32; setInstanceGroup++)
            DTR0 (setInstanceGroup)
            SET INSTANCE GROUP 1, send to instance InstanceNumber (i)
            answer = QUERY INSTANCE GROUP 1, send to instance InstanceNumber (i)
            if (answer != setGroup)
                error 1 “instanceGroup1” is not set correctly to setInstanceGroup
            endif

            // check for no response if instance group 1 is set to setInstanceGroup
```

```
            for (checkGroup = 0; checkGroup < 32, checkGroup++)
                answer = QUERY INSTANCE STATUS, send to instance InstanceGroup
                (checkGroup), accept no answer

                if (checkGroup == setInstanceGroup)
                    if (answer == NO)
                        error 2 No response from instance group checkGroup with
                        “instanceGroup1” set to setInstanceGroup
                    endif
                else
                    if (answer != NO)
                        error 3 Response received from instance group checkGroup with
                        “instanceGroup1” set to setInstanceGroup
                    endif
                endif
            endfor
        endfor

        // check if MASK is set properly for instance group 1
        DTR0 (MASK)
        SET INSTANCE GROUP 1, send to instance InstanceNumber (i)
        answer = QUERY INSTANCE GROUP 1, send to instance InstanceNumber (i)
        if (answer != MASK)
            error 4 “instanceGroup1” is not set to MASK
        endif

        // check for no response if instance group 1 is set to MASK
        for (checkGroup = 0; checkGroup < 32, checkGroup++)
            answer = QUERY INSTANCE STATUS, send to instance InstanceGroup
            (checkGroup), accept no answer

            if (answer != NO)
                error 5 Response received with “instanceGroup1” = MASK
            endif
        endfor

        // set instance group 1 to 9 for the following check of invalid values
        DTR0 (9)
        SET INSTANCE GROUP 1, send to instance InstanceNumber (i)
        answer = QUERY INSTANCE GROUP 1, send to instance InstanceNumber (i)
        if (answer != 9)
            error 6 “instanceGroup1” is not set to 9
        endif

        // instance group 1 is 9 and should not change when group is set to an invalid value
        for (setInstanceGroup = 32; setInstanceGroup < 255; setInstanceGroup++)
            DTR0 (setInstanceGroup)
            SET INSTANCE GROUP 1, send to instance InstanceNumber (i)
            answer = QUERY INSTANCE GROUP 1, send to instance InstanceNumber (i)
            if (answer != 9)
                error 7 “instanceGroup1” is not 9
            endif
        endfor
    endfor
endif

ResetDevice (false)
```

12.9.5 Instance Group Combinations

This sequence checks for multiple instance group assignments. For this, the instance group 1 (SET INSTANCE GROUP 1 (DTR0)) and instance group 2 (SET INSTANCE GROUP 2

(DTR0)) are set to certain group numbers and the sequence then sets the primary instance group (SET PRIMARY INSTANCE GROUP (DTR0)) one number after the other and checks for the reaction to an instance group addressed command (QUERY INSTANCE STATUS), being executed according to the current group assignments.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice (true)

numberOfInstances = **GetNumberOfInstances** ()

if (*numberOfInstances* == 0)
    report 1 No instances available
else
    // clear all instance groups
    DTR0 (MASK)
    SET PRIMARY INSTANCE GROUP, send to instance **InstanceBroadcast** ()
    SET INSTANCE GROUP 1, send to instance **InstanceBroadcast** ()
    SET INSTANCE GROUP 2, send to instance **InstanceBroadcast** ()

    for (*i* = 0; *i* < *numberOfInstances*; *i*++)
        // check when all group slots are the same group
        DTR0 (5)
        SET PRIMARY INSTANCE GROUP, send to instance **InstanceNumber** (*i*)
        SET INSTANCE GROUP 1, send to instance **InstanceNumber** (*i*)
        SET INSTANCE GROUP 2, send to instance **InstanceNumber** (*i*)

        status = QUERY INSTANCE STATUS, send to instance **InstanceGroup** (5), **accept** no answer
        if (*status* == NO)
            error 1 Instance *i* does not respond to the instance group 5
        endif

        // check when only two group slots are the same group
        DTR0 (255)
        SET PRIMARY INSTANCE GROUP, send to instance **InstanceNumber** (*i*)
        DTR0 (6)
        SET INSTANCE GROUP 1, send to instance **InstanceNumber** (*i*)
        SET INSTANCE GROUP 2, send to instance **InstanceNumber** (*i*)

        status = QUERY INSTANCE STATUS, send to instance **InstanceGroup** (6), **accept** no answer
        if (*status* == NO)
            error 2 Instance *i* does not respond to the instance group 6
        endif

        // check when only two group slots are the same group
        DTR0 (7)
        SET PRIMARY INSTANCE GROUP, send to instance **InstanceNumber** (*i*)
        SET INSTANCE GROUP 1, send to instance **InstanceNumber** (*i*)
        DTR0 (255)
        SET INSTANCE GROUP 2, send to instance **InstanceNumber** (*i*)

        status = QUERY INSTANCE STATUS, send to instance **InstanceGroup** (7), **accept** no answer
        if (*status* == NO)
            error 3 Instance *i* does not respond to the instance group 7
        endif

```
        // check when only two group slots are the same group
        DTR0 (8)
        SET PRIMARY INSTANCE GROUP, send to instance InstanceNumber (i)
        SET INSTANCE GROUP 2, send to instance InstanceNumber (i)
        DTR0 (255)
        SET INSTANCE GROUP 1, send to instance InstanceNumber (i)

        status = QUERY INSTANCE STATUS, send to instance InstanceGroup (8), accept no answer
        if (status == NO)
            error 4 Instance i does not respond to the instance group 8
        endif
    endfor
endif

ResetDevice (false)
```

12.9.6 Multiple Instances Answer

If there is more than one instance implemented, this test sequence is checking for correct answer when more than one instance is queried. In a first step all event priorities are set to the same value through an instance broadcast. Afterwards the priority is queried also through broadcast and the answer shall be a correct frame displaying the event priority set.

Afterwards the first instance event priority is set to another value than the others and the instances are queried again through a broadcast. Here the answer shall be a frame, containing a bit timing error.

Test sequence shall be run for each selected logical unit.

Test description:

```
ResetDevice (true)

numberOfInstances = GetNumberOfInstances ()

if (numberOfInstances < 2)
    report 1 No multiple instances available
else
    // clear all instance groups
    DTR0 (MASK)
    SET PRIMARY INSTANCE GROUP, send to instance InstanceBroadcast ()
    SET INSTANCE GROUP 1, send to instance InstanceBroadcast ()
    SET INSTANCE GROUP 2, send to instance InstanceBroadcast ()

    // set all instances to instance group 3
    DTR0 (3)
    SET PRIMARY INSTANCE GROUP, send to instance InstanceBroadcast ()
    answer = QUERY PRIMARY INSTANCE GROUP, send to instance InstanceBroadcast ()
    if (answer != 3)
        error 1 Timing violation or no answer answer of multiple instances
    endif

    // set one instances (one before last) to instance group 5 => expect error when broadcast
    DTR0 (5)
    SET PRIMARY INSTANCE GROUP, send to instance InstanceNumber (numberOfInstances-2)
    answer = QUERY PRIMARY INSTANCE GROUP, send to instance InstanceGroup (3)
    if (answer != 3)
        error 2 Wrong answer of instance group 3
    endif
    answer = QUERY PRIMARY INSTANCE GROUP, send to instance InstanceGroup (5)
```

if (*answer* != 5)
error 3 Wrong answer of instance group 5
endif
answer = QUERY PRIMARY INSTANCE GROUP, send to instance **InstanceBroadcast** (), **accept** violation
if (*answer* != ERROR)
error 4 Wrong answer with different return values
endif
endif

ResetDevice (false)

12.10 Instance configuration instructions

12.10.1 Instance Enable/Disable

This sequence first tests if the device has instances implemented (QUERY INSTANCE NUMBER). If any instances are implemented, they are disabled (DISABLE INSTANCE) and enabled (ENABLE INSTANCE) again, one after each other and the enable status (QUERY INSTANCE STATUS Bit1 and QUERY INSTANCE ENABLED) will be observed.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice (true)

numberOfInstances = **GetNumberOfInstances** ()

if (*numberOfInstances* == 0)
report 1 No instances available

// check if input control is not enabled after enabling
ENABLE INSTANCE, send to instance **InstanceBroadcast** ()
enabled = QUERY INSTANCE ENABLED, send to instance **InstanceBroadcast** ()
if (*enabled*)
error 1 Wrong answer without any existent instance.
endif

// check if input control is not enabled after disabling
DISABLE INSTANCE, send to instance **InstanceBroadcast** ()
enabled = QUERY INSTANCE ENABLED, send to instance **InstanceBroadcast** ()
if (*enabled*)
error 2 Wrong answer without any existent instance.
endif
else
// check if all instances are enabled as preparation for next tests
// at this point all instances should be enabled
enable = QUERY INSTANCE ENABLED, send to instance **InstanceBroadcast** ()
if (!*enabled*)
error 3 Instances are not enabled.
endif
for (*i* = 0; *i* < *numberOfInstances*; *i*++)
enabled = QUERY INSTANCE ENABLED, send to instance **InstanceNumber** (*i*)
if (!*enabled*)
error 4 Instances *i* is not enabled.
endif

status = QUERY INSTANCE STATUS, send to instance **InstanceNumber** (*i*)
if (*status* != XXXX XX1Xb)
error 5 Instance *i* is not enabled.
endif

```
endfor

// disable each instance one by one and do a full check after each disable
for (i = 0; i < numberOfInstances; i++)
    // at this point at least one instance should be enabled
    enable = QUERY INSTANCE ENABLED, send to instance InstanceBroadcast ()
    if (!enabled)
        error 6 Instances are not enabled.
    endif

    DISABLE INSTANCE, send to instance InstanceNumber (i)

    for (checkInstance = 0; checkInstance < numberOfInstances; checkInstance++)
        enabled = QUERY INSTANCE ENABLED, send to instance InstanceNumber
        (checkInstance)
        status = QUERY INSTANCE STATUS, send to instance InstanceNumber
        (checkInstance)

        if (checkInstance <= i)
            // instance should be disabled
            if (enabled)
                error 7 Instance checkInstance is enabled.
            endif

            if (status != XXXX XX0Xb)
                error 8 Instance checkInstance is enabled .
            endif
        else
            // instance should be enabled
            if (!enabled)
                error 9 Instance checkInstance is not enabled.
            endif

            if (status != XXXX XX1Xb)
                error 10 Instance checkInstance is not enabled.
            endif
        endif
    endfor
endfor

// at this point no instance should be enabled
enable = QUERY INSTANCE ENABLED, send to instance InstanceBroadcast ()
if (enabled)
    error 11 Instances are enabled.
endif

// enable each instance one by one and do a full check after each enable
for (i = 0; i < numberOfInstances; i++)
    ENABLE INSTANCE, send to instance InstanceNumber (i)

    // at this point at least one instance should be enabled
    enable = QUERY INSTANCE ENABLED, send to instance InstanceBroadcast ()
    if (!enabled)
        error 12 Instances are not enabled.
    endif

    for (checkInstance = 0; checkInstance < numberOfInstances; checkInstance++)
        enabled = QUERY INSTANCE ENABLED, send to instance InstanceNumber
        (checkInstance)
        status = QUERY INSTANCE STATUS, send to instance InstanceNumber
        (checkInstance)
```

```
            if (checkInstance <= i)
                // instance should be enabled
                if (!enabled)
                    error 13 Instance checkInstance is not enabled.
                endif

                if (status != XXXX XX1Xb)
                    error 14 Instance checkInstance is not enabled.
                endif
            else
                // instance should be disabled
                if (enabled)
                    error 15 Instance checkInstance is enabled.
                endif

                if (status != XXXX XX0Xb)
                    error 16 Instance checkInstance is enabled.
                endif
            endif
        endfor
    endfor

    // check if all instances are disabled
    DISABLE INSTANCE, send to instance InstanceBroadcast ()
    enable = QUERY INSTANCE ENABLED, send to instance InstanceBroadcast ()
    if (enabled)
        error 17 Instances are enabled.
    endif
    for (i = 0; i < numberOfInstances; i++)
        enabled = QUERY INSTANCE ENABLED, send to instance InstanceNumber (i)
        if (enabled)
            error 18 Instances i is enabled.
        endif

        status = QUERY INSTANCE STATUS, send to instance InstanceNumber (i)
        if (status != XXXX XX0Xb)
            error 19 Instance i is enabled.
        endif
    endfor

    // check if all instances are enabled
    ENABLE INSTANCE, send to instance InstanceBroadcast ()
    enable = QUERY INSTANCE ENABLED, send to instance InstanceBroadcast ()
    if (!enabled)
        error 20 Instances are not enabled.
    endif
    for (i = 0; i < numberOfInstances; i++)
        enabled = QUERY INSTANCE ENABLED, send to instance InstanceNumber (i)
        if (!enabled)
            error 21 Instances i is not enabled.
        endif

        status = QUERY INSTANCE STATUS, send to instance InstanceNumber (i)
        if (status != XXXX XX1Xb)
            error 22 Instance i is not enabled.
        endif
    endfor
endif

ResetDevice (false)
```

12.10.2 Event Scheme

This sequence operates in three steps.

In the first step different event schemes (1 to 4, 0) are set for all instances one after another. The DUT is configured in a way that all requirements for allowing the event scheme to be set are met. All instance event schemes are verified through a QUERY EVENT SCHEME, also checking that the event schemes of other instances' remain unchanged.

For the second step the configuration of the DUT is changed in a way that the requirements for allowing the event schemes other than 0 are not met. For all instances the event schemes 1 to 4 are set and the fall back to event scheme 0 is verified.

In the third step the DUT meets the requirements for certain event schemes at the moment when there are set. The configuration is changed later in a way that the requirements are not met any longer and the implicit fall back to event scheme 0 is verified.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice (true)

numberOfInstances = **GetNumberOfInstances** ()

if (*numberOfInstances* == 0)
    report 1 No instances available
else
    // store current short address for later restore
    oldDeviceShortAddress = GLOBAL_currentDeviceShortAddress

    // TEST CASE: set without fallback + go to fallback after setting (when removing address)

    // DUT has already a short address
    // Add to device group 0
    AddDeviceGroups(0x0000 0001)

    // set primary instance group of all instances
    DTR0 (0)
    SET PRIMARY INSTANCE GROUP, send to instance **InstanceBroadcast** ()

    // loop for all instances and test setting and querying event scheme –
    for (*i* = 0; *i* < *numberOfInstances*; *i*++)
        // set event scheme to default test value
        DTR0 (4)
        SET EVENT SCHEME, send to instance **InstanceBroadcast** ()
        answer = QUERY EVENT SCHEME, send to instance **InstanceBroadcast** ()
        if (*answer* != 4)
            error 1 Not all instances have set their event scheme to 4.
        endif

        for (*dtrValue* = 0; *dtrValue* < 256; *dtrValue* ++)
            DTR0 (*dtrValue*)
            SET EVENT SCHEME, send to instance **InstanceNumber** (*i*)

            // cross check with all instances
            for (*checkInstance* = 0; *checkInstance* = *numberOfInstances*; *checkInstance*++)
                answer = QUERY EVENT SCHEME, send to instance **InstanceNumber** (*checkInstance*)

```
if (checkInstance == i)
    // check instance which event scheme is set to DTR0
    if (dtrValue >= 0 AND dtrValue <= 4)
        if (answer != dtrValue)
            error 2 Instance checkInstance has changed the event scheme to answer
        endif

        // if event scheme is set between 1 and 4 then check fallback when removing address or groups
        if (dtrValue >= 1 AND dtrValue <= 4)

            if (dtrValue == 1 OR dtrValue == 2)
                // remove group assignments
                RemoveDeviceGroups(0x0000 0001)

                // remove primary instance group
                DTR0 (255)
                SET PRIMARY INSTANCE GROUP, send to instance InstanceNumber (i)

                // check if event scheme stays same
                answer = QUERY EVENT SCHEME, send to instance InstanceNumber (checkInstance)
                if (answer != dtrValue)
                    error 3 Instance checkInstance has already changed event scheme to answer before removing device short address
                endif

                // remove DUTs short address
                DTR0 (255)
                SET SHORT ADDRESS

                // check if event scheme is fallback
                answer = QUERY EVENT SCHEME (device broadcast unaddressed, instance number checkInstance)
                if (answer != 0)
                    error 4 Instance checkInstance has not changed to fallback event scheme after removing device short address
                endif
            else if (dtrValue == 3)
                // remove primary instance group
                DTR0 (255)
                SET PRIMARY INSTANCE GROUP, send to instance InstanceNumber (i)

                // remove DUTs short address
                DTR0 (255)
                SET SHORT ADDRESS

                // check if event scheme stays same
                answer = QUERY EVENT SCHEME (device broadcast unaddressed, instance number checkInstance)
                if (answer != dtrValue)
                    error 5 Instance checkInstance has already changed event scheme to answer before removing group assignment
                endif
```

```
                    // remove group assignments
                    RemoveDeviceGroups(0x0000 0001,
                    BroadcastUnaddressed ())

                    // check if event scheme is fallback
                    answer = QUERY EVENT SCHEME (device broadcast
                    unaddressed, instance number checkInstance)
                    if (answer != 0)
                        error 6 Instance checkInstance has not changed to
                        fall back event scheme after removing group
                        assignment
                else if (dtrValue == 4)
                    // remove group assignments
                    RemoveDeviceGroups(0x0000 0001)

                    // remove DUTs short address
                    DTR0 (255)
                    SET SHORT ADDRESS

                    // check if event scheme stays same
                    answer = QUERY EVENT SCHEME (device broadcast
                    unaddressed, instance number checkInstance)
                    if (answer != dtrValue)
                        error 7 Instance checkInstance has already
                        changed event scheme to answer before removing
                        primary instance group
                    endif

                    // remove primary instance group
                    DTR0 (255)
                    SET PRIMARY INSTANCE GROUP (device broadcast
                    unaddressed, instance number i)

                    // check if event scheme is fallback
                    answer = QUERY EVENT SCHEME (device broadcast
                    unaddressed, instance number checkInstance)
                    if (answer != 0)
                        error 8 Instance checkInstance has not changed to
                        fallback event scheme after removing primary
                        instance group
                    endif
                endif

                // restore DUTs short address
                DTR0 (oldDeviceShortAddress)
                SET SHORT ADDRESS (device broadcast unaddressed)

                // restore group assignment
                AddDeviceGroups(0x0000 0001)

                // restore primary instance group
                DTR0 (0)
                SET PRIMARY INSTANCE GROUP, send to instance
                InstanceNumber (i)

                // restore event scheme
                DTR0 (dtrValue)
                SET EVENT SCHEME, send to instance InstanceNumber (i)
            endif
        else
            // check instance where DTR0 not change event scheme
            if (answer != 4)
```

```
                    error 9 Instance checkInstance has changed the event scheme to answer
                endif
            endif
        else
            // check instance which event scheme should not changed
            if (answer != 4)
                error 10 Instance checkInstance has changed the event scheme to answer
            endif
        endif
    endfor
endfor
endfor

// ------------------------------------------------
// TEST CASE: go to fallback when setting
// ------------------------------------------------

// clear all group assignments
ClearAllDeviceGroups ()

// remove primary instance group of all instances
DTR0 (255)
SET PRIMARY INSTANCE GROUP, send to instance InstanceBroadcast ()

// remove DUTs short address
DTR0 (255)
SET SHORT ADDRESS

// loop for all instances and test setting and querying event scheme
for (i = 0; i < numberOfInstances; i++)
    // set event scheme to default test value
    DTR0 (0)
    SET EVENT SCHEME (device broadcast unaddressed, instance broadcast)
    answer = QUERY EVENT SCHEME (device broadcast unaddressed, instance broadcast)
    if (answer != 0)
        error 11 Not all instances have set their event scheme to 0.
    endif

    for (dtrValue = 0; dtrValue < 256; dtrValue ++)
        DTR0 (dtrValue)
        SET EVENT SCHEME (device broadcast unaddressed, instance number i)

        // cross check with all instances
        for (checkInstance = 0; checkInstance = numberOfInstances; checkInstance++)
            answer = QUERY EVENT SCHEME (device broadcast unaddressed, instance number checkInstance)

            // in this test case the event scheme should be always set to 0
            if (answer != 0)
                error 12 Instance checkInstance has changed the event scheme to answer
            endif
        endfor
    endfor
endfor

// restore DUTs short address
DTR0 (oldDeviceShortAddress)
SET SHORT ADDRESS (device broadcast unaddressed)
```

endif

ResetDevice (false)

12.10.3 Input Resolution & Input Value

This test sequence reads the resolution for all implemented instances. The results are put into the test report. This test sequence is checking the readout of an input value (QUERY INPUT VALUE, QUERY INPUT VALUE LATCH) respective the resolution of the instance (QUERY RESOLUTION). The test performs N+1 read actions, the first command is QUERY INPUT VALUE and all following commands (if applicable) are QUERY INPUT VALUE LATCH with N = round up (resolution / 8). All commands up to N shall have an answer containing a value, the last one (N+1) shall have no answer.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice (true)

numberOfInstances = **GetNumberOfInstances** ()

if (*numberOfInstances* == 0)
    report 1 No instances available
else
    // get resolution and check number of input value queries
    for (*i* = 0; *i* < *numberOfInstances*; *i*++)
        resolution = QUERY RESOLUTION, send to instance **InstanceNumber** (*i*)
        NBytesNeeded = **RoundUp** (*resolution* / 8)
        report 2 Instance *i* has a resolution of *resolution* bit(s) / *NBytesNeeded* byte(s)

        answer = QUERY INPUT VALUE, send to instance **InstanceNumber** (*i*), **accept** no answer
        if (*answer* == NO)
            error 1 No response for input value byte
        endif

        for (*b* = 1; *b* < *NBytesNeeded*; *b*++)
            answer = QUERY INPUT VALUE LATCH, send to instance **InstanceNumber** (*i*)
            if (***answer*** == NO)
                error 2 No response for input value latch byte ***b***
            endif
        endfor

        answer = QUERY INPUT VALUE LATCH
        if (*answer* != NO)
            error 3 Response for input value latch after last byte
        endif
    endfor
endif

ResetDevice (false)

12.10.4 Event Filter

This test procedure sets for all instances with instance type 0 (event filter is 24 bit wide), one after another, the event filter to a certain test value. The event filter is checked by the corresponding query command.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice ()
numberOfInstances = **GetNumberOfInstances** ()
if (*numberOfInstances* == 0)
    report 1 No instances available
else
    // loop for all instances and test setting and querying event filter
    for (*i* = 0; *i* < *numberOfInstances*; *i*++)
        answer = QUERY INSTANCE TYPE, send to instance **InstanceNumber** (*i*)
        if (*answer* == 0)
            SetEventFilter (send to instance **InstanceNumber** (*i*), 0x010203)
            answer = **GetEventFilter** (send to instance **InstanceNumber** (*i*))
            if (*answer* != 0x010203)
                error 1 Instance event filter not correctly set and queried afterwards. Actual: *answer*. Expected: 0x010203.
            endif
        else
            report 1 Instance type is not 0. The width of the event filter is redefined by part 3 *answer* of this standard.
        endif
    endfor
endif
ResetDevice ()

12.11 Instance queries

12.11.1 Instance Number and Types

This sequence reads first the number of implemented instances (QUERY NUMBER OF INSTANCES). This number is then crosschecked by reading each instance type (QUERY INSTANCE TYPE) and put the result into the test sequence report. This sequence implicitly tests the instance number addressing. Also the instance numbers of not implemented instances (number of instances up to 31) are checked to result in no answer.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice (true)
numberOfInstances = **GetNumberOfInstances** ()
// check each possible instance even if they do not exist
for (*instance* = 0; *instance* < 32; *instance*++)
    answer = QUERY INSTANCE TYPE, send to instance **InstanceNumber** (*instance*),
    accept no answer
    if (*instance* < *numberOfInstances*)
        // expect answer of instance i
        if (*answer* == NO)
            error 1 Instance *instance* does not report its type
        else
            report 1 Instance *instance* is from type *answer*
        endif
    else
        // expect no answer of instance i
        if (*answer* != NO)
            error 2 Instance *instance* reports type *answer* but instance should not exist
        else
            report 2 Instance *instance* does not exist
        endif
    endif

endfor
ResetDevice (false)

12.11.2 Instance Status

This sequence checks the unused bits of QUERY INSTANCE STATUS answer. The error bit is not checked, as there is no definition of a specific error to be tested. The instance active bit is tested by the procedure Instance Enable / Disable.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice (true)

numberOfInstances = **GetNumberOfInstances** ()

if (*numberOfInstances* == 0)
    report 1 No instances available
else
    //check reserved bits
    for (*i* = 0; *i* < *numberOfInstances*; *i*++)
        status = QUERY INSTANCE STATUS, send to instance **InstanceNumber** (*i*)
        if (*status* != 0000 00XX)
            error 1 Reserved bits of instance *i* status are not 0
        endif
    endfor
endif

ResetDevice (false)

12.11.3 Instance Error

This test sequence checks that the instance error information is the same from QUERY INSTANCE ERROR and Bit0 of QUERY INSTANCE STATUS. If there is no error reported in the status byte (QUERY INSTANCE STATUS Bit0 equals zero), also the query instance error shall show no answer. If there is an error reported in the status byte, the query instance error shall show up with some value (specified in Parts 3xx).

There is a warning, if an error state is detected, as it is strongly recommended to resolve the error before conducting the test.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice (true)

numberOfInstances = **GetNumberOfInstances** ()

if (*numberOfInstances* == 0)
    report 1 No instances available

    // check if input control has errors
    error = QUERY INSTANCE ERROR, send to instance **InstanceBroadcast** ()
    if (*error*)
        error 1 Wrong answer without any existent instances

```
    endif
else
    // check if instances has errors
    for (i = 0; i < numberOfInstances; i++)
        error = QUERY INSTANCE ERROR, send to instance InstanceNumber (i)
        status = QUERY INSTANCE STATUS, send to instance InstanceNumber (i)
        if (status == XXXX XXX1b AND error != NO)
            error 2 Instance i has an error code error and report is consistent. Please
            resolve error before conducting this test.
        else if ((status == XXXX XXX0b AND error != NO) OR (status == XXXX XXX1b AND
        error == NO))
            error 3 Instance i has an error code error but report is inconsistent
        endif
    endfor
endif

ResetDevice (false)
```

12.12 Instance cross contamination

12.12.1 Instance Event Priority

Through this sequence different event priorities are set (send twice SET EVENT PRIORITY) for one instance after the other. The priority set is checked through QUERY EVENT PRIORITY. There is also a crosscheck that other instances' event priorities remain unchanged.

Test sequence shall be run for each selected logical unit.

Test description:

```
ResetDevice (true)

numberOfInstances = GetNumberOfInstances ()

if (numberOfInstances == 0)
    report 1 No instances available
else
    // set priority to default test value (to the lowest one)
    DTR0 (5)
    SET EVENT PRIORITY, send to instance InstanceBroadcast ()
    answer = QUERY EVENT PRIORITY, send to instance InstanceBroadcast ()
    if (answer != 5)
        error 1 Not all instances have set their event priority to level 5
    endif

    for (i = 0; i < numberOfInstances; i++)
        for (dtrValue = 0; dtrValue < 256; dtrValue++)
            DTR0 (dtrValue)
            SET EVENT PRIORITY, send to instance InstanceNumber (i)

            // cross check with all instances
            for (checkInstance = 0; checkInstance = numberOfInstances; checkInstance++)
                answer = QUERY EVENT PRIORITY, send to instance InstanceNumber
                (checkInstance)

                if (checkInstance == i)
                    // check instance which event priority is set by DTR0
                    if (dtrValue >= 2 AND dtrValue <= 5)
                        if (answer != dtrValue)
```

```
                        error 2 Instance checkInstance has changed the event priority to the wrong level answer
                    endif
                else
                    // check instance where DTR0 not change event priority
                    if (answer != 5)
                        error 3 Instance checkInstance has changed the event priority to the wrong level answer
                    endif
                endif
            else
                // check instance which event priority should not changed
                if (answer != 5)
                    error 4 Instance checkInstance has changed the event priority to the wrong level answer
                endif
            endif
        endfor
    endfor
endif

ResetDevice (false)
```

12.13 Reserved Commands

12.13.1 Reserved standard device commands

The test sequence checks whether device reacts on one of the reserved standard commands.

Test sequence shall be run for all logical units in parallel.

Test description:

```
ResetDevice (false)

for (i = 0; i <= 3; i++)

    for (opcode = fromCMD[i]; opcode <= toCMD[i]; opcode ++)

        answer = Send twice device broadcast command with opcode opcode, accept Violation, Value

        if (answer != NO)

            error 1 Answer after receiving reserved standard command with opcode opcode.

        endif

        answer = QUERY RESET STATE

        if (answer != YES)

            error 2 DUT not in reset state after receiving reserved standard command with opcode opcode.

            ResetDevice ()

        endif

    endfor

endfor
```

ResetDevice (true)

Table 66 – Parameters for test sequence Reserved commands: standard device commands

Test step i	fromCMD	toCMD
0	0x02	0x0F
1	0x12	0x13
2	0x22	0x2F
3	0x49	0xFF

12.13.2 Reserved instance commands (instance type 0)

The test sequence checks whether device reacts on one of the reserved instance type 0 instance commands.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice ()

numberOfInstances = **GetNumberOfInstances** ()
if (*numberOfInstances* == 0)
    report 1 No instances available
else
    for (*instance* = 0; *instance* < *numberOfInstances*; *instance* ++)
        answer = QUERY INSTANCE TYPE, send to **InstanceNumber** (*instance*)

        if (*answer* == 0)

            for (*i* = 0; *i* <= 4; *i*++)

                for (*opcode* = *fromCMD*[*i*]; *opcode* <= *toCMD*[*i*]; *opcode* ++)

                    answer = Send twice, send to **InstanceNumber** (*instance*) command with opcode *opcode*, **accept** Violation, Value

                    if (*answer* != NO)

                        error 1 Answer after receiving reserved instance command (instance type 0) with opcode *opcode*.

                    endif

                    answer = QUERY RESET STATE

                    if (*answer* != YES)

                        error 2 DUT not in reset state after receiving reserved instance command (instance type 0) with opcode *opcode*.

                        ResetDevice ()

                    endif

                endfor

endfor

endif

endfor

endif

Table 67 – Parameters for test sequence Reserved instance commands (instance type 0)

Test step i	fromCMD	toCMD
0	0x00	0x60
1	0x69	0x7F
2	0x85	0x85
3	0x87	0x87
4	0x93	0xFF

12.13.3 Reserved special commands

The test sequence checks whether device reacts on one of the reserved special commands.

Test sequence shall be run for each selected logical unit.

Test description:

ResetDevice ()

for (*i* = 0; *i* <= 14; *i*++)

for (*specialCMD = fromCMD*[*i*]; *specialCMD <= toCMD*[*i*]; *specialCMD* ++)

for (*k* = 0; *k* <= 14; *k*++)

answer = Send twice special command *specialCMD* with data *opcode*[*k*], **accept** Violation, Value

if (*answer* != NO)

error 1 Answer after receiving reserved special command *specialCMD* with data *opcode*[*k*].

endif

answer = QUERY RESET STATE

if (*answer* != YES)

error 2 DUT not in reset state after receiving reserved special command *specialCMD* with data *opcode*[*k*].

ResetDevice ()

endif

endif

endfor

endfor

Table 68 – Parameters for test sequence Reserved special commands

Test step i	fromCMD	toCMD
0	0xC10B	0xC11F
1	0xC122	0xC12F
2	0xC134	0xC1FF
3	0xC300	0xC3FF
4	0xCB00	0xBAFF
5	0xCD00	0xCDFF
6	0xCF00	0xCFFF
7	0xD100	0xD1FF
8	0xD300	0xD3FF
9	0xD500	0xD5FF
10	0xD700	0xD7FF
11	0xD900	0xD9FF
12	0xDB00	0xDBFF
13	0xDD00	0xDDFF
14	0xDF00	0xDFFF

Test step k	0	1	2	3	4	5	6	7	8	9
opcodeByte	0	1	3	5	9	17	33	65	129	255

12.14 General subsequences

12.14.1 Reset Device

This sequence is sending a RESET command and afterwards is waiting for the maximum time period allowed for the command to be executed.

Test description:

ResetDevice ()

RESET
wait 300ms
return

This sequence is sending a RESET command and optionally enables or disables the application controller and all instances.

Test description:

ResetDevice (*enable*)

ResetDevice ()
// enable or disable the application controller and all instances
if (*enable*)
    EnableApplicationControllerAndAllInstances ()
else
    DisableApplicationControllerAndAllInstances ()
endif
return

12.14.2 EnableApplicationControllerAndAllInstances

This sequence enables the application controller and all instances.

Test description:

EnableApplicationControllerAndAllInstances ()

ENABLE APPLICATION CONTROLLER
ENABLE INSTANCE, send to instance **InstanceBroadcast** ()
return

12.14.3 DisableApplicationControllerAndAllInstances

This sequence disables the application controller and all instances.

Test description:

DisableApplicationControllerAndAllInstances ()

DISABLE APPLICATION CONTROLLER
DISABLE INSTANCE, send to instance **InstanceBroadcast** ()
return

12.14.4 HasApplicationController

This test procedure returns, if the DUT contains an application controller or not.

Test description:

result = **HasApplicationController** ()

features = QUERY DEVICE CAPABILITIES
// check application controller
if (*features* == XXXX XXX1b)
    return true
else
    return false
endif

12.14.5 GetVersionNumber

Test subsequence returns the version number of the control gear.

Test description:

versionNumber = **GetVersionNumber** ()

versionNumber = -1

answer = QUERY VERSION NUMBER, **accept** Value

if (*answer* > 3)

minor = answer & 0x03

major = answer >> 2

versionNumber = major + *minor* / 10

else

versionNumber = answer

endif

return *versionNumber*

12.14.6 AddDeviceGroups

This test procedure sets one or more device groups according to the given mask.

Test description:

AddDeviceGroups (*mask*)

DTR2:DTR1 ((*mask* >> 8) & 0xFF, *mask* & 0xFF)
ADD TO DEVICE GROUPS 0-15
DTR2:DTR1 ((*mask* >> 24) & 0xFF, (*mask* >> 16) & 0xFF)
ADD TO DEVICE GROUPS 16-31
return

This test procedure adds one or more device groups according to the given mask, additionally using the given addressing byte.

Test description:

AddDeviceGroups (*mask*, *addressByte*)

DTR2:DTR1 ((*mask* >> 8) & 0xFF, *mask* & 0xFF)
ADD TO DEVICE GROUPS 0-15, send to device *addressByte*
DTR2:DTR1 ((*mask* >> 24) & 0xFF, (*mask* >> 16) & 0xFF)
ADD TO DEVICE GROUPS 16-31, send to device *addressByte*
return

12.14.7 RemoveDeviceGroups

This test procedure removes one or more device groups according to the given mask.

Test description:

RemoveDeviceGroups (*mask*)

DTR2:DTR1 ((*mask* >> 8) & 0xFF, *mask* & 0xFF)
REMOVE FROM DEVICE GROUPS 0-15
DTR2:DTR1 ((*mask* >> 24) & 0xFF, (*mask* >> 16) & 0xFF)
REMOVE FROM DEVICE GROUPS 16-31
return

This test procedure removes one or more device groups according to the given mask, additionally using the given addressing byte.

Test description:

RemoveDeviceGroups (*mask*, *addressByte*)

DTR2:DTR1 ((*mask* >> 8) & 0xFF, *mask* & 0xFF)
REMOVE FROM DEVICE GROUPS 0-15, send to device *addressByte*
DTR2:DTR1 ((*mask* >> 24) & 0xFF, (*mask* >> 16) & 0xFF)
REMOVE FROM DEVICE GROUPS 16-31, send to device *addressByte*
return

12.14.8 ClearAllDeviceGroups

This test procedure removes all device groups.

Test description:

ClearAllDeviceGroups ()

DTR2:DTR1 (0xFF,0xFF)
REMOVE FROM DEVICE GROUPS 0-15
REMOVE FROM DEVICE GROUPS 16-31

12.14.9 CheckDeviceGroups

This subsequence checks for the device group memberships by reading the device group assignments and also by the reaction on device group addressing using QUERY DEVICE CAPABILITIES.

Test description:

CheckGroupAssignment (*checkGroupAssignment*)

deviceGroups = **GetDeviceGroups** ()
if (*checkGroupAssignment* != *deviceGroups*)
    error 1 Group assignment does not match
endif
// check all group addresses for correct reaction
for (*i* = 0; *i* < 32; *i*++)
    status = QUERY DEVICE CAPABILITIES, send to device **GroupAddress** (*i*), **accept** No Answer
    if (*checkGroupAssignment* & (0x00000001 << *i*) == 0)
        // group should be not assigned => device should not react on this group
        if (*status* != NO)
            error 2 Device reacts on device group *i*
        endif
    else
        // group should be assigned => device should react on this group
        if (*status* == NO)
            error 3 Device does not react on device group *i*
        endif
    endif
endfor
return

12.14.10 GetDeviceGroups

This subsequence reads all 32 possible device group assignments.

Test description:

groups0to31 = **GetDeviceGroups**()

answer0 = QUERY DEVICE GROUPS 0-7
answer1 = QUERY DEVICE GROUPS 8-15
answer2 = QUERY DEVICE GROUPS 16-23
answer3 = QUERY DEVICE GROUPS 24-31
return (*answer3* << 24) | (*answer2* << 16) | (*answer1* << 8) | (*answer0*)

This subsequence reads all 32 possible device group assignments at a given device address.

Test description:

groups0to31 = **GetDeviceGroups**(*addressByte*)

answer0 = QUERY DEVICE GROUPS 0-7, send to device *addressByte*
answer1 = QUERY DEVICE GROUPS 8-15, send to device *addressByte*
answer2 = QUERY DEVICE GROUPS 16-23, send to device *addressByte*
answer3 = QUERY DEVICE GROUPS 24-31, send to device *addressByte*
return (*answer3* << 24) | (*answer2* << 16) | (*answer1* << 8) | (*answer0*)

12.14.11 PowerCycle

This subsequence performs an external power cycle for both bus powered and external bus powered devices. The duration of the power interruption is given in s.

Test description:

PowerCycle (*delay*)

```
if (GLOBAL_busPowered)
    if (delay == 5)
        delay = 0,550 // since a bus powered device has a power cycle of 550ms
    endif
    Disconnect (interface)
    wait delay s
    Connect (interface)
else
    Switch_off (external power)
    wait delay s
    Switch_on (external power)
endif
return
```

12.14.12 PowerCycleAndWaitForBusPower

This subsequence performs a PowerCycle and waits for the bus power to be restored. The duration of the power interruption is given in s. Subsequence returns the time (in ms) between finishing PowerCycle and the bus power being available.

Test description:

Time = **PowerCycleAndWaitForBusPower** (*delay*)

```
if (GLOBAL_internalBPS)
    // Switch off test power supply
    Apply (Current of 0 mA on bus terminals)
endif
```

PowerCycle (*delay*)
start_timer (*timer*)
if (*GLOBAL_internalBPS*)
    do
        voltage = **Measure** (Voltage on bus terminals in V)
        timestamp = **get_timer** (*timer*) *// Get time in seconds*
        if (*timestamp* > 7)
            halt 1 Internal bus power supply not available after 7 s.
        endif
    while (*voltage* < 12)
    if (*timestamp* > 5)
        error 1 Internal bus power supply not available after 5 s.
    endif
    // Restore test power supply
    Apply (Current of *GLOBAL_Ibus* mA on bus terminals)
endif
return (**get_timer** (*timer*))

12.14.13 PowerCycleAndWaitForDecoder

This subsequence performs a PowerCycleAndWaitForBusPower and waits for the decoder to be ready. The duration of the power interruption is given in s. The subsequence works for both bus powered and external bus powered devices.

Test description:

PowerCycleAndWaitForDecoder (*delay*)

timestamp = **PowerCycleAndWaitForBusPower** (*delay*)
if (*GLOBAL_busPowered*)
    waitTime = 1 200
else
    // Check for note 5, table 6, IEC 62386-101 Ed2.0
    if (*timestamp* < 350)
        waitTime = 450 - *timestamp*
    else
        waitTime = 100
    endif
endif
wait *waitTime* ms
return

12.14.14 SetupTestFrame

This subsequence is setting DTR0, DTR1 and DTR2 as preparation to request a test frame to be sent through the SEND_TESTFRAME command.

Test description:

SetupTestFrame (*frame*)

DTR0 ((*frame* >> 16) & 0xFF)
DTR1 ((*frame* >> 8) & 0xFF)
DTR2 ((*frame*) & 0xFF)

return

12.14.15 GetNumberOfInstances

Test subsequence returns the number of instances present in the bus unit.

Test description:

numberOfInstances = **GetNumberOfInstances** ()

numberOfInstances = QUERY NUMBER OF INSTANCES

return *numberOfInstances*

12.14.16 GetEventFilter

Test subsequence returns the event filter.

Test description:

eventFilter = **GetEventFilter** (*address*)

answer0-7 = QUERY EVENT FILTER 0-7, send to *address*

answer8-15 = QUERY EVENT FILTER 8-15, send to *address*

answer16-23 = QUERY EVENT FILTER 16-23, send to *address*

eventFilter = *answer16-23* << 16 + *answer8-15* << 8 + *answer0-7*

return *eventFilter*

12.14.17 SetEventFilter

Test subsequence sets the event filter.

Test description:

SetEventFilter (*address*, *data*)

DTR2 (*data* >> 16)

DTR1 ((*data* >> 8) & 0x00FF)

DTR0 (*data* & 0x0000FF)

SET EVENT FILTER, send to *address*

return

12.14.18 GetNumberOfLogicalUnits

Test subsequence returns the number of the logical control gear units present in the bus unit.

Test description:

numberLogicalUnits = **GetNumberOfLogicalUnits** ()

DTR1 (0)

DTR0 (0x19)

answer = READ MEMORY LOCATION

return *answer*

12.14.19 GetIndexOfLogicalUnit

Test subsequence returns the index number of the logical control gear unit.

Test description:

indexNumberLogicalUnit = **GetIndexOfLogicalUnit** (*address*)

DTR1 (0)

DTR0 (0x1A)

answer = READ MEMORY LOCATION, send to device (**ShortAddress** (*address*))

return *answer*

12.14.20 GetRandomAddress

Test subsequence returns the random address.

Test description:

randomAddress = **GetRandomAddress** ()

answerH = QUERY RANDOM ADDRESS (H)

answerM = QUERY RANDOM ADDRESS (M)

answerL = QUERY RANDOM ADDRESS (L)

randomAddress = answerH << 16 + answerM << 8 + answerL

return *randomAddress*

12.14.21 GetLimitedRandomAddress

Test subsequence tries 50 times to find a random address which has each generated byte different that 0x00 and 0xFF.

Test description:

randomAddress = **GetLimitedRandomAddress** (*logicalUnit*)

randomAddress = 0xFF FF FF

TERMINATE

INITIALISE (*logicalUnit*)

for (*i* = 0; *i* < 50; *i*++)

RANDOMISE

wait 100 ms

randomH = QUERY RANDOM ADDRESS (H), send to *logicalUnit*

randomM = QUERY RANDOM ADDRESS (M), send to *logicalUnit*

randomL = QUERY RANDOM ADDRESS (L), send to *logicalUnit*

if ((*randomH* != 0x00) AND (*randomH* != 0xFF) AND (*randomM* != 0x00) AND (*randomM* != 0xFF) AND (*randomL* != 0x00) AND (*randomL* != 0xFF))

randomAddress = answerH << 16 + answerM << 8 + answerL

break

endif

endfor

TERMINATE

return *randomAddress*

12.14.22 SetSearchAddress

Test subsequence sets the search address to 'data'.

Test description:

SetSearchAddress (*data*)

SEARCHADDRH (*data* >> 16)

SEARCHADDRM ((*data* >> 8) & (0x00 FF))

SEARCHADDRL (*data* & 0x00 00 FF)

return

12.14.23 SetShortAddress

Test subsequence sets new short address (toAddress) using SET SHORT ADDRESS, and using the following addressing mode:

- short address of logical unit: if logical unit already has a short address assigned (fromAddress)
- broadcast unaddressed: if logical unit has no short address assigned

Test description:

SetShortAddress (*fromAddress*; *toAddress*)

if (*toAddress* == 255)

dtrValue = 255

else if (*toAddress* <= 63)

dtrValue = *toAddress*

else

halt 1 Invalid toAddress argument in subsequence SetShortAddress. Actual: *toAddress.*

endif

DTR0 (*dtrValue*)

if (*fromAddress* != 255)

if (*fromAddress* <= 63)

answer = QUERY DEVICE CAPABILITIES, send to device **ShortAddress** (*fromAddress*), **accept** NO

if (*answer* != NO)

SET SHORT ADDRESS, send to device **ShortAddress** (*fromAddress*)

else

halt 2 Invalid fromAddress argument in subsequence SetShortAddress. Actual: *fromAddress.*

endif

else

halt 3 Invalid fromAddress argument in subsequence SetShortAddress. Actual: *fromAddress.*

endif

else

answer = QUERY DEVICE CAPABILITIES, send to device (**broadcast unaddressed**) , **accept** NO

if (*answer* != NO)

SET SHORT ADDRESS, send to device (**broadcast unaddressed**)

else

halt 4 Invalid fromAddress argument in subsequence SetShortAddress. Actual: *fromAddress.*

endif

endif

return

12.14.24 ReadMemBankMultibyteLocation

Test subsequence returns content of 'nrBytes' memory bank locations. If a gap is encountered, the value -1 is returned.

Test description:

multibyte = **ReadMemBankMultibyteLocation** (*nrBytes*)

multibyte = 0

for (*i* = 0; *i* < *nrBytes*; *i*++)

answer = READ MEMORY LOCATION

if (*answer* == NO)

multibyte = -1

break

endif

multibyte = multibyte + answer * 256 $^{nrBytes - 1 - i}$

endfor

return *multibyte*

12.14.25 FindImplementedMemoryBank

Test subsequence returns the number of the first implemented memory bank above memory bank 0 and the address of the last accessible memory location of that memory bank, for the selected logical unit.

Test description:

(*memoryBankNr*, *memoryBankLoc*) = **FindImplementedMemoryBank** ()

memoryBankNr = 0

memoryBankLoc = 0

DTR0 (2)

DTR1 (0)

lastMemBank = READ MEMORY LOCATION

for (*i* = 1; *i* <= *lastMemBank*; *i*++)

DTR0 (0)

DTR1 (*i*)

answer = READ MEMORY LOCATION

if (*answer* != NO)

memoryBankNr = i

memoryBankLoc = answer

break

endif

endfor

return (*memoryBankNr*, *memoryBankLoc*)

12.14.26 FindAllImplementedMemoryBanks

Test subsequence returns all implemented memory banks and the address of the last accessible memory location of each implemented memory bank above bank 0.

Test description:

(*memoryBankNr[]*, *memoryBankLoc[]*) = **FindAllImplementedMemoryBanks** ()

memoryBankNr[0] = 0

memoryBankLoc[0] = 0

count = 0

DTR0 (2)

DTR1 (0)

lastMemBank = READ MEMORY LOCATION

for (*i* = 1; *i* <= *lastMemBank*; *i*++)

DTR0 (0)

DTR1 (*i*)

answer = READ MEMORY LOCATION

if (*answer* != NO)

memoryBankNr[*count*] = *i*

memoryBankLoc[*count*] = *answer*

count++

endif

endfor

return (*memoryBankNr*[]; *memoryBankLoc*[])

12.14.27 ShortAddress

This subsequence returns a device address byte indicating a device short address generated from the given parameter shortAddress.

Test description:

(*addressByte*) = **ShortAddress** (*shortAddress*)

if (*shortAddress* > 63)
    halt 1 Short address out of range. Actual: *shortAddress*. Expected: [0..63].
endif
return (0000 0001b | (*shortAddress* << 1))

12.14.28 GroupAddress

This subsequence returns a device address byte indicating a device group address generated from the given parameter groupAddress.

Test description:

(*addressByte*) = **ShortAddress** (*groupAddress*)

if (*groupAddress* > 31)
    halt 1 Group address out of range. Actual: *groupAddress*. Expected: [0..31].
endif
return (1000 0001b | (*groupAddress* <<1))

12.14.29 Broadcast

This subsequence returns a device address byte indicating a device broadcast address.

Test description:

(*addressByte*) = **Broadcast** ()

return (1111 1111b)

12.14.30 BroadcastUnaddressed

This subsequence returns a device address byte indicating a device broadcast unaddressed address.

Test description:

(*addressByte*) = **BroadcastUnaddressed** ()

return (1111 1101b)

~~**12.14.31 InstanceNumber**~~

~~This subsequence returns an instance address byte indicating an instance number address generated from the given parameter instanceNumber.~~

~~**Test description:**~~

~~(*instanceByte*) = **InstanceNumber** (*instanceNumber*)~~

~~**if** (*instanceNumber* > 31)~~
~~**halt 1** Instance number out of range. Actual: *instanceNumber*. Expected: [0..31].~~
~~**endif**~~
~~**return** (0000 0000b | *instanceNumber*)~~

~~**12.14.32 InstanceGroup**~~

~~This subsequence returns an instance address byte indicating an instance group address generated from the given parameter instanceGroup.~~

~~**Test description:**~~

~~(*instanceByte*) = **InstanceGroup** (*instanceGroup*)~~

~~**if** (*instanceGroup* > 31)~~
~~**halt 1** Instance group out of range. Actual: *instanceGroup*. Expected: [0..31].~~
~~**endif**~~
~~**return** (1000 0000b | *instanceGroup*)~~

~~**12.14.33 InstanceType**~~

~~This subsequence returns an instance address byte indicating an instance type address generated from the given parameter instanceType.~~

~~**Test description:**~~

~~(*instanceByte*) = **InstanceType** (*instanceType*)~~

~~**if** (*instanceType* > 31)~~
~~**halt 1** Instance type out of range. Actual: *instanceType*. Expected: [0..31].~~
~~**endif**~~
~~**return** (1100 0000b | *instanceType*)~~

~~**12.14.34 InstanceBroadcast**~~

~~This subsequence returns an instance address byte indicating an instance broadcast address.~~

~~**Test description:**~~

~~(*instanceByte*) = **InstanceBroadcast** ()~~

~~**return** (1111 1111b)~~

~~**12.14.35 FeatureOfInstanceNumber**~~

~~This subsequence returns an instance address byte indicating a feature on instance number address generated from the given parameter instanceNumber.~~

Test description:

(*instanceByte*) = **FeatureOfInstanceNumber** (*instanceNumber*)

if (*instanceNumber* > 31)
    halt 1 Instance number (for feature) out of range. Actual: *instanceNumber*. Expected: [0..31].
endif
return (0010 0000b | *instanceNumber*)

12.14.36 FeatureOfInstanceGroup

This subsequence returns an instance address byte indicating a feature on instance group address generated from the given parameter instanceGroup.

Test description:

(*instanceByte*) = **FeatureOfInstanceGroup** (*instanceGroup*)

if (*instanceGroup* > 31)
    halt 1 Instance group (for feature) out of range. Actual: *instanceGroup*. Expected: [0..31].
endif
return (1010 0000b | *instanceGroup*)

12.14.37 FeatureOfInstanceType

This subsequence returns an instance address byte indicating a feature on instance type address generated from the given parameter instanceType.

Test description:

(*instanceByte*) = **FeatureOfInstanceType** (*instanceType*)

if (*instanceType* > 31)
    halt 1 Instance type (for feature) out of range. Actual: *instanceType*. Expected: [0..31].
endif
return (0110 0000b | *instanceType*)

12.14.38 FeatureOfInstanceBroadcast

This subsequence returns an instance address byte indicating a feature on instance broadcast address.

Test description:

(*instanceByte*) = **FeatureOfInstanceBroadcast** ()

return (1111 1101b)

12.14.39 FeatureOfDevice

This subsequence returns a device and instance address byte. The device address byte indicates a device short address generated from the given parameter shortAddress. The insatnce address byte indicates a feature of device address.

Test description:

(*addressByte*, *instanceByte*) = **FeatureOfDeviceWithShortAddress** (*shortAddress*)

if (*shortAddress* > 63)
    halt 1 Short address out of range. Actual: *shortAddress*. Expected: [0..63].
endif
addressByte = 0000 0001b | (*shortAddress* << 1)
instanceByte = 1111 1100b
return (*addressByte*, *instanceByte*)

12.14.40 FeatureOfDeviceWithGroupAddress

This subsequence returns a device and instance address byte. The device address byte indicates a device group address generated from the given parameter groupAddress. The insatnce address byte indicates a feature of device address.

Test description:

(*addressByte*, *instanceByte*) = **FeatureOfDeviceWithGroupAddress** (*groupAddress*)

if (*groupAddress* > 31)
    halt 1 Group address out of range. Actual: *groupAddress*. Expected: [0..63].
endif
addressByte = 1000 0001b | (*groupAddress* << 1)
instanceByte = 1111 1100b
return (*addressByte*, *instanceByte*)

12.14.41 FeatureOfDeviceWithBroadcast

This subsequence returns a device and instance address byte. The device address byte indicates a device broadcast address. The insatnce address byte indicates a feature of device address.

Test description:

(*addressByte*, *instanceByte*) = **FeatureOfDeviceWithBroadcast** ()

addressByte = 1111 1111b
instanceByte = 1111 1100b
return (*addressByte*, *instanceByte*)

Bibliography

~~[1] CISPR 15, *Limits and methods of measurement of radio disturbance characteristics of electrical lighting and similar equipment*~~

~~[2] IEC 60598-1, *Luminaires – Part 1: General requirements and tests*~~

~~[3] IEC 60669-2-1, *Switches for household and similar fixed electrical installations – Part 2-1, Particular requirements – Electronic switches*~~

~~[4] IEC 60921, *Ballasts for tubular fluorescent lamps – Performance requirements*~~

~~[5] IEC 60923, *Auxiliaries for lamps – Ballasts for discharge lamps (excluding tubular fluorescent lamps) – Performance requirements*~~

~~[6] IEC 61347 (all parts), *Lamp controlgear*~~

~~[7] IEC 60929, *AC and/or DC-supplied electronic control gear for tubular fluorescent lamps – Performance requirements*~~

~~[8] IEC 61347-1, *Lamp controlgear – Part 1: General and safety requirements*~~

~~[9] IEC 61347-2-3, *Lamp control gear – Part 2-3: Particular requirements for a.c and/or d.c. supplied electronic control gear for fluorescent lamps*~~

~~[10] IEC 61547, *Equipment for general lighting purposes – EMC immunity requirements*~~

~~[11] IEC 62034, *Automatic test systems for battery powered emergency escape lighting*~~

[1] *GS1 General Specification*, latest version, available at: www.gs1.org

[2] IEC 62386-104:2019, *Digital addressable lighting interface – Part 104: General requirements – Wireless and alternative wired system components*

[3] IEC 62386-103:2014[2], *Digital addressable lighting interface – Part 103: General requirements – Control devices*

2 The first edition will be withdrawn and replaced upon publication of this second edition.

List of comments

1 Figure 1 is updated since the IEC 62386 series was extended by new parts.

2 This is primarily used for memory bank content.

3 This is a clarification as event message encoding is defined in Subclause 7.2.2.

4 This is a clarification as event messages are described in Subclause 9.7.

5 Deleted since this is redundant information.

6 For input devices which measure a signal with a certain resolution (e.g. IEC 62386-304 light sensor), the description of event generation based on hysteresis was not clear enough, as the input value had two representations: A value with "*resolution*" bits, and the N-byte "*inputValue*" variable. Therefore the new term "measured value" is introduced representing the value based on the resolution.

Example: An input signal which is measured with 14-bit resolution will be represented as a 14-bit measured value, which is scaled into the 2-byte "*inputValue*" variable.

7 This is a clarification of behaviour during quiescent mode.

8 All updates within Subclause 9.11 are made to be in line with IEC 62386-102 behaviour.

9 The RAM and NVM memory types are sub-divided into RO and RW types. This distinction is expected to be required in some future parts 3xx of IEC 62386, such as part 351.

This table clarifies the volatility and possibility for modification over the interface, for each memory type.

10 Protectable memory locations may be introduced in future IEC 62386 parts 3xx.

11 These requirements are moved to Subclause 9.11.5.3.

12 This paragraph is incorporated in Subclause 9.11.5.2.

13 This allows for multi-byte values to be introduced in future additions or updates to IEC 62386 parts 3xx. The requirements here allow a consistent method to reliably read multi-byte values. Starting a read operation from the first byte fetches the latest value, and latches the remaining bytes, allowing them to be read without the possibility of their values changing before they can be read out.

14 Unimplemented locations now differ depending on whether MASK is part of the allowed range of validity or not; this is related to protectable memory locations described in Subclause 9.11.4.

15 Temporarily unavailable locations with value TMASK may be introduced in future additions or updates to IEC 62386 parts 3xx.

16 This latching mechanism of complete memory banks allows for future additions or updates to IEC 62386 parts 3xx. It allows for the read-out of a memory bank's contents over a period of time, with all content relating to the same point in time (the time when the bank was latched).

17 This mechanism for writing multi-byte values allows for future additions or changes to IEC 62386 parts 3xx.

18 "Current bus unit configuration" is a new and optional feature, described in Subclause 9.20.

19 The command SAVE PERSISTENT VARIABLES is removed. Subclause 9.18 gives the updated conditions for saving of the NVM variables.

20 This clarifies the required use of the priorities, but does not change the requirements from previous edition 1.0.

21 Instance configuration is new and optional functionality. See Subclause 9.19.

22 Storing NVM variables is changed in a way that control devices must now automatically save NVM variables to flash (non-volatile memory), ensuring they are saved in case a power-cycle occurs within a defined time of the changed variables. The control devices may no longer rely on a command to force this save to non-volatile memory. Therefore the command SAVE PERSISTENT VARIABLES is removed.

23 Input devices can have physical inputs or sensors that could be supported by more than one part 3xx of IEC 62386. The actual desired function may depend on the connected signal source, or the intended application.

Example: An input device for manual control could provide functionality according to IEC 62386-301 if a push button is connected, or it could provide functionality according to IEC 62386-302 if a switch is connected.

Since it is not possible for most input devices to detect the type of signal source automatically, this new functionality in edition 2.0 now provides the possibility to change the instance types and related instance configuration of an input device.

24 This is a new and optional feature, introduced in line with IEC 62386-102 edition 3.0.

Some control devices can be programmed in the factory to operate in different bus unit configurations, whereas the GTIN remains the same. This edition 2.0 now provides a new byte in memory bank 0 to read out the current configuration. Currently there are no configurations standardized for control devices, but the values shown as "reserved" in Table 18 allow for future standardized configurations.

25 This is a clarification as there are actually two variables "*eventPriority*": One is a device variable, one is an instance variable.

26 This is a new and optional feature described in Subclause 9.19.

27 This is a clarification as there are actually two variables "*eventPriority*": One is a device variable, one is an instance variable.

28 This is a clarification rather than a change in requirements.

29 The possibilities for device identification are extended to allow, in some cases, identification via manually triggered event messages as an alternative to the identification procedure triggered by the command IDENTIFY DEVICE.

30 This is a clarification rather than a change.

31 Storing NVM variables is changed in a way that control devices must now automatically save NVM variables to flash (non-volatile memory), ensuring they are saved in case a power-cycle occurs within a defined time of the changed variables. The control devices may no longer rely on a command to force this save to non-volatile memory. Therefore the command SAVE PERSISTENT VARIABLES is removed.

32 This command description is added here to show explicitly that this command exists at device level, as well as at instance level. See Subclause 11.8.8.

33 This is a clarification rather than a change.

34 This command description is added here to show explicitly that this command exists at device level, as well as at instance level. See Subclause 11.9.13.

35 This is a clarification rather than a change.

36 This is a clarification rather than a change.

37 Input devices can have physical inputs or sensors that could be supported by more than one part 3xx of IEC 62386. The actual desired function may depend on the connected signal source, or the intended application.

Example: An input device for manual control could provide functionality according to IEC 62386-301 if a push button is connected, or it could provide functionality according to IEC 62386-302 if a switch is connected.

Since it is not possible for most input devices to detect the type of signal source automatically, this new functionality in edition 2.0 now provides the possibility to change the instance types and related instance configuration of an input device.

38 Input devices can have physical inputs or sensors that could be supported by more than one part 3xx of IEC 62386. The actual desired function may depend on the connected signal source, or the intended application.

Example: An input device for manual control could provide functionality according to IEC 62386-301 if a push button is connected, or it could provide functionality according to IEC 62386-302 if a switch is connected.

Since it is not possible for most input devices to detect the type of signal source automatically, this new functionality in edition 2.0 now provides the possibility to change the instance types and related instance configuration of an input device.

39 This is a clarification rather than a change.

40 Input devices can have physical inputs or sensors that could be supported by more than one part 3xx of IEC 62386. The actual desired function may depend on the connected signal source, or the intended application.

Example: An input device for manual control could provide functionality according to IEC 62386-301 if a push button is connected, or it could provide functionality according to IEC 62386-302 if a switch is connected.

Since it is not possible for most input devices to detect the type of signal source automatically, this new functionality in edition 2.0 now provides the possibility to change the instance types and related instance configuration of an input device.

41 Input devices can have physical inputs or sensors that could be supported by more than one part 3xx of IEC 62386. The actual desired function may depend on the connected signal source, or the intended application.

Example: An input device for manual control could provide functionality according to IEC 62386-301 if a push button is connected, or it could provide functionality according to IEC 62386-302 if a switch is connected.

Since it is not possible for most input devices to detect the type of signal source automatically, this new functionality in edition 2.0 now provides the possibility to change the instance types and related instance configuration of an input device.

42 This is important to note for design of application controllers that assign short addresses to control gear and control devices.

43 Test procedures are no longer part of the IEC document.

IEC 62386-103

Edition 2.0 2022-11

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Digital addressable lighting interface – Part 103: General requirements – Control devices

Interface d'éclairage adressable numérique – Partie 103: Exigences générales – Dispositifs de commande

IEC 62386-103:2022-11(en-fr)

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

DIGITAL ADDRESSABLE LIGHTING INTERFACE –

Part 103: General requirements – Control devices

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

IEC 62386-103 has been prepared by IEC technical committee 34: Lighting. It is an International Standard.

This second edition cancels and replaces the first edition published in 2014 and Amendment 1:2018. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

a) the scope has been updated;
b) quiescent mode has been updated;
c) non-volatile memory (NVM) save time has been added, and SAVE PERSISTENT VARIABLES command removed;
d) memory bank 0 has been modified, and common memory bank requirements have been added;

e) IDENTIFY DEVICE has been updated;

f) version number has been changed;

g) bus unit configuration has been added; and

h) instance types and configuration have been added.

The text of this International Standard is based on the following documents:

Draft	Report on voting
34/946/FDIS	34/990/RVD

Full information on the voting for its approval can be found in the report on voting indicated in the above table.

The language used for the development of this International Standard is English.

This document was drafted in accordance with ISO/IEC Directives, Part 2, and developed in accordance with ISO/IEC Directives, Part 1 and ISO/IEC Directives, IEC Supplement, available at www.iec.ch/members_experts/refdocs. The main document types developed by IEC are described in greater detail at www.iec.ch/standardsdev/publications.

This Part 103 of IEC 62386 is intended to be used in conjunction with Part 101, which contains general requirements for the relevant product type (system), and with the appropriate Parts 3xx (particular requirements for control devices) containing clauses to supplement or modify the corresponding clauses in Part 101 and Part 103 in order to provide the relevant requirements for each type of product.

A list of all parts in the IEC 62386 series, published under the general title *Digital addressable lighting interface*, can be found on the IEC website.

The committee has decided that the contents of this document will remain unchanged until the stability date indicated on the IEC website under webstore.iec.ch in the data related to the specific document. At this date, the document will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

IMPORTANT – The "colour inside" logo on the cover page of this document indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

INTRODUCTION

IEC 62386 contains several parts, referred to as series. The IEC 62386 series specifies a bus system for control by digital signals of electronic lighting equipment. The IEC 62386-1xx series includes the basic specifications. Part 101 contains general requirements for system components, Part 102 extends this information with general requirements for control gear and Part 103 extends it further with general requirements for control devices. Part 104 and Part 105 can be applied to control gear or control devices. Part 104 gives requirements for wireless and alternative wired system components. Part 105 describes firmware transfer. Part 150 gives requirements for an auxiliary power supply which can be stand-alone, or built into control gear or control devices.

The IEC 62386-2xx series extends the general requirements for control gear with lamp specific extensions (mainly for backward compatibility with Edition 1 of IEC 62386) and with control gear specific features.

The IEC 62386-3xx series extends the general requirements for control devices with input device specific extensions describing the instance types as well as some common features that can be combined with multiple instance types.

This second edition of IEC 62386-103 is intended to be used in conjunction with IEC 62386-101 and with the various parts that make up the IEC 62386-3xx series of particular requirements for control devices, and can be used together with IEC 62386-102 and with the various parts that make up the IEC 62386-2xx series for control gear. The division into separately published parts provides for ease of future amendments and revisions. Additional requirements will be added as and when a need for them is recognised.

The setup of the standards is graphically represented in Figure 1 below.

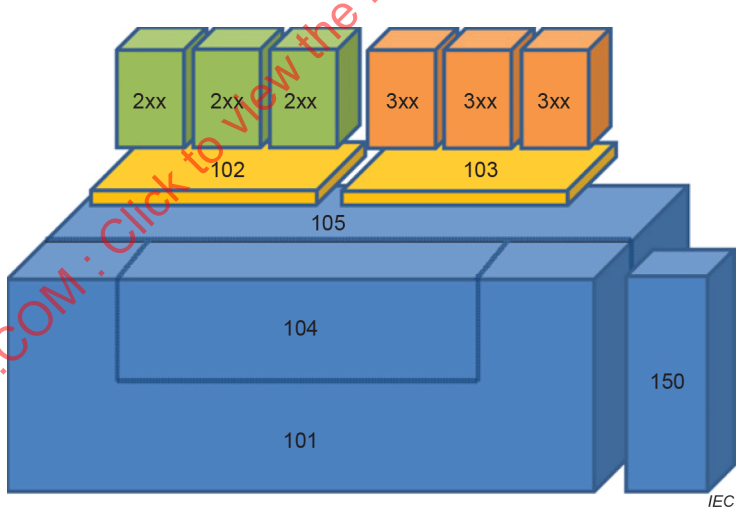


Figure 1 – IEC 62386 graphical overview

When this part of IEC 62386 refers to any of the clauses of the other parts of the IEC 62386-1xx series, the extent to which such a clause is applicable is specified. The other parts also include additional requirements, as necessary.

All numbers used in this document are decimal numbers unless otherwise noted. Hexadecimal numbers are given in the format 0xVV, where VV is the value. Binary numbers are given in the format XXXXXXXXb or in the format XXXX XXXX, where X is 0 or 1, "x" in binary numbers means "don't care".

The following typographic expressions are used:

Variables: *variableName* or *variableName[3:0]*, giving only bits 3 to 0 of *variableName*;

Range of values: [lowest, highest];

Command: "COMMAND NAME".

DIGITAL ADDRESSABLE LIGHTING INTERFACE –

Part 103: General requirements – Control devices

1 Scope

This part of IEC 62386 is applicable to control devices for control by digital signals of electronic lighting equipment.

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.

IEC 62386-101:2022, *Digital addressable lighting interface – Part 101: General requirements – System components*

IEC 62386-102:2022, *Digital addressable lighting interface – Part 102: General requirements – Control gear*

IEC 62386-3xx (all parts), Digital addressable lighting interface – Part 3xx: Particular requirements for control devices

3 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 62386-101 and the following apply.

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

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

3.1
broadcast
type of address used to simultaneously address all control devices in the system

3.2
broadcast unaddressed
type of address used to simultaneously address all control devices in the system that have no short address

3.3
device command
command which addresses the control device and has a value of 0xFE in the instance byte of the command frame but is not an event message

3.4
device group
type of address used to address a group of control devices in the system at once

3.5
DTR
data transfer register
multipurpose register used to exchange data

3.6
event
instance report, characterized by its event number, of a change or a defined sequence of changes of its input value

Note 1 to entry: The event number is specific to the type of instance that sends the report.

3.7
event scheme
characterisation of the information, as provided by an instance when producing an event message, that identifies the source of the event

3.8
feature
optional extension at instance and/or device level

3.9
feature command
command which addresses one or more features of an input device or device instance and has a value different from 0xFE in the instance byte of the command frame but is not an instance command or an event message

3.10
GTIN
global trade item number
number used for the unique identification of trade items worldwide

Note 1 to entry: For further information see http://en.wikipedia.org/wiki/GTIN.

Note 2 to entry: The global trade item number is comprised of a GS1 or U.P.C. company prefix followed by an item reference number and a check digit. It is described in the "GS1 General Specifications" (see [1]).

3.11
input signal
physical value that an instance of an input device is designed to detect and process

Note 1 to entry: Examples for physical values are "illuminance" and "button state".

3.12
identification
temporary state used during commissioning that allows the installer to identify particular control devices

3.13
input value
encoded data, representing the input signal

Note 1 to entry: The way in which the input signal is encoded depends on the instance type.

3.14
instance command
command which addresses one or more instances of an input device and has a value different from 0xFE in the instance byte of the command frame but is not a feature command or an event message

3.15
MASK
value with all binary digits set to 1

Note 1 to entry: This means that an 8-bit backward frame of MASK is a value of 0xFF, and a multi-byte memory location of 24 bits containing MASK is a value of 0xFFFFFF.

3.16
NO
answer to a query where no backward frame is sent

Note 1 to entry: If a query is asked where the answer is NO, there will be no response, such that the sender of the query will conclude "no backward frame" following IEC 62386-101:2022, 8.2.5.

Note 2 to entry: The answer NO could also be triggered by a missed query.

3.17
NVM
non-volatile memory
non-volatile read/write memory, the content of which can be changed and will not be lost due to a power cycle

3.18
NVM-RO
NVM that cannot be written using any command

3.19
NVM-RW
NVM that can be modified using one or more commands

3.20
opcode
operation code
that part of a command frame that identifies the command to be executed

3.21
operating mode
set of states identified by a number in the range [0,255], characterised by a collection of variables and memory settings, and used to select a set of functionalities to be exhibited by a control device, including its required reaction to commands

Note 1 to entry: Control devices can support more than one operating mode.

3.22
PING
16-bit forward frame with bits [15:0] equal to 0xAD00

Note 1 to entry: As specified in IEC 62386-102, PING has no meaning for control gear.

3.23
quiescent mode
temporary mode in which the device does not send forward frames

3.24
RAM
volatile read/write memory, the content of which can be changed and will be lost due to a power cycle

3.25
RAM-RO
RAM that cannot be written using any command

3.26
RAM-RW
RAM that can be modified using one or more commands

3.27
random address
random 24-bit number generated by the control device on request during system initialisation

3.28
reset state
state in which all NVM variables of the control device have their reset value, except those that are marked "no change" or are otherwise explicitly excluded

3.29
ROM
non-volatile read-only memory, the content of which is fixed

Note 1 to entry: In this document read-only is meant from a system perspective. A ROM variable may actually be implemented in NVM, but this document does not provide any mechanism to change its value.

3.30
search address
24-bit number used to identify an individual control device in the system during initialisation

3.31
short address
type of address used to address an individual control device in the system

3.32
TMASK
value with the least significant bit set to 0 and all other binary digits set to 1

Note 1 to entry: This means that an 8-bit backward frame of TMASK is a value of 0xFE, and a multi-byte memory location of 24 bits containing TMASK is a value of 0xFFFFFE.

3.33
YES
answer to a query where a backward frame of MASK is sent

4 General

4.1 General

The requirements of IEC 62386-101:2022, Clause 4 apply, with the restrictions, changes and additions identified below.

4.2 Version number

This subclause replaces IEC 62386-101:2022, 4.2.

The version shall be in the format "x.y", where the major version number x is in the range of 0 to 62 and the minor version number y is in the range of 0 to 2. When the version number is encoded into a byte, the major version number x shall be placed in bits 7 to 2 and the minor version number y shall be placed in bits 1 to 0.

At each amendment to an edition of IEC 62386-103 the minor version number shall be incremented by one.

At a new edition of IEC 62386-103 the major version number shall be incremented by one and the minor version number shall be set to 0.

The current version number is "*versionNumber*" as defined in Table 19.

NOTE IEC documents are generally subject to two amendments before a new edition is prepared.

5 Electrical specification

The requirements of IEC 62386-101:2022, Clause 5 apply.

6 Bus power supply

If a bus power supply is integrated into a control device, the requirements of IEC 62386-101:2022, Clause 6 apply.

7 Transmission protocol structure

7.1 General

The requirements of IEC 62386-101:2022, Clause 7 apply with the following additions.

7.2 24-bit forward frame encoding

7.2.1 Frame format for instructions and queries

7.2.1.1 General

For 7.2.1, commands shall be interpreted as instructions and queries, and exclude event messages. The 24-bit forward frame shall be encoded as shown in Table 1 and Table 2.

Table 1 – 24-bit command frame encoding

Bytes/Bits										Device addressing
Address byte								Instance byte	Opcode byte	
23	22	21	20	19	18	17	16	15…8	7…0	
0	64 short addresses						1	Device or instance or feature, see Table 2		Short addressing
1	0	32 device group addresses					1			Device group addressing
1	1	1	1	1	1	0	1			Broadcast unaddressed
1	1	1	1	1	1	1	1			Broadcast
1	1	0	16 special command spaces				1	Command specific		Special command
1	1	1	0	x	x	x	1	Reserved		Reserved
1	1	1	1	0	x	x	1			
1	1	1	1	1	0	x	1			

Table 2 – Instance byte in a command frame

Instance byte								Addressing
15	14	13	12	11	10	09	08	
0	0	0	32 Instance numbers					Instance number
1	0	0	32 Instance groups					Instance group
1	1	0	32 Instance types					Instance type
0	0	1	32 Instance numbers					Feature on instance number level
1	0	1	32 Instance groups					Feature on instance group level
0	1	1	32 Instance types					Feature on instance type level
1	1	1	1	1	0	0	1	Feature broadcast
1	1	1	1	1	1	0	1	Feature on instance broadcast level
1	1	1	1	1	1	1	1	Instance broadcast
1	1	1	1	1	1	0	0	Feature on device level
1	1	1	1	1	1	1	0	Device
0	1	0	x	x	x	x	x	Reserved
1	1	1	0	x	x	x	x	
1	1	1	1	0	x	x	x	
1	1	1	1	1	0	1	x	
1	1	1	1	1	0	0	0	

7.2.1.2 Address byte

The address byte provides

- the method of device addressing used by the transmitter;
- the indication that a command, not an event message, is being transmitted: bit 16 is set for commands;
- 16 special command spaces;
- reserved device addresses. Reserved addresses shall not be used by the transmitter.

7.2.1.3 Instance byte

The instance byte provides

- for standard commands, the indication of whether a device command, feature command or an instance command is being transmitted;
- for standard instance commands, the method of instance addressing used by the transmitter;
- command-specific information for special commands;
- for standard commands, reserved instance addresses. Reserved instance addresses shall not be used by the transmitter;
- for standard feature commands, the feature that is being addressed;
- reserved information for reserved commands.

7.2.1.4 Opcode byte

The opcode byte provides

- for standard commands, the opcode;
- command-specific information for special commands;
- reserved information for reserved commands.

7.2.2 Frame format for event messages

7.2.2.1 General

For event messages, the 24-bit forward frame shall be encoded as shown in Table 3.

Table 3 – 24-bit event message frame encoding

Bits														Event scheme[a] / Source	
Event source information														Event information	
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9...0	
0	64 short addresses						0	0	32 instance types					Event	1 Device
0	64 short addresses						0	1	32 instance numbers						2 Device and instance
1	0	32 device groups					0	0	32 instance types						3 Device group
1	0	32 instance types					0	1	32 instance numbers						0 Instance
1	1	32 instance groups					0	0	32 instance types						4 Instance group
1	1	0	x	x	x	x	0	1	x	x	x	x	x	Reserved	Reserved
1	1	1	0	x	x	x	0	1	x	x	x	x	x		
1	1	1	1	0	x	x	0	1	x	x	x	x	x		
1	1	1	1	1	0	x	0	1	x	x	x	x	x		
1	1	1	1	!	1	0	0	1	x	x	x	x	x		
1	1	1	!	1	1	1	0	1	0	x	x	x	x		
1	1	1	1	1	1	1	0	1	1	0	x	x	x		
1	1	1	1	1	1	1	0	1	1	1	Short address and device group information, refer to 9.7.2.				Device power cycle

[a] Refer to 9.7.3 for further information on event schemes.

7.2.2.2 Event source information

The event source information in Table 3 provides:

- the indication that an event message, not an instruction or query, is being transmitted: bit 16 shall be clear for event messages;
- relevant event instance type information, such that the receiver of an event message will be able to understand the meaning of the event;
- relevant event source information, such that the receiver of an event message is able to understand where the message is coming from;
- reserved values.

Events are instance type specific. Application controllers can derive, either explicitly or implicitly, the instance type of the transmitting instance, from the event source information.

NOTE The event source schemes are not equally valuable in terms of telling the receiver where the event message originated.

7.2.2.3 Event information

The event information provides the 10-bit event number and/or event data. Event information is instance type specific and is defined in the applicable parts of the IEC 62386-3xx series that describe the instance type.

8 Timing

The requirements of IEC 62386-101:2022, Clause 8 apply.

9 Method of operation

9.1 General

The requirements of IEC 62386-101:2022, Clause 9 apply with the following additions.

9.2 Device features

The IEC 62386 series allows for the future publication of feature extensions that extend the requirements in this document, or exempt particular requirements.

The features for a device can be queried by "QUERY FEATURE TYPE" and "QUERY NEXT FEATURE TYPE" while the instance byte is set to "feature on device level" (see Table 2).

Table 5 shows the feature type encoding. For further information on the different feature types see parts of the IEC 62386-3xx series.

9.3 Application controller

9.3.1 General

An application controller is that part of a control system that makes the system "work":

- an application controller can commission and configure the system (including available control gear);
- an application controller can make the system react to changes in the environment (based on information coming from input devices);

- an application controller can change the system response of control gear in the system (possibly using any command defined in IEC 62386-102:2022).

9.3.2 Single-master application controller

A single-master application controller is not intended to share the bus with other control devices.

A single-master application controller can try to configure other control devices on the bus, and/or change the system response of control gear in the system, thereby using any command defined in IEC 62386-102 and/or instructions and queries defined in this document.

NOTE Especially if the single-master application controller does not handle collisions appropriately, any such attempt can fail and affect the system negatively.

On the other hand, a single-master application controller is not required to have a receiver on board. For this reason, the following holds:

From 9.3.3 onwards, this document assumes a control device to be a multi-master control device.

In order to make itself known as a possibly anonymous transmitting bus unit, a single-master application controller shall transmit a PING message at regular intervals of 10 min ± 1 min. The first such PING message shall appear at a random time between 5 min and 10 min after completion of the power-on procedure.

9.3.3 Multi-master application controller

From 9.3.3 onwards, this document assumes a control device to be a multi-master control device.

A control device that includes an application controller shall have *"applicationControllerPresent"* set to TRUE. *"applicationControllerPresent"* shall be set to FALSE otherwise.

NOTE 1 *"applicationControllerPresent"* can be observed through "QUERY DEVICE CAPABILITIES".

In most cases, a system will have only one application controller active (refer to 9.10.1), but multiple application controllers can be operational in a single system.

An application controller shall accept commands (from other application controllers) according to Table 23 and Table 24. It is part of the system integration to ensure that the application controllers will accept commands in such a way that it results in a correctly functioning system.

NOTE 2 System integrity is easiest to achieve by allowing only a single application controller to do commissioning and configuration.

NOTE 3 An application controller can be commissioned through alternative interfaces.

An application controller shall not transmit event messages other than for the device power cycle event.

NOTE 4 If an application controller is active, it can send 24-bit forward frames for purposes other than transmitting events.

An application controller shall not transmit PING messages.

9.4 Input device

Input devices make a system sensitive to changes in its environment, by transmitting event messages.

Input devices shall be multi-master control devices and shall allow commissioning and configuration by an application controller.

Input devices shall use forward frames only to transmit event messages.

9.5 Instances of input devices

9.5.1 General

An input device shall have at least one instance and a maximum of 32 instances, as shall be indicated by *"numberOfInstances"*, which can be queried using "QUERY NUMBER OF INSTANCES".

A control device that is only an application controller shall have a *"numberOfInstances"* equal to 0.

9.5.2 Instance number

Each instance shall have a unique *"instanceNumber"* in the range [0,*"numberOfInstances" – 1*].

9.5.3 Instance type

The instance type for each of the instances of an input device can be different. It can be queried by "QUERY INSTANCE TYPE" while the instance byte is set to either "instance type" or " instance number" scheme (see Table 2). The meaning of event information transmitted by means of "INPUT NOTIFICATION (*device/instance, event*)" depends on the instance type.

Table 4 shows the instance type encoding. For further information on the different instance types see the relevant parts of the IEC 62386-3xx series.

Table 4 – Instance types

Instance type	IEC 62386	Used for
0	Part 103	Generic purpose, input devices that are not defined. Another method of identifying the device shall be implemented, to allow application controller to interpret the events.
1 to 31	Part 301 to Part 331	These IEC 62386-3xx parts describe instance types, where xx ranges from 01 to 31.

9.5.4 Instance features

The IEC 62386 series allows for the future publication of feature extensions that extend the requirements in this document, or exempt particular requirements.

The features for each of the instances of an input device can be different. They can be queried by "QUERY FEATURE TYPE" and "QUERY NEXT FEATURE TYPE" while the instance byte is set to either "feature on instance type level" or "feature on instance number level" scheme (see Table 2).

Table 5 shows the feature type encoding. For further information on the different feature types see the relevant parts of the IEC 62386-3xx series.

Table 5 – Feature types

Feature type	IEC 62386	Used for
32 to 96	Part 332 to Part 396	These IEC 62386-3xx parts describe feature extensions, where xx ranges from 32 to 96.

9.5.5 Instance groups

Instance groups are a means for an application controller to put instances into logical groups, across input devices. Consequently, these logical groups can be used to configure multiple instances at once.

An application controller can use up to 32 such groups, numbered in the range [0,31]. Each instance can be declared to be a member of up to 3 instance groups and shall expose instance group variables as given in Table 6.

Table 6 – Instance group variables

Variable	Description
"instanceGroup0"	Primary instance group number, MASK if no membership defined.
"instanceGroup1"	Additional instance group number, MASK if no membership defined.
"instanceGroup2"	Additional instance group number, MASK if no membership defined.

Instance groups are assigned and queried by using the following instance operations:

- "SET PRIMARY INSTANCE GROUP (*DTR0*)", "QUERY PRIMARY INSTANCE GROUP";
- "SET INSTANCE GROUP 1 (*DTR0*)", "QUERY INSTANCE GROUP 1";
- "SET INSTANCE GROUP 2 (*DTR0*)", "QUERY INSTANCE GROUP 2".

The primary group is special in the sense that only this number shall be used when reporting events (if instance group event reporting is used). Additional groups are a means of configuring multiple instances at once.

9.6 Commands excluding event messages

9.6.1 General

A control device shall check the device addressing scheme to see if it is addressed by a command. The control device shall accept the command, unless any of the following conditions hold:

- the command is sent using short addressing, and the given short address is not equal to *"shortAddress"*;
- the command is sent using device group addressing, and the given device group does not match any of the groups identified by *"deviceGroups"*;
- the command is sent using broadcast unaddressed addressing and *"shortAddress"* is not MASK;
- the command is sent using reserved addressing;
- the command is not defined;
- the command is sent using feature addressing, and the given feature is not implemented.

NOTE For instance and feature commands, additional conditions for command acceptance hold. These are given in 9.6.3 and 9.6.4.

9.6.2 Device commands

The instance byte shall be 0xFE for device commands. If the instance byte is not equal to 0xFE, the control device shall not accept these commands.

NOTE This addressing mechanism allows the opcode values for device commands and instance commands to overlap.

9.6.3 Instance commands

For instance commands that are accepted by an input device (refer to 9.5), the instance addressing scheme determines the intended (set of) receiving instances within that device. An instance shall accept the instance command, unless any of the following additional conditions hold:

- the command is sent using instance number addressing and the given instance number is not equal to *"instanceNumber"*;
- the command is sent using instance group addressing, and the given instance group does not match any of the groups identified by *"instanceGroup0"*, *"instanceGroup1"* and *"instanceGroup2"* (see Table 6);
- the command is sent using instance type addressing and the given instance type is not equal to *"instanceType"*;
- the command is sent using reserved addressing.

9.6.4 Feature commands

For feature commands that are accepted by a device or instance (refer to 9.5), the feature addressing scheme determines the intended (set of) receiving features within that device (see Table 2).

A feature on device level shall accept the feature command unless any of the following additional conditions hold:

- the command is sent using feature addressing other than feature addressing on the device level;
- the command is sent with instance byte reserved addressing.

A feature on instance level shall accept the feature command unless any of the following additional conditions hold:

- the command is sent using feature on instance number level addressing and the given instance number is not equal to "*instanceNumber*";
- the command is sent using feature on instance group level addressing and the given instance group does not match any of the groups identified by "*instanceGroup0*", "*instanceGroup1*" and "*instanceGroup2*";
- the command is sent using feature on instance type level addressing and the given instance type is not equal to "*instanceType*";
- the command is sent using feature on device level addressing;
- the command is sent with instance addressing;
- the command is sent with device addressing;
- the command is sent with instance byte reserved addressing.

A feature broadcast command shall be used to address a given feature both on device level and instance level. A feature broadcast shall be accepted unless any of the following additional conditions hold:

- the command is sent using feature addressing other than feature broadcast;
- the command is sent with instance byte reserved addressing.

If the feature command is accepted, the opcode byte determines which feature is addressed.

9.7 Event messages

9.7.1 Response to event messages

An application controller is free to act upon reception of any event message or to ignore the message.

NOTE A disabled application controller does not send any forward frames, but can still update its internal state based on messages received.

9.7.2 Device power cycle event

Since the power cycle event (see 9.13.2) is a device event, it does not adhere to the default event frame format. Bits 12 through 0 shall carry device address information as indicated in Table 7.

Table 7 – Device address information in power cycle event

Bits												
12	11	10	09	08	07	06	05	04	03	02	01	00
1 = device group valid	Lowest device group					1 = short address valid	Short address					

Bit 12 shall be set if and only if the transmitting control device is member of at least one device group. Bits [11:7] shall indicate the lowest device group number of membership in that case. If bit 12 is not set, bits [11:7] shall be clear.

Bit 6 shall be set if and only if the transmitting control device has a *"shortAddress"* different from MASK. Bits [5:0] shall indicate the device short address in that case. If bit 6 is not set, bits [5:0] shall be clear.

9.7.3 Input notification event

An instance of an input device shall, when transmitting an event message, use the selected event source addressing scheme as defined in Table 8.

Table 8 – Event addressing schemes

"eventScheme"	Description
0 (default)	Instance addressing, using instance type and number.
1	Device addressing, using short address and instance type.
2	Device and instance addressing, using short address and instance number.
3	Device group addressing, using device group and instance type.
4	Instance group addressing, using instance group and type.

An application controller can set and query the *"eventScheme"* by means of "SET EVENT SCHEME (*DTR0*)" and "QUERY EVENT SCHEME" respectively.

NOTE 1 An instance can only implement an event scheme while certain conditions have been satisfied by the application controller as well. Instance addressing is the only addressing scheme that will work under all circumstances.

In the following situations, the instance shall immediately revert *"eventScheme"* to the default instance addressing scheme shown in Table 8:

- *"eventScheme"* has been set to 1 or 2 whereas the containing logical unit has no short address;
- *"eventScheme"* has been set to 3 whereas the containing logical unit is not a member of a device group;
- *"eventScheme"* has been set to 4 whereas the instance has no primary instance group membership (see 9.5.5).

NOTE 2 The above situations can occur because of a new "SET EVENT SCHEME (*DTR0*)" command and/or because of a change of conditions.

NOTE 3 This implies that the command "SET EVENT SCHEME (*DTR0*)" can "fail", rather than setting a preference that is granted sometime later. To avoid this failure, application controllers can set the desired event scheme after completing those configuration aspects that influence event scheme operation.

Furthermore, and given a viable addressing scheme, the instance shall

- only refer to *"instanceNumber"* as the instance number;
- only refer to *"instanceType"* as the instance type;
- only refer to *"instanceGroup0"* as the instance group;
- only refer to *"shortAddress"* as the short address of the containing logical unit;
- only refer to the lowest device group number that the containing logical unit is a member of.

9.7.4 Event message filter

The event message filter can be used to enable and disable specific events. While the event filter of a specific event is disabled, this specific event shall not be generated. To enable or disable all events see 9.10.3.

An application controller can set the *"eventFilter"* by means of SET EVENT FILTER (*DTR2:DTR1:DTR0*) and can query the variable by means of QUERY EVENT FILTER 0-7, QUERY EVENT FILTER 8-15 and QUERY EVENT FILTER 16-23 respectively.

The relevant parts of the IEC 62386-3xx series shall define the meaning of the bits in "*eventFilter*" and can reduce the width of the variable *"eventFilter"* if needed. If the width is reduced to 2 bytes, DTR2 shall be ignored for SET EVENT FILTER (*DTR2:DTR1:DTR0*) and QUERY EVENT FILTER 16-23 shall answer NO. Similarly, if the width is reduced to 1 byte, DTR1 shall additionally be ignored for SET EVENT FILTER (*DTR2:DTR1:DTR0*) and QUERY EVENT FILTER 8-15 shall also answer NO.

9.8 Input signal, measured value and "*inputValue*"

9.8.1 General

An instance shall process its input signal into a measured value and expose this value to the system, as described in 9.8.2 to 9.8.4.

9.8.2 Input resolution

The processing shall be done with a precision which is indicated by *"resolution"*. The actual resolution used for a particular instance (type) can be subject to requirements in relevant parts of the IEC 62386-3xx series and/or manufacturer choice.

The measured value shall be contained in the N-byte variable *"inputValue"*, where N is the minimum number of bytes needed to contain at least *"resolution"* bits.

NOTE N is computed as (*"resolution"*/8) rounded up to the nearest integer. With *"resolution"* in the range [1,255], *"inputValue"* can span up to 32 bytes.

The measured value shall be most significant bit (msb)-aligned in *"inputValue"*. Unused bits in *"inputValue"* shall contain a repeating pattern of the most significant bit(s) of the measured value.

Table 9 provides an example, which shows a measured value of almost 50 % latched into a 1-byte *"inputValue"* after being processed with a *"resolution"* of 3, 4 and 5 bits respectively.

Table 9 – Measured value (≈ 50 %) versus resolution and "*inputValue*"

		Bits								
Resolution	**Measured value**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**"*inputValue*"**
3-bits	3 of [0, 7]	0	1	1	0	1	1	0	1	109
4-bits	7 of [0, 15]	0	1	1	1	0	1	1	1	119
5-bits	15 of [0, 31]	0	1	1	1	1	0	1	1	123
NOTE The grey shaded bits are (part of) the (first) repetition of the significant bits.										

This method allows an application controller to interpret the input value correctly as an 8-bit value, regardless of the actual instance resolution or sensor precision. The minimum value of all bytes in *"inputValue"* is always 0, the maximum value 0xFF, for all resolutions. The relative measured value corresponds (albeit with variable accuracy) to the relative input value.

9.8.3 Getting the input value

An instance shall support a latching mechanism that allows an application controller to obtain a consistent multi-byte input value. An example of such latching scenario is given in Table 10.

Application controllers can start reading a multi-byte value by sending the command "QUERY INPUT VALUE". This command shall trigger a latch that contains a copy of *"inputValue"* such that the remaining bytes can be read using a sequence of "QUERY INPUT VALUE LATCH" queries. After having returned the last byte of the latch, the instance shall not answer further "QUERY INPUT VALUE LATCH" queries until after the next "QUERY INPUT VALUE".

Table 10 – Example of querying sequence to read a 4-byte input value

Input signal	*"inputValue"*	Command	Answer	Latched *"inputValue"*
"12340000"	0x12340000	…	…	unspecified
"12345678"	0x12345678"	"QUERY INPUT VALUE"	0x12	0x12345678
"852"	0x00000852	"QUERY INPUT VALUE LATCH"	0x34	0x12345678
"124852"	0x00124852	"QUERY INPUT VALUE LATCH"	0x56	0x12345678
"124852"	0x00124852	"QUERY INPUT VALUE LATCH"	0x78	0x12345678
"124852"	0x00124852	"QUERY INPUT VALUE LATCH"	NO	0x12345678

The *"inputValue"* that is latched is the *"inputValue"* at the moment "QUERY INPUT VALUE" is executed.

NOTE 1 This implies that if an application controller queries the *"inputValue"* because of an event message it has just received, the value obtained is not necessarily the same value that triggered the event.

The latched value shall be updated only when the next "QUERY INPUT VALUE" is executed. If the application controller uses "QUERY INPUT VALUE LATCH" without having used "QUERY INPUT VALUE" as the command before this one, the answer can contain old or invalid data.

NOTE 2 To prevent concurrent access to the latched data, an application controller can transmit the necessary queries for this scenario within a transaction, exiting the scenario at any point.

NOTE 3 If an application controller can work sufficiently accurately with 16-bit input for the given instance type, it can stop after having received the most significant 16 bits of input value, and handle those bits as if they were delivered by an instance with *"resolution"* equal to 16. This allows straightforward resolution-independent algorithm implementation.

9.8.4 Notification of changes

A change or a sequence of changes in the input signal of an instance shall result in an event message as required by this document or that part of the IEC 62386-3xx series that describes the *"instanceType"* (see 9.5.3) of that instance.

The event message shall be sent using "INPUT NOTIFICATION (*device/instance, event*)", as described in 11.3.1.

NOTE The manufacturer of the input device can ensure that no event is lost by providing a queue. Parts of the IEC 62386-3xx series can impose additional restrictions, e.g. to avoid event flooding.

9.9 System failure

An application controller should detect system failure and recovery. Preferably, it should act upon any bus power failure with a duration longer than 40 ms, thus anticipating a power cycle of bus powered devices.

NOTE Bus powered devices can shut down due to a power outage of more than 40 ms.

Next, when the system failure is resolved, the application controller should ensure that the system resumes normal operation.

9.10 Operating a control device

9.10.1 Enable/disable the application controller

If present, the application controller is either active or not-active, as shall be reflected by *"applicationActive"*. While deactivated, the application controller shall not send any forward frames, except possibly a power cycle notification (see 9.13.2).

"applicationActive" shall have no influence on the response to incoming forward transmissions, including the transmission of backward frames following queries.

NOTE This allows the application controller to monitor the bus, but the application controller cannot use forward frames to react.

"applicationActive" shall be stored in the NVM of the application controller. The default value shall be TRUE in case there is an application controller present, which can be changed by another application controller using the commands ENABLE APPLICATION CONTROLLER and DISABLE APPLICATION CONTROLLER. "*applicationActive*" can be queried using "QUERY APPLICATION CONTROLLER ENABLED".

9.10.2 Application controller always active

If an application controller is present it can be always active. This shall be reflected by "*applicationControllerAlwaysActive*" being TRUE.

When "*applicationControllerAlwaysActive*" is TRUE, "*applicationControllerPresent*" and "*applicationActive*" shall always be TRUE.

"*applicationControllerAlwaysActive*" can be observed through "QUERY DEVICE CAPABILITIES" and "QUERY APPLICATION CONTROLLER ALWAYS ACTIVE".

9.10.3 Enable/disable event messages

Event messages are either enabled or disabled, as shall be reflected by *"instanceActive"*. While deactivated, the instance shall not send any forward frames. That is, the instance will not produce any event messages.

"instanceActive" shall have no influence on the response to incoming forward transmissions, including the transmission of backward frames following queries.

"instanceActive" shall be stored in the persistent memory of the input device. The default value shall be TRUE, which can be changed by an application controller using the commands "ENABLE INSTANCE" and "DISABLE INSTANCE". "*instanceActive*" can be queried using "QUERY INSTANCE ENABLED".

To limit the event messages when enabled, filtering is also available, see 9.7.4.

NOTE Queries are the only way to get information from an instance when event messages are disabled.

9.10.4 Quiescent mode

In quiescent mode, the control device shall not produce any forward frames except as a possible result of execution of SEND TESTFRAME. No commands (see also 9.10.1), and no event messages (see also 9.10.3) shall be transmitted, regardless of *"applicationActive"* or any *"instanceActive"*. Event messages shall not be queued and shall not be set as pending. This means that such events are discarded.

Quiescent mode is a temporary mode which is started or restarted with the command "START QUIESCENT MODE". It shall end automatically 15 min ± 1,5 min after the last "START QUIESCENT MODE" command was executed. Additionally, the command "STOP QUIESCENT MODE" shall terminate quiescent mode immediately.

In quiescent mode, a control device shall still respond to commands. "QUERY QUIESCENT MODE" can be used to determine whether or not a control device is in quiescent mode.

At power on of the control device, quiescent mode shall be DISABLED.

NOTE 1 Quiescent mode can be used by the application controller during initialisation (see 9.15) to ensure that random address comparisons are not frustrated by forward frames from other devices on the bus.

NOTE 2 Quiescent mode works independently from *"applicationActive"* and *"instanceActive"*. This implies that ending quiescent mode does not necessarily enable forward frame transmissions.

9.10.5 Modes of operation

9.10.5.1 General

Different operating modes can be selected at device level by means of command "SET OPERATING MODE (*DTR0*)". The currently selected *"operatingMode"* can be queried by means of "QUERY OPERATING MODE".

Operating modes 0x00 to 0x7F are defined in this document. At least operating mode 0x00 shall be available. Operating modes 0x80 to 0xFF are manufacturer specific. The query "QUERY MANUFACTURER SPECIFIC MODE" can be used to determine whether the control device is in an IEC 62386 standard operating mode, or in a manufacturer-specific mode.

9.10.5.2 Operating mode 0x00: standard mode

If a device is in *"operatingMode"* 0x00, its behaviour shall be as is required by this document, until it is set in an operating mode different from 0x00.

9.10.5.3 Operating mode 0x01 to 0x7F: reserved

Operating modes 0x01 to 0x7F are reserved and shall not be used.

9.10.5.4 Operating mode 0x80 to 0xFF: manufacturer-specific modes

Manufacturer-specific modes should only be used if the features required by the application are not covered by the IEC 62386 series. If a control device is in a manufacturer-specific operating mode, the behaviour of the control device may be manufacturer specific as well, with the following exceptions:

- with regard to bus access, the control device shall adhere to IEC 62386-101:2022.
- the control device shall adhere to this document at least as far as the following commands are concerned:
  - "SET OPERATING MODE (*DTR0)*", and "QUERY OPERATING MODE" and "QUERY MANUFACTURER SPECIFIC MODE".
  - All special commands (see 11.10) except WRITE MEMORY LOCATION (*DTR1, DTR0, data*), WRITE MEMORY LOCATION – NO REPLY (*DTR1, DTR0, data*) and DIRECT WRITE MEMORY (*DTR1, offset, data*).

For the above commands, the requirements of 7.2.1 shall apply.

It is recommended that even in manufacturer-specific modes, the commands as specified in all implemented parts of the IEC 62386 series still be obeyed.

9.11 Memory banks

9.11.1 General

Memory banks are freely accessible memory spaces containing information or configuration settings of the control device. Not all consecutive memory banks need to be implemented. Also within a memory bank not all consecutive locations need to be implemented. All implemented memory bank locations of all implemented memory banks are readable using memory access commands. Part of the memory is read-only and either programmed by the manufacturer of the control device or updated by the control device itself. For all other parts, write access using memory access commands can be enabled by the manufacturer. Except for location 0x02, all writable locations of a memory bank are lockable unless specified in the corresponding memory bank table. Locations within the memory banks shall be implemented using the memory types shown in Table 11.

Table 11 – Memory types

Memory type	Accessibility via the bus[a]	Volatility[b]	May be changed autonomously by the control device during run time	Description
ROM	RO	NV	No	For all fixed values that will not change during run time of the control device. NOTE ROM is read-only by its nature, but can be changed during production programming.
RAM-RO	RO	V	Yes	For values that will not be retained through a power-cycle. Cannot be changed over the interface.
RAM-RW	RW	V	Yes	For values that will not be retained through a power-cycle. Can be changed over the interface
NVM-RO	RO	NV	Yes	For values that will be retained through a power-cycle. Cannot be changed over the interface.
NVM-RW	RW	NV	Yes	For values that will be retained through a power-cycle. Can be changed over the interface.

[a] RO: Read-only. RW: Read-write.

[b] V: volatile (not retained through a power-cycle). NV: non-volatile (retained through a power-cycle).

The addressable memory space is limited to a maximum of 256 memory banks of maximum 255 bytes each (approximately 64 kBytes). As this document specifies how to implement memory bank 0 and 1 (if present), and reserves memory banks 200 to 255, this leaves room for 198 memory banks for manufacturer-specific purposes in the range of [2,199].

9.11.2 Memory map

If a manufacturer-specific memory bank in the range of [2,199] is implemented, allocation of its content shall comply with the memory map provided in Table 12.

Table 12 – Basic memory map of memory banks

Address	Description	Default value (factory)	Reset value[b]	Memory type
0x00	Address of last accessible memory location	factory burn-in, range [0x03,0xFE]	no change	ROM
0x01	Indicator byte[a]	[a]	[a]	any[a]
0x02	Memory bank lock byte. Lockable bytes in the memory bank shall be read-only while the lock byte has a value different from 0x55.	0xFF	0xFF[c]	RAM-RW
[0x03,0xFE]	Memory bank content[a]	[a]	[a]	any[a]
0xFF	Reserved – not implemented	answer NO	no change	n.a.

[a] The purpose, default values, power on values, reset values, and memory types of these bytes shall be defined by the manufacturer.

[b] Reset value after "RESET MEMORY BANK".

[c] Also used as power on value unless explicitly stated otherwise.

The byte in location 0x00 of each bank contains the address of the last accessible memory location of the bank. The value shall be in the range [0x03,0xFE].

The byte in location 0x01 is manufacturer specific. If implemented, the usage of this byte should be described by the manufacturer (as well as the entire content of the memory bank).

NOTE 1 It could be used for example to store a checksum in case of a memory bank with static content. Using a checksum on a memory bank where the content is changed by the control device is not useful.

The byte in location 0x02 shall be used to lock write access. Memory location 0x02 itself shall never be locked for writing. While this memory location contains any value different from 0x55, all memory locations marked "(lockable)" of the corresponding memory bank shall be read only. The control device shall not change the value of the lock byte other than as a consequence of a power cycle or of a "RESET MEMORY BANK (*DTR0*)" command or other command affecting the lock byte.

Location 0xFF is a reserved location in every memory bank, and is not accessible. This location shall not be implemented as a normal memory bank location. When addressed, the control device shall respond as if this location is not implemented, and it shall not increment "*DTR0*".

NOTE 2 This location is reserved in order to stop the auto increment of "*DTR0*".

9.11.3 Selecting a memory bank location

In order to select a memory bank location a combination of memory bank number and location inside the memory bank is required.

The memory bank shall be selected by setting the memory bank number in *"DTR1"*. The location in the memory bank shall be selected by the value in *"DTR0"*.

9.11.4 Protectable memory locations

Memory bank locations marked as "protectable" can have read or write protection applied or removed using a manufacturer-specific method.

Protectable locations with read protection enabled shall reply MASK as a result of "READ MEMORY LOCATION (*DTR1, DTR0*)". In this case, the implementation or provision of the related feature is optional.

NOTE 1 Read protection can be enabled by default. If read protection is enabled, this means that the related feature is possibly not implemented and there is no obligation for the manufacturer to provide a mechanism to deactivate read protection.

Protectable locations with write protection enabled cannot be overwritten.

NOTE 2 This means no reply to the WRITE MEMORY LOCATION command when attempting to write to a write-protected location.

9.11.5 Memory bank reading

9.11.5.1 General

A selected memory bank location can be read using command "READ MEMORY LOCATION (*DTR1, DTR0*)". The answer shall be the value of the byte at the addressed memory bank location.

If the selected memory bank is not implemented, the command shall be discarded.

If the selected memory bank location is below location 0xFF, *"DTR0"* shall be incremented by one, even if the memory location is not implemented. Otherwise, *"DTR0"* shall not change. This mechanism allows for easy consecutive reading of memory bank locations.

After reading a number of bytes from a memory bank, the application controller should check the value of "*DTR0*" to verify it is at the expected or desired location. Any mismatch indicates an error while reading.

9.11.5.2 Reading multi-byte values

To ensure consistent data when reading a multi-byte value from a memory bank, a mechanism shall be implemented that latches all bytes of the multi-byte value when the first byte of the multi-byte value is read, and holds them latched until the first byte of any multi-byte or single byte value in any memory bank of the same logical unit is read.

Reading from a multi-byte location shall reply with the stored value, and not values from the write-buffer that is used to buffer values before the complete value is written to memory (see 9.11.6.3).

9.11.5.3 Unimplemented locations

If a memory bank exists, and the selected memory bank location is

- not implemented and MASK is not shown in the allowed range of values, or
- above the last accessible memory location,

the answer to "READ MEMORY LOCATION (*DTR1, DTR0*)" shall be NO.

If a memory bank exists, and the selected memory bank location is

- not implemented, and
- the allowed range for the value stored in the memory bank location includes MASK,

the answer to "READ MEMORY LOCATION (*DTR1, DTR0*)" shall be MASK.

NOTE The use of MASK in the range of allowed values shown in a memory bank table, effectively allows the implementation of the location to be optional.

9.11.5.4 Temporarily unavailable locations

For memory locations which include TMASK in the range of allowed values in a memory bank table, the answer to "READ MEMORY LOCATION (*DTR1, DTR0*)" shall be TMASK if the value is temporarily unavailable. Except in fault conditions, the control device shall provide a valid value within 30 s.

EXAMPLE After an external supply power cycle, memory bank values such as measurements can be temporarily unavailable due to the settling time of digital filters, and is indicated by the use of TMASK.

NOTE If TMASK is returned repeatedly over a period of time, application controllers can deduce that there is a fault in the control device.

9.11.5.5 Latching a complete memory bank for reading

Memory bank tables that show the lock byte value 0xAA latches the complete bank, shall operate as follows:

If the lock byte contains a value other than 0xAA, writing the following values to the lock byte shall cause the stated result:

- 0xAA: All locations in the memory bank shall be latched and shall not change until the lock byte is written, or a power cycle occurs.
- Other values: The memory bank latch shall not be affected.

If the lock byte contains the value 0xAA, writing the following values to the lock byte shall cause the stated result:

- 0xAA: All locations in the memory bank shall be re-latched (updated) and shall not change again until the lock byte is written, or a power cycle occurs.
- Other values: The memory bank latch shall be removed. Memory reads shall result in the latest values being returned.

An attempt to write to any location other than the lock byte of a latched memory bank shall result in the same behaviour as if the memory location is not implemented.

Latching of a full memory bank shall not affect reading or writing of other memory banks.

EXAMPLE The following example demonstrates two application controllers accessing a latched memory bank: If application controller A latches a full memory bank 202, then application controller B needs to be aware that memory bank 202 is latched. When application controller B reads from memory bank 202, application controller B will get the latched data. Application controller B can determine this by either monitoring the bus activity related to writing the lock byte, or by reading the lock byte before reading other locations.

9.11.6 Memory bank writing

9.11.6.1 General

Write commands are special commands and therefore not addressable. In order to select the correct control device(s) the addressable command "ENABLE WRITE MEMORY" shall be used. Upon execution of "ENABLE WRITE MEMORY", the addressed control device(s) shall set *"writeEnableState"* to ENABLED.

Only while *"writeEnableState"* is ENABLED, and the addressed memory bank is implemented, the control device shall execute the following commands to write to a selected memory bank location:

- "WRITE MEMORY LOCATION (*DTR1, DTR0, data*)": The control device shall confirm writing a memory location with an answer equal to the value *data*.

NOTE 1 The value that can be read from the memory bank location is not necessarily *data*.

- "WRITE MEMORY LOCATION – NO REPLY (*DTR1, DTR0, data*)": Writing a memory location shall not cause the control device to reply.
- "DIRECT WRITE MEMORY (*DTR1, offset, data*)": The address of the memory location inside the selected bank is given by the content of the instance byte. *offset* is copied to "*DTR0*", after which the command is treated as "WRITE MEMORY LOCATION (*DTR1, DTR0, data*)". The control device shall confirm writing a memory location by replying with an answer equal to *data*.

A control device shall set *"writeEnableState"* to DISABLED if any command other than one of the following commands is accepted:

- "WRITE MEMORY LOCATION (*DTR1, DTR0, data*)",
  "WRITE MEMORY LOCATION – NO REPLY (*DTR1, DTR0, data*)",
  "DIRECT WRITE MEMORY (*DTR1, offset, data*)";
- "DTR0 (*data*)", "DTR1 (*data*)", "DTR1:DTR0 (*data1, data0*)", DTR2 (*data*),
  DTR2:DTR1 (*data2, data1*);
- "QUERY CONTENT DTR0", "QUERY CONTENT DTR1", "QUERY CONTENT DTR2".

NOTE 2 This means that commands other than those shown above, which are accepted and then discarded, still set "*writeEnableState*" to DISABLED.

9.11.6.2 Write failures

If the selected memory bank location is

- not implemented, or
- above the last accessible memory location, or
- locked (see 9.11.2), or
- not writeable, or
- is a protectable location that is currently write protected (see 9.11.4), or
- there is an attempt to write a value outside of the permitted range,

the answer to "WRITE MEMORY LOCATION (*DTR1, DTR0, data*)" and "DIRECT WRITE MEMORY (*DTR1, offset, data*)" shall be NO and no memory location shall be written to. If this applies when writing to a multi-byte location, the reply of NO shall occur after attempting to write to the least significant byte (LSB).

If the selected memory bank location is below 0xFF, *"DTR0"* shall be incremented by one. Otherwise, *"DTR0"* shall not change. This mechanism allows for easy consecutive writing to memory bank locations.

9.11.6.3 Writing multi-byte values

To ensure consistent data when writing a multi-byte value into a memory bank, a RAM buffer shall be used such that the buffer stores the temporary bytes being written until the LSB of the multi-byte value is written, at which point the complete value is written to the memory bank locations.

NOTE 1 The contents of buffers used to store temporary bytes for multi-byte writing can be lost when a write operation is started to any other multi-byte value in the same memory bank, or during a power-cycle, or as a result of execution of RESET MEMORY BANK (*DTR0*). This means that it is sufficient to include only one such RAM buffer for writing, per memory bank.

After writing a number of bytes to a memory bank, the application controller should check the value of "*DTR0*" to verify it is at the expected or desired location. Any mismatch indicates an error while writing.

NOTE 2 "*DTR0*" is also incremented if a non-implemented memory bank location is addressed before 0xFF is reached.

9.11.7 Memory bank 0

Memory bank 0 contains information about the control device. Memory bank 0 shall be implemented in all multi-master control devices.

Memory bank 0 shall be implemented using the memory map shown in Table 13, with at least the memory locations up to address 0x7F implemented, excluding reserved locations.

Table 13 – Memory map of memory bank 0

Address	Description	Default value (factory)	Memory type
0x00	Address of last accessible memory location	factory burn-in	ROM
0x01	Reserved – not implemented	answer NO	n.a.
0x02	Number of last accessible memory bank	factory burn-in, range [0,0xFF]	ROM
0x03	GTIN byte 0 (MSB)[a]	factory burn-in	ROM
0x04	GTIN byte 1	factory burn-in	ROM
0x05	GTIN byte 2	factory burn-in	ROM
0x06	GTIN byte 3	factory burn-in	ROM
0x07	GTIN byte 4	factory burn-in	ROM
0x08	GTIN byte 5 (LSB)	factory burn-in	ROM
0x09	Firmware version (major)	factory burn-in	ROM
0x0A	Firmware version (minor)	factory burn-in	ROM
0x0B	Identification number byte 0 (MSB)	factory burn-in	ROM
0x0C	Identification number byte 1	factory burn-in	ROM
0x0D	Identification number byte 2	factory burn-in	ROM
0x0E	Identification number byte 3	factory burn-in	ROM
0x0F	Identification number byte 4	factory burn-in	ROM
0x10	Identification number byte 5	factory burn-in	ROM
0x11	Identification number byte 6	factory burn-in	ROM
0x12	Identification number byte 7 (LSB)	factory burn-in	ROM
0x13	Hardware version (major)	factory burn-in	ROM
0x14	Hardware version (minor)	factory burn-in	ROM
0x15	101 version number[b]	factory burn-in, according to implemented version number	ROM
0x16	102 version number of all integrated control gear[c]	factory burn-in, according to implemented version number	ROM
0x17	103 version number of all integrated control devices[d]	factory burn-in, according to implemented version number	ROM
0x18	Number of logical control device units in the bus unit	factory burn-in, range [1,64]	ROM
0x19	Number of logical control gear units in the bus unit	factory burn-in, range [0,64]	ROM
0x1A	Index number of this logical control device unit	factory burn-in, range [0,(location 0x18) − 1]	ROM
0x1B[f]	Current bus unit configuration[f]	factory burn-in[g]	ROM
[0x1C,0x7F]	Reserved – not implemented	answer NO	n.a.
[0x80,0xFE]	Additional control device information[e]	[e]	ROM
0xFF	Reserved – not implemented	answer NO	n.a.

Key	
GTIN	global trade item number
LSB	least significant byte
MSB	most significant byte
[a]	It is recommended that the product GTIN is not re-used within the expected lifetime of the product after installation.
[b]	Format of the version number is defined in IEC 62386-101:2022, 4.2.
[c]	Format of the version number is defined in IEC 62386-102:2022, 4.2. If not implemented, this is indicated by 0xFF.
[d]	Format of the version number is defined in 4.2.
[e]	Purpose and (default) value of these bytes shall be defined by the manufacturer.
[f]	See 9.20. If this location is not implemented, the answer shall be NO.
[g]	The current bus unit configuration can be changed by a manufacturer-specific method.

If there is more than one logical unit built into one bus unit, all logical units shall have the same values in memory bank locations 0x03 up to and including 0x19. All control device logical units shall have the same values in location 0x1B.

A bus unit can contain both control gear and control devices. They share various numbers (e.g. GTIN, identification number). To avoid problems when reading, and getting different answers depending on the addressing scheme used, the memory bank layout is the same for control gear and for control devices up to and including location 0x19. The data shall be the same as well. The application controller can use either the commands specified in IEC 62386-102:— or the commands specified in this document to identify the basic data, provided both are implemented.

The bytes in locations 0x03 to 0x08 ("GTIN 0" to "GTIN 5") shall contain the global trade item number (GTIN) in binary (see [1][1]). The bytes shall be stored most significant first and filled with leading zeroes.

The bytes in locations 0x09 and 0x0A ("firmware version") shall contain the firmware version of the bus unit.

The bytes in locations 0x0B to 0x12 ("identification number byte 0" to "identification number byte 7") shall contain 64 bits of an identification number of the bus unit. It is recommended that this identification number is the serial number. The identification number shall be stored with the least significant byte in "identification number byte 7" and unused bits shall be filled with 0.

The combination of the identification number and the GTIN number shall be unique.

The byte in location 0x13 and 0x14 ("hardware version") shall contain the hardware version of the bus unit.

The byte in location 0x15 shall contain the implemented IEC 62386-101 version number of the bus unit.

The byte in location 0x16 shall contain the implemented IEC 62386-102 version number of the bus unit. If no control gear is implemented, the version number shall be 0xFF.

[1] Numbers in square brackets refer to the Bibliography.

The byte in location 0x17 shall contain the implemented IEC 62386-103 version number of the bus unit.

The byte in location 0x18 shall contain the number of logical control device units integrated into the bus unit. The number of logical units shall be in the range of 1 to 64.

The byte in location 0x19 shall contain the number of logical control gear units integrated into the bus unit. The number of logical units shall be in the range of 0 to 64.

The byte in location 0x1A shall represent the unique index number of the logical control device unit that implements that memory bank. The valid range of this index number is 0 to the total number of logical control device units in the bus unit minus one.

EXAMPLE A product can contain three control device logical units with three different short addresses. Each of these logical units has the same GTIN and identification number, each reports the number of logical control device units with the value 3 and the index of the three control device logical units is reported as 0, 1 or 2 respectively. Reading location 0x1A using broadcast yields a backward frame according to IEC62386-101:2022, 9.6.2 (corrupted backward frame).

The byte in location 0x1B, if present, shall contain the current bus unit configuration, which shall indicate the currently selected implementation of application controllers and logical units. See 9.20.

9.11.8 Memory bank 1 (optional)

Memory bank 1 is reserved for use by an original equipment manufacturer (OEM), (e.g. a luminaire manufacturer) to store additional information, which has no impact on the functionality of the control device. Implementation of memory bank 1 is optional.

If implemented, memory bank 1 shall at least implement the memory locations up to and including address 0x10. The fixed usage for location 0x00 to 0x02 and the recommended memory map usage for locations 0x03 to 0x10 is shown in Table 14.

Table 14 – Memory map of memory bank 1

Address	Description	Default value (factory)	Reset value[b]	Memory type
0x00	Address of last accessible memory location	factory burn-in, range [0x10,0xFE]	no change	ROM
0x01	Indicator byte[a]	[a]	[a]	any[a]
0x02	Memory bank 1 lock byte. Lockable bytes in the memory bank shall be read-only while the lock byte has a value different from 0x55.	0xFF	0xFF[c]	RAM-RW
0x03	OEM GTIN byte 0 (MSB)	0xFF	no change	NVM-RW (lockable)
0x04	OEM GTIN byte 1	0xFF	no change	NVM-RW (lockable)
0x05	OEM GTIN byte 2	0xFF	no change	NVM-RW (lockable)
0x06	OEM GTIN byte 3	0xFF	no change	NVM-RW (lockable)
0x07	OEM GTIN byte 4	0xFF	no change	NVM-RW (lockable)
0x08	OEM GTIN byte 5 (LSB)	0xFF	no change	NVM-RW (lockable)
0x09	OEM identification number byte 0 (MSB)	0xFF	no change	NVM-RW (lockable)

Address	Description	Default value (factory)	Reset value[b]	Memory type
0x0A	OEM identification number byte 1	0xFF	no change	NVM-RW (lockable)
0x0B	OEM identification number byte 2	0xFF	no change	NVM-RW (lockable)
0x0C	OEM identification number byte 3	0xFF	no change	NVM-RW (lockable)
0x0D	OEM identification number byte 4	0xFF	no change	NVM-RW (lockable)
0x0E	OEM identification number byte 5	0xFF	no change	NVM-RW (lockable)
0x0F	OEM identification number byte 6	0xFF	no change	NVM-RW (lockable)
0x10	OEM identification number byte 7 (LSB)	0xFF	no change	NVM-RW (lockable)
≥ 0x11	Additional control device information[a]	[a]	[a]	[a]
0xFF	Reserved – not implemented	answer NO	no change	n.a.

Key

GTIN global trade item number

LSB least significant byte

MSB most significant byte

OEM original equipment manufacturer

[a] The purpose, default value, power on value, reset value and memory type of these bytes shall be defined by the manufacturer.

[b] Reset value after "RESET MEMORY BANK".

[c] Also used as power on value.

The bytes in locations 0x03 to 0x08 ("OEM GTIN 0" to "OEM GTIN 5") should be used to identify the product containing the control device. If the bytes are used for the GTIN, the bytes shall be stored most significant bit first and filled with leading zeroes. These bytes should be programmed by the OEM.

The bytes in locations 0x09 to 0x10 ("OEM identification number byte 0" to "OEM identification number byte 7") should contain 64 bits of an identification number of the OEM product. If the bytes are used for the identification number, it shall be stored with the least significant byte in "identification number byte 7" and unused bits shall be filled with 0. These bytes should be programmed by the OEM.

The combination of OEM GTIN and OEM identification number should be unique.

9.11.9 Manufacturer-specific memory banks

The manufacturer may use additional memory banks in the range of 2 to 199 to store additional information. The memory map of additional banks shall comply with Table 12.

9.11.10 Reserved memory banks

Memory banks 200 to 255 are reserved for future use, or are described in the IEC 62386-3xx series. Implementation or use other than described in the IEC 62386-3xx series is not permitted.

9.12 Reset

9.12.1 Reset operation

A control device shall implement a reset operation to set all device variables and instance variables (see Table 19 and Table 20) to their reset values.

NOTE For some variables this operation could have no effect at all.

The reset operation shall take at most 300 ms to complete. While the reset operation is in progress, the control device can respond to commands or not. However, until the reset operation is complete, none of the affected variables needs to have a defined value.

An application controller can trigger the reset operation using the "RESET" instruction and should wait at least 350 ms to ensure all control devices have finished the reset operation.

9.12.2 Reset memory bank operation

A control device shall implement a reset operation to set the content of all unlocked memory banks to their reset values (see 9.11), followed by locking the memory banks.

NOTE For some memory bank locations this operation could have no effect at all.

The reset memory bank operation shall take at most 10 s to complete. While this reset operation is in progress, the control device can respond to commands or not. However, until this reset memory bank operation is complete, none of the affected memory locations have a defined value.

An application controller can trigger the reset operation for a specific memory bank, or for all implemented memory banks, using the "RESET MEMORY BANK (*DTR0*)" instruction and it should then wait for at least 10,1 s to ensure all control devices have enough time to finish the reset memory bank operation.

9.13 Power on behaviour

9.13.1 Power on

After an external power cycle (see IEC 62386-101:2022, 4.11.1 and 4.11.5), the device shall maintain its most recent configuration of NVM variables saved according to 9.18, with the following exceptions:

- all variables mentioned in Table 19 and Table 20 shall be set to the value indicated in the power on value column. The variables that are marked with 'no change' in the power on value column shall not be considered. The variables defined in implemented parts of the IEC 62386-3xx series shall be included;
- the memory bank write enable state shall be disabled for all memory banks and the lock byte shall be set to 0xFF;
- quiescent mode shall be cancelled (see 9.10.4);
- all running timers shall be stopped and cancelled (reset);
- *"powerCycleSeen"* shall be set to TRUE.

"powerCycleSeen" can be observed through "QUERY DEVICE STATUS".

In order to observe a subsequent power cycle, the application controller should clear *"powerCycleSeen"*, using the command "RESET POWER CYCLE SEEN.

NOTE In a system with multiple application controllers that make use of power cycle information of other control devices in the system, application controllers can take consideration when clearing *"powerCycleSeen"*.

9.13.2 Power cycle notification

After completing its power on behaviour according to 9.13.1, a bus unit shall generate one power cycle event message per device if the *"powerCycleNotification"* is ENABLED for at least one of its logical units.

An application controller can use "ENABLE POWER CYCLE NOTIFICATION" and "DISABLE POWER CYCLE NOTIFICATION" to enable/disable power cycle events for specific control devices. *"powerCycleNotification"* shall be DISABLED by default.

NOTE 1 The power cycle notification is not inhibited by *"applicationActive"* nor by any *"instanceActive"*.

The event shall be generated using the "POWER NOTIFICATION (*device*)" message as described in 11.2. The event message shall be sent once using priority 2 and with a uniformly distributed delay between 1,3 s and 5,0 s after completion of the power-on procedure.

NOTE 2 Applying a random delay helps to avoid collisions of power cycle notifications.

9.14 Priority use

9.14.1 General

The purpose of forward frame priorities is to facilitate appropriate system behaviour within a multi-master system. Priorities ensure that transmissions for time critical system reaction will have precedence over transmissions for non-time critical system operation.

- The first forward frame in a transaction (see IEC 62386-101:2022, 9.3) shall be sent with priority 2 to 5. All other forward frames in a transaction shall be sent with priority 1.
- Forward frames that are not part of a transaction shall be sent with priority 2 to 5.
- Priority 2 should be used to execute user instigated actions for switching or dimming the lights. This implies appropriate event messages and level instructions. Priority 2 can also be used during commissioning (e.g. addressing).

EXAMPLE 1 Switching or dimming actions triggered via push-button or presence detector.

- Priority 3 should be used for configuration of a bus unit and for those event messages that are not covered by Priorities 2 and 4.

EXAMPLE 2 Writing to memory banks or feedback events.

- Priority 4 should be used to execute automatic actions for switching or dimming the lights. This means sending appropriate event messages and level instructions.

EXAMPLE 3 Switching or dimming actions triggered by a light sensor.

- Priority 5 should be used for periodic query commands.

9.14.2 Priority of input notifications

An instance shall use a default *"eventPriority"* equal to Priority 4 when transmitting an event message to produce an "INPUT NOTIFICATION (*device/instance, event*)". For particular instance types, this default priority is subject to change by the relevant parts of the IEC 62386-3xx series.

In a system, the default *"eventPriority"* can be overruled by the application controller using "SET EVENT PRIORITY (*DTR0*)". "QUERY EVENT PRIORITY" can be used to observe the currently active *"eventPriority"*.

9.15 Assigning short addresses

9.15.1 General

"shortAddress" shall be derived from *data* or *"DTR0"* depending on the command used. It shall be set on execution of "PROGRAM SHORT ADDRESS (*data*)" or "SET SHORT ADDRESS (*DTR0*)" as follows:

- if *data* or "*DTR0*" = MASK: MASK (effectively deleting the short address);
- if *data* or *"DTR0"* < 0x40: *data* or *"DTR0"*;
- in all other cases: no change.

9.15.2 Random address allocation

A control device shall implement an initialisation state, only in which, apart from the other operations identified in this document, a set of commands are enabled that allow an application controller to detect and uniquely identify control devices available on the bus and assign short addresses to these devices.

The initialisation state is a temporary state which is entered with the command "INITIALISE (*device*)". It shall end automatically 15 min ± 1,5 min after the last "INITIALISE (*device*)" command was executed. Additionally, a power cycle or the command "TERMINATE" shall cause the control device to leave the initialisation state immediately.

The control device shall have three possible values for *"initialisationState"*:

- DISABLED, not in initialisation state;
- ENABLED, in initialisation state;
- WITHDRAWN, in initialisation state, yet identified and withdrawn.

The following (special) commands are initialisation commands:

- "RANDOMISE", "COMPARE" and "WITHDRAW";
- "SEARCHADDRH (*data*)", "SEARCHADDRM (*data*)" and "SEARCHADDRL (*data*)";
- "PROGRAM SHORT ADDRESS (*data*)", "VERIFY SHORT ADDRESS (*data*)" and "QUERY SHORT ADDRESS";
- "IDENTIFY DEVICE".

NOTE "IDENTIFY DEVICE" is by itself not an initialisation command, but typically used during initialisation.

9.15.3 Identification of a device

No variables shall be affected by the identification procedure. Normal processing shall continue, i.e. events can be generated on a change of the input value; this shall not stop identification. Where appropriate, variables can be temporarily ignored, so that after the identification has ended, there are no side effects.

Identification shall be stopped upon execution of any instruction other than INITIALISE (*device*) or "IDENTIFY DEVICE".

Identification can be started by sending the instruction "IDENTIFY DEVICE". This shall start or restart a 10 s ± 1 s timer. While the timer is running, a procedure enabling an observer to identify the selected control device shall run. If the timer expires, identification shall stop.

NOTE The actual procedure is manufacturer specific.

When identification is stopped by an application controller, the corresponding timer shall be cancelled immediately.

9.16 Exception handling

Control devices and instances shall expose whether an error has occurred by setting (in case of error) and resetting (in case of no error) the following flags.

- An application controller shall change *"applicationControllerError"*. This status can be queried through "QUERY DEVICE STATUS" (see 9.17.2). Detailed error information can be obtained from "QUERY APPLICATION CONTROLLER ERROR" (see 11.6.4).
- A control device that is not an application controller shall have *"applicationControllerError"* set to FALSE.
- An input device shall change *"inputDeviceError"* to indicate an error in any of the input device instances of the logical unit. This status can be queried through "QUERY DEVICE STATUS" (see 9.17.2). Detailed error information can be obtained from "QUERY INPUT DEVICE ERROR" (see 11.6.5).
- A control device that is not an input device shall have *"inputDeviceError"* set to FALSE.
- An instance shall change *"instanceError"*. This status can be queried through "QUERY INSTANCE STATUS" (see 9.17.3). Detailed error information can be obtained from "QUERY INSTANCE ERROR" (see 11.9.4).

9.17 Device capabilities and status information

9.17.1 Device capabilities

Each control device shall expose its features as a combination of device capabilities as given in Table 15:

Table 15 – Control device capabilities

Bit	Description	Value	See Subclause
0	*"applicationControllerPresent"* is TRUE?	"1" = "Yes"	9.3.3
1	*"numberOfInstances"* is greater than 0?	"1" = "Yes"	9.5.2
2	*"applicationControllerAlwaysActive"* is TRUE?	"1" = "Yes"	9.10.2
3 to 4	Reserved for IEC 62386-104 (see [2])	"0" = default value	
5	At least one instance supports configuration of *"instanceType"* or *"instanceConfiguration[]"*	"1" = "Yes"	9.19
6 to 7	unused	"0" = default value	

The device capabilities can be queried using "QUERY DEVICE CAPABILITIES".

9.17.2 Device status

Each control device shall expose its status as a combination of device properties as given in Table 16:

Table 16 – Control device status

Bit	Description	Value	See Subclause
0	*"inputDeviceError"* is TRUE?	"1" = "Yes"	9.16
1	*"quiescentMode"* is ENABLED?	"1" = "Yes"	9.10.4
2	*"shortAddress"* is MASK?	"1" = "Yes"	9.15.1
3	*"applicationActive"* is TRUE?	"1" = "Yes"	9.10.1
4	*"applicationControllerError"* is TRUE?	"1" = "Yes"	9.16
5	*"powerCycleSeen"* is TRUE?	"1" = "Yes"	9.13
6	*"resetState"* is TRUE?	"1" = "Yes"	See below
7	Reserved for IEC 62386-104 (see [2])	"0" = default value	

The device status can be queried using "QUERY DEVICE STATUS".

Bit 6: *"resetState"* shall be set to TRUE if all the NVM variables mentioned in Table 19 and Table 20 are at their reset value. The NVM variables that are marked with 'no change' in the reset value column shall not be considered. NVM variables defined in implemented parts of the IEC 62386-3xx series shall be included. In all other cases the bit shall be set to FALSE.

9.17.3 Instance status

Each instance shall expose its status as a combination of instance properties as given in Table 17:

Table 17 – Instance status

Bit	Description	Value	See subclause
0	*"instanceError"* is TRUE?	"1" = "Yes"	9.16
1	*"instanceActive"* is TRUE?	"1" = "Yes"	9.10.3
2 to 7	unused	"0" = default value	

The instance status can be observed using "QUERY INSTANCE STATUS".

9.18 Non-volatile memory

After any change to an NVM variable, the new value shall be restored after a power cycle, provided a period of at least 30 s followed the NVM variable change, before the power cycle.

In case the above condition is not met, it is possible that the changed NVM variable will not be restored with the new value.

NOTE The method used to ensure NVM variables are saved is manufacturer specific. One possible method is to save some NVM variables within 30 s of them being changed, with others being saved at the time the external power failure occurs. The method chosen is likely to influence the lifetime of the non-volatile memory.

9.19 Instance types and configuration

Instances may optionally support a change of *"instanceType"* with the instruction "SET INSTANCE TYPE (*DTR0*)".

The current instance type can be determined by using "QUERY INSTANCE TYPE", and all available instance types can be discovered by use of "QUERY AVAILABLE INSTANCE TYPES".

Instances may optionally support a change of *"instanceConfiguration[]"* with the instruction "SET INSTANCE CONFIGURATION (*DTR0, DTR2:DTR1*)".

The current instance configuration can be discovered by using "QUERY INSTANCE CONFIGURATION (*DTR0*)".

Any standardised values of *"instanceConfiguration[]"* are published in the appropriate parts of the IEC 62386-3xx series. A manufacturer-specific range of *"instanceConfiguration[]"* may be used, but use of such values shall not remove any requirements in this document.

If instance configuration is supported, the manual shall state which locations within *"instanceConfiguration[]"* are supported, shall describe the use of such locations in the case that these are in the manufacturer-specific range, and shall state their memory type (readable and/or writeable).

EXAMPLE 1 Changing the instance type: An instance with volt-free inputs can support a change of instance type between type 1 (push-button) and type 2 (absolute/switch-input device).

EXAMPLE 2 Changing the instance configuration: An occupancy sensor instance with multiple sensor technologies such as PIR, microwave/doppler and ultra-sonic can allow configuration of which sensor technologies are currently active.

9.20 Current bus unit configuration

If present, the value of current bus unit configuration from memory bank 0 shall indicate the currently active implementation of:

- the value of "*applicationControllerPresent*".

For the given value of current bus unit configuration, the implementation shall be as shown in Table 18.

Table 18 – Current bus unit configuration

Current bus unit configuration	"*applicationControllerPresent*"	Description
0 to 191[a]	Reserved	
192 to 255	Manufacturer-specific	
[a] Values reserved for future updates to this document.		

A manufacturer-specific method is permitted for selection of the current configuration. After changing, the bus unit can require a certain time, power-cycle or other requirement before the new configuration becomes active.

After changing the configuration, the value of "Current bus unit configuration" in memory bank 0 shall indicate the active configuration, and the bus unit operation shall be updated according to the new configuration.

10 Declaration of variables

Table 19 shows the default values, the reset values, the range of validity and the type of memory of the defined instance independent variables.

Table 20 shows the default values, the reset values, the range of validity and the type of memory of the defined variables of each of the instances.

The variables that are declared in this Clause 10 shall not be made available for writing through a memory bank.

Table 19 – Declaration of device variables

Variable	Default value (factory)	Reset value	Power on value	Range of validity	Memory type
"shortAddress"	MASK (no address)	no change	no change	[0,63], MASK	NVM
"deviceGroups"	0x0000 0000	0x0000 0000	no change	[0,0xFFFF FFFF]	NVM
"searchAddress"	[a]	0xFF FF FF	0xFF FF FF	[0,0xFF FF FF]	RAM
"randomAddress"	0xFF FF FF	0xFF FF FF	no change	[0,0xFF FF FF]	NVM
"DTR0"	[a]	no change	0x00	[0,0xFF]	RAM
"DTR1"	[a]	no change	0x00	[0,0xFF]	RAM
"DTR2"	[a]	no change	0x00	[0,0xFF]	RAM
"numberOfInstances"	factory burn-in	no change	no change	[0,32]	ROM
"operatingMode"	factory burn-in	no change	no change	0, [0x80,0xFF]	NVM
"quiescentMode"	[a]	DISABLED	DISABLED	[ENABLED, DISABLED]	RAM
"applicationActive"	*applicationControllerPresent*	no change	no change	[TRUE, FALSE]	NVM
"writeEnableState"	[a]	DISABLED	DISABLED	[ENABLED, DISABLED]	RAM
"applicationControllerPresent"	factory burn-in	no change	no change	[TRUE, FALSE]	ROM
"applicationControllerAlwaysActive"	factory burn-in	no change	no change	[TRUE, FALSE]	ROM
"powerCycleSeen"	[a]	FALSE	TRUE	[TRUE, FALSE]	RAM
"powerCycleNotification"	DISABLED	no change	no change	[ENABLED, DISABLED]	NVM
"initialisationState"	[a]	no change	DISABLED	[ENABLED, DISABLED, WITHDRAWN]	RAM
"applicationControllerError"	[a]	[c]	FALSE[b]	[TRUE, FALSE]	RAM
"inputDeviceError"	[a]	[c]	FALSE[b]	[TRUE, FALSE]	RAM
"resetState"	TRUE	TRUE	TRUE[b]	[TRUE, FALSE]	RAM
"eventPriority"[d]	4	no change	no change	[2,5]	NVM
"versionNumber"	3.0	no change	no change	00001100b	ROM

[a] Not applicable.

[b] The value should reflect the actual situation as soon as possible.

[c] The value could change as a consequence of the RESET command execution.

[d] This variable is independent of the instance variable(s) "*eventPriority*".

Table 20 – Declaration of instance variables

Variable	Default value (factory)	Reset value	Power on value	Range of validity	Memory type
"instanceGroup0"	MASK	MASK	no change	[0,31], MASK	NVM
"instanceGroup1"	MASK	MASK	no change	[0,31], MASK	NVM
"instanceGroup2"	MASK	MASK	no change	[0,31], MASK	NVM
"instanceActive"	TRUE	no change	no change	[TRUE, FALSE]	NVM
"instanceType"	factory burn-in	no change	no change	[0,31]	ROM or NVM
"resolution"	factory burn-in	no change	no change	[1,255]	ROM
"inputValue"	[a]	no change	no change[b]	$[0,2^{N*8}-1]$[c]	RAM
"instanceNumber"	factory burn-in	no change	no change	[0, *"numberOfInstances"* – 1]	ROM
"eventFilter"[d]	0xFF FF FF	0xFF FF FF	no change	[0, 0xFF FF FF]	NVM
"eventScheme"	0	0	no change	[0,4]	NVM
"eventPriority"[d] [f]	4	no change	no change	[2,5]	NVM
"instanceError"	[a]	[e]	FALSE[b]	[TRUE, FALSE]	RAM
"instanceConfiguration[]"[g]	factory burn-in	no change	no change	[0, 0xFFFF][g]	NVM or ROM

[a] Not applicable.

[b] The value should reflect the actual situation as soon as possible.

[c] *N* computed as (*"resolution"*/8) rounded up to the nearest integer.

[d] For particular instance types, the values belonging to this variable can be changed by the relevant part of the IEC 62386-3xx series.

[e] The value could change as a consequence of the RESET command execution.

[f] Instance variable(s) "*eventPriority*" are independent of the device variable "*eventPriority*".

[g] The size of this array is manufacturer-specific, from 0 to 256 locations. Implemented locations are up to 16 bits in size.

11 Definition of commands

11.1 General

Unused opcodes are reserved for future needs.

11.2 Overview sheets

Table 21 to Table 24 give an overview of the control device's event messages, standard commands and special commands.

Table 21 – Instance event messages

Event message name	Event source information	Event information	References	Command subclause
INPUT NOTIFICATION (*device/instance, event*)	*device/instance*	*event*	9.7.3 and 9.8.4	11.3.1

Table 22 – Device event messages

Event message name	Bits [23,13]	Bits [12,0]	References	Command subclause
POWER NOTIFICATION (*device*)	0x7F7	*device*	9.7.2 and 9.13.2	11.3.2

Table 23 – Standard commands

Command name	Address byte	Instance byte			Opcode byte	App Ctrl	Input device	DTR0	DTR1	DTR2	Answer	Send twice	References	Command subclause
		Device	Instance	Feature										
IDENTIFY DEVICE	*Device*	✓			0x00	✓	✓					✓	9.15.3	11.4.2
RESET POWER CYCLE SEEN	*Device*	✓			0x01	✓	✓					✓	9.13.1	11.4.3
RESET	*Device*	✓			0x10	✓	✓					✓	9.12.1	11.5.2
RESET MEMORY BANK (*DTR0*)	*Device*	✓			0x11	✓	✓	✓				✓	9.12.2	11.5.3
SET SHORT ADDRESS (*DTR0*)	*Device*	✓			0x14	✓	✓	✓				✓	9.15.1	11.5.4
ENABLE WRITE MEMORY	*Device*	✓			0x15	✓	✓					✓	9.11.6	11.5.5
ENABLE APPLICATION CONTROLLER	*Device*	✓			0x16	✓						✓	9.10.1	11.5.6
DISABLE APPLICATION CONTROLLER	*Device*	✓			0x17	✓						✓	9.10.1	11.5.7
SET OPERATING MODE (*DTR0*)	*Device*	✓			0x18	✓	✓	✓				✓	9.10.5	11.5.8
ADD TO DEVICE GROUPS 0-15 (*DTR2:DTR1*)	*Device*	✓			0x19	✓	✓		✓	✓		✓		11.5.9
ADD TO DEVICE GROUPS 16-31 (*DTR2:DTR1*)	*Device*	✓			0x1A	✓	✓		✓	✓		✓		11.5.10
REMOVE FROM DEVICE GROUPS 0-15 (*DTR2:DTR1*)	*Device*	✓			0x1B	✓	✓		✓	✓		✓		11.5.11
REMOVE FROM DEVICE GROUPS 16-31 (*DTR2:DTR1*)	*Device*	✓			0x1C	✓	✓		✓	✓		✓		11.5.12
START QUIESCENT MODE	*Device*	✓			0x1D	✓	✓					✓	9.10.4	11.5.13
STOP QUIESCENT MODE	*Device*	✓			0x1E	✓	✓					✓	9.10.4	11.5.14
ENABLE POWER CYCLE NOTIFICATION	*Device*	✓			0x1F	✓	✓					✓	9.13.2	11.5.15
DISABLE POWER CYCLE NOTIFICATION	*Device*	✓			0x20	✓	✓					✓	9.13.2	11.5.16
Reserved[a]	*Device*	✓			0x21[a]	✓	✓					✓		
Reserved for IEC 62386-104 (see [2])	*Device*	✓			0x22	✓		✓						

Command name	Address byte	Instance byte			Opcode byte	App Ctrl	Input device	DTR0	DTR1	DTR2	Answer	Send twice	References	Command subclause
		Device	Instance	Feature										
Reserved for IEC 62386-104 (see [2])	*Device*	✓			0x23	✓								
Reserved for IEC 62386-104 (see [2])	*Device*	✓			0x24	✓								
QUERY DEVICE STATUS	*Device*	✓			0x30	✓	✓				✓		9.17.2	11.6.3
QUERY APPLICATION CONTROLLER ERROR	*Device*	✓			0x31	✓					✓		9.16	11.6.4
QUERY INPUT DEVICE ERROR	*Device*	✓			0x32		✓				✓		9.16	11.6.5
QUERY MISSING SHORT ADDRESS	*Device*	✓			0x33	✓	✓				✓			11.6.6
QUERY VERSION NUMBER	*Device*	✓			0x34	✓	✓				✓		4.2	11.6.7
QUERY NUMBER OF INSTANCES	*Device*	✓			0x35	✓	✓				✓		9.5	11.6.9
QUERY CONTENT DTR0	*Device*	✓			0x36	✓	✓	✓			✓			11.6.8
QUERY CONTENT DTR1	*Device*	✓			0x37	✓	✓		✓		✓			11.6.10
QUERY CONTENT DTR2	*Device*	✓			0x38	✓	✓			✓	✓			11.6.11
QUERY RANDOM ADDRESS (H)	*Device*	✓			0x39	✓	✓				✓			11.6.12
QUERY RANDOM ADDRESS (M)	*Device*	✓			0x3A	✓	✓				✓			11.6.13
QUERY RANDOM ADDRESS (L)	*Device*	✓			0x3B	✓	✓				✓			11.6.14
READ MEMORY LOCATION (*DTR1, DTR0*)	*Device*	✓			0x3C	✓	✓	✓	✓		✓		9.11.5	11.6.15
QUERY APPLICATION CONTROLLER ENABLED	*Device*	✓			0x3D	✓					✓		9.10.1	11.6.16
QUERY OPERATING MODE	*Device*	✓			0x3E	✓	✓				✓		9.10.5	11.6.17
QUERY MANUFACTURER SPECIFIC MODE	*Device*	✓			0x3F	✓	✓				✓		9.10.5	11.6.18
QUERY QUIESCENT MODE	*Device*	✓			0x40	✓	✓				✓		9.10.4	11.6.19
QUERY DEVICE GROUPS 0-7	*Device*	✓			0x41	✓	✓				✓			11.6.20
QUERY DEVICE GROUPS 8-15	*Device*	✓			0x42	✓	✓				✓			11.6.21

Command name	Address byte	Instance byte			Opcode byte	App Ctrl	Input device	DTR0	DTR1	DTR2	Answer	Send twice	References	Command subclause
		Device	Instance	Feature										
QUERY DEVICE GROUPS 16-23	*Device*	✓			0x43	✓	✓				✓			11.6.22
QUERY DEVICE GROUPS 24-31	*Device*	✓			0x44	✓	✓				✓			11.6.23
QUERY POWER CYCLE NOTIFICATION	*Device*	✓			0x45	✓	✓				✓		9.13.2	11.6.24
QUERY DEVICE CAPABILITIES	*Device*	✓			0x46	✓	✓				✓		9.17.1	11.6.2
QUERY EXTENDED VERSION NUMBER(*DTR0*)	*Device*	✓			0x47	✓	✓	✓			✓			11.6.25
QUERY RESET STATE	*Device*	✓			0x48	✓	✓				✓		9.17.2	11.6.26
QUERY APPLICATION CONTROLLER ALWAYS ACTIVE	*Device*	✓			0x49	✓					✓		9.10.2	11.6.27
SET EVENT PRIORITY (*DTR0*)	*Device*	✓	✓		0x61		✓	✓				✓	9.14.2	11.8.8, 11.5.17
ENABLE INSTANCE	*Device*		✓		0x62		✓					✓	9.10.3	11.8.2
DISABLE INSTANCE	*Device*		✓		0x63		✓					✓	9.10.3	11.8.3
SET PRIMARY INSTANCE GROUP (*DTR0*)	*Device*		✓		0x64		✓	✓				✓	9.5.5	11.8.4
SET INSTANCE GROUP 1 (*DTR0*)	*Device*		✓		0x65		✓	✓				✓	9.5.5	11.8.5
SET INSTANCE GROUP 2 (*DTR0*)	*Device*		✓		0x66		✓	✓				✓	9.5.5	11.8.6
SET EVENT SCHEME (*DTR0*)	*Device*		✓		0x67		✓	✓				✓	9.7.3	11.8.7
SET EVENT FILTER (*DTR2, DTR1, DTR0*)	*Device*		✓		0x68		✓	✓	✓	✓		✓	9.7.4	11.8.9
SET INSTANCE TYPE (*DTR0*)	*Device*		✓		0x69		✓	✓				✓	9.19	11.8.10
SET INSTANCE CONFIGURATION (*DTR0, DTR2:DTR1*)	*Device*		✓		0x6A		✓	✓	✓	✓		✓	9.19	11.8.11
QUERY INSTANCE TYPE	*Device*		✓		0x80		✓				✓		9.5.3	11.9.2
QUERY RESOLUTION	*Device*		✓		0x81		✓				✓		9.8.2	11.9.3
QUERY INSTANCE ERROR	*Device*		✓		0x82		✓				✓		9.16	11.9.4

Command name	Address byte	Instance byte			Opcode byte	App Ctrl	Input device	DTR0	DTR1	DTR2	Answer	Send twice	References	Command subclause
		Device	Instance	Feature										
QUERY INSTANCE STATUS	*Device*		✓		0x83		✓				✓		9.17.3	11.9.5
QUERY EVENT PRIORITY	*Device*	✓	✓		0x84		✓				✓		9.14.2	11.9.13, 11.6.30
QUERY INSTANCE ENABLED	*Device*		✓		0x86		✓				✓		9.10.3	11.9.6
QUERY PRIMARY INSTANCE GROUP	*Device*		✓		0x88		✓				✓		9.5.5	11.9.7
QUERY INSTANCE GROUP 1	*Device*		✓		0x89		✓				✓		9.5.5	11.9.8
QUERY INSTANCE GROUP 2	*Device*		✓		0x8A		✓				✓		9.5.5	11.9.9
QUERY EVENT SCHEME	*Device*		✓		0x8B		✓				✓		9.7.3	11.9.10
QUERY INPUT VALUE	*Device*		✓		0x8C		✓				✓		9.8.3	11.9.11
QUERY INPUT VALUE LATCH	*Device*		✓		0x8D		✓				✓		9.8.3	11.9.12
QUERY FEATURE TYPE	*Device*	✓	✓		0x8E	✓	✓				✓		9.2, 9.5.4	11.9.14, 11.6.28
QUERY NEXT FEATURE TYPE	*Device*	✓	✓		0x8F	✓	✓				✓		9.2, 9.5.4	11.9.15, 11.6.29
QUERY EVENT FILTER 0-7	*Device*		✓		0x90		✓				✓		9.7.4	11.9.16
QUERY EVENT FILTER 8-15	*Device*		✓		0x91		✓				✓		9.7.4	11.9.17
QUERY EVENT FILTER 16-23	*Device*		✓		0x92		✓				✓		9.7.4	11.9.18
QUERY INSTANCE CONFIGURATION (*DTR0*)	*Device*		✓		0x93		✓	✓	✓	✓	✓		9.19	11.9.19
QUERY AVAILABLE INSTANCE TYPES	*Device*		✓		0x94		✓	✓	✓	✓	✓		9.19	11.9.20

[a] Reserved to maintain backward compatibility due to use in Edition 1 of IEC 62386-103:2014 (see [3]).

Table 24 – Special commands (implemented by both application controller and input device)

Command name	Address byte	Instance byte	Opcode byte	DTR0	DTR1	DTR2	Answer	Send twice	References	Command subclause
TERMINATE	0xC1	0x00	0x00							11.10.2
INITIALISE (*device*)	0xC1	0x01	*device*					✓	9.15	11.10.3
RANDOMISE	0xC1	0x02	0x00					✓	9.15	11.10.4
COMPARE	0xC1	0x03	0x00				✓		9.15	11.10.5
WITHDRAW	0xC1	0x04	0x00						9.15	11.10.6
SEARCHADDRH (*data*)	0xC1	0x05	*data*						9.15	11.10.7
SEARCHADDRM (*data*)	0xC1	0x06	*data*						9.15	11.10.8
SEARCHADDRL (*data*)	0xC1	0x07	*data*						9.15	11.10.9
PROGRAM SHORT ADDRESS (*data*)	0xC1	0x08	*data*						9.15	11.10.10
VERIFY SHORT ADDRESS (*data*)	0xC1	0x09	*data*				✓		9.15	11.10.11
QUERY SHORT ADDRESS	0xC1	0x0A	0x00				✓		9.15	11.10.12
Reserved for IEC 62386-104 (see [2])	0xC1	0x0B	*data*	✓			✓			
Reserved for IEC 62386-104 (see [2])	0xC1	0x0C	*data*							
Reserved for IEC 62386-104 (see [2])	0xC1	0x0D	*data*							
WRITE MEMORY LOCATION (*DTR1, DTR0, data*)	0xC1	0x20	*data*	✓	✓		✓		9.11.6	11.10.13
WRITE MEMORY LOCATION – NO REPLY (*DTR1, DTR0, data*)	0xC1	0x21	*data*	✓	✓				9.11.6	11.10.14
DTR0 (*data*)	0xC1	0x30	*data*	✓						11.10.15
DTR1 (*data*)	0xC1	0x31	*data*		✓					11.10.16
DTR2 (*data*)	0xC1	0x32	*data*			✓				11.10.17
SEND TESTFRAME (*data*)	0xC1	*0x33*	*data*	✓	✓	✓				11.10.21
DIRECT WRITE MEMORY (*DTR1, offset, data*)	0xC5	*offset*	*data*	✓	✓		✓		9.11.6	11.10.18
DTR1:DTR0 (*data1, data0*)	0xC7	*data1*	*data0*	✓	✓					11.10.19
DTR2:DTR1 (*data2, data1*)	0xC9	*data2*	*data1*		✓	✓				11.10.20

11.3 Event messages

11.3.1 INPUT NOTIFICATION (*device/instance, event*)

The event message notifies of a change or a series of changes of *"inputValue"* at the instance of an input device as required by this document or by the relevant part of the IEC 62386-3xx series corresponding to the *"instanceType"* of the instance.

The transmitting instance shall

- generate the event message only while *"instanceActive"* is TRUE;
- generate the event message only while it is not in an error condition that prevents operation (see 9.16);
- use the currently active *"eventScheme"*;
- use the requested *"eventPriority"*.

Refer to 9.7 and 9.8 for further information.

11.3.2 POWER NOTIFICATION (*device*)

The event notifies of a control device power cycle completion and shall be generated following the requirements as stated in 9.7.2 and 9.13.2.

11.4 Device control instructions

11.4.1 General

Device control instructions are used to modify property values of a control device. For this reason a device control instruction shall be discarded, unless it is accepted twice according to the requirements specified in IEC 62386-101:2022, 9.4.

Unless explicitly stated otherwise in the description of the particular device control instruction, the following holds:

- the instruction shall be ignored if so required by the provisions of 9.6;
- the control device shall not reply to the instruction;
- the instruction shall apply to device variables.

11.4.2 IDENTIFY DEVICE

The control device shall start or restart a 10 s ± 1 s identification procedure which shall enable an observer to distinguish any control device(s) running this process from any devices (of the same type) which are not running it. On expiry of this timer, the identification procedure shall stop.

The identification shall be stopped immediately upon execution of any instruction other than "INITIALISE (*device*)" or "IDENTIFY DEVICE".

NOTE 1 Identification can be used during commissioning, allowing an installer to locate devices and allocate the particular identified device to a particular device group.

The indication can be done in various ways, such as by flashing an LED, by producing a sound, other visual or audible means, or alternative methods such as a wireless transmission to a smart device or tool. The exact process used to identify is manufacturer specific and should be described in the manual. In choosing the method, consideration should be given for the availability of the required tools for the intended lifetime of the product.

Support for IDENTIFY DEVICE is optional provided all of the following conditions are met:

- there is no controllable emitter that could be used for identification purposes (such as an LED, buzzer or wireless transmitter), and
- the device can issue an INPUT NOTIFICATION in response to a user generated trigger, such as a push-button or light sensor, and
- a statement is included in the product documents, "This control device does not support identification by means of an LED, buzzer or other emitter."

If any of these conditions are not met, support for IDENTIFY DEVICE shall be implemented.

For the case when IDENTIFY DEVICE is not implemented, it is recommended that the GTIN and serial (identification) number, as used in memory bank 0, are shown on the product label in decimal format. If serial numbers are formatted in hexadecimal, then it is recommended to prefix with "0x".

NOTE 2 Application controllers can support event messages from input devices, as a method of identifying a device during commissioning.

NOTE 3 The application controller can also stop the identification process using a "RESET" command.

Refer to 9.15.3 for further information.

11.4.3 RESET POWER CYCLE SEEN

"powerCycleSeen" shall be set to FALSE.

Refer to 9.13.1 for further information.

11.5 Device configuration instructions

11.5.1 General

Device configuration instructions are used to change the configuration and/or the mode of operation of the control device. For this reason a device configuration instruction shall be discarded, unless it is accepted twice according to the requirements specified in IEC 62386-101:2022, 9.4.

Unless explicitly stated otherwise in the description of the particular device configuration instruction, the following holds:

- the instruction shall be ignored if so required by the provisions of 9.6;
- the control device shall not reply to the instruction;
- the instruction shall apply to device variables.

11.5.2 RESET

All variables shall be changed to their reset values. Control devices shall start to react properly to commands no later than 300 ms after the execution of the instruction has started.

If during a reset mains power fails, it is not guaranteed that "RESET" is completed.

Refer to 9.12.1, Table 19 and Table 20 for further information.

11.5.3 RESET MEMORY BANK (*DTR0*)

The command shall trigger the process to change the memory bank content to its reset values as follows:

- if *"DTR0"* = 0: all implemented and unlocked memory banks except memory bank 0 shall be reset;
- in all other cases: the memory bank identified by *"DTR0"* shall be reset, provided it is implemented and unlocked.

A memory bank needs to be unlocked to allow both lockable and non-lockable locations to be reset.

The control device shall start to react properly to commands no later than 10 s after the execution of the instruction has started.

Refer to 9.12.2 for further information.

11.5.4 SET SHORT ADDRESS (*DTR0*)

The *"shortAddress"* shall be set to *"DTR0"*.

The command shall be discarded if *"DTR0"* does not contain a valid *"shortAddress"* value.

Refer to 9.15.1 for further information.

11.5.5 ENABLE WRITE MEMORY

"writeEnableState" shall be set to ENABLED.

NOTE There is no command to explicitly disable memory write access, since any command that is not directly involved with writing into memory banks will reset *"writeEnableState"* back to DISABLED.

Refer to 9.11.6 for further information.

11.5.6 ENABLE APPLICATION CONTROLLER

If "*applicationControllerPresent*" is TRUE, *"applicationActive"* shall be set to TRUE, otherwise this command shall be discarded.

Refer to 9.10.1 for further information.

11.5.7 DISABLE APPLICATION CONTROLLER

If "*applicationControllerAlwaysActive*" is TRUE, this command shall be discarded.

If "*applicationControllerPresent*" is TRUE, *"applicationActive"* shall be set to FALSE, otherwise this command shall be discarded.

Refer to 9.10.1 and 9.10.2 for further information.

11.5.8 SET OPERATING MODE (*DTR0*)

"operatingMode" shall be set to *"DTR0"*.

If *"DTR0"* does not correspond to an implemented operating mode, the command shall be discarded.

Refer to 9.10.5 for further information.

11.5.9 ADD TO DEVICE GROUPS 0-15 (*DTR2:DTR1*)

The control device shall set those bits in *"deviceGroups[15:0]"* that are set in [*"DTR2:DTR1"*]. The other bits shall not change.

11.5.10 ADD TO DEVICE GROUPS 16-31 (*DTR2:DTR1*)

The control device shall set those bits in *"deviceGroups[31:16]"* that are set in [*"DTR2:DTR1"*]. The other bits shall not change.

11.5.11 REMOVE FROM DEVICE GROUPS 0-15 (*DTR2:DTR1*)

The control device shall clear those bits in *"deviceGroups[15:0]"* that are set in [*"DTR2:DTR1"*]. The other bits shall not change.

11.5.12 REMOVE FROM DEVICE GROUPS 16-31 (*DTR2:DTR1*)

The control device shall clear those bits in *"deviceGroups[31:16]"* that are set in [*"DTR2:DTR1"*]. The other bits shall not change.

11.5.13 START QUIESCENT MODE

The control device shall start or restart quiescent mode by setting *"quiescentMode"* to ENABLED and (re-)triggering the timer.

Refer to 9.10.4 for further information.

11.5.14 STOP QUIESCENT MODE

"quiescentMode" shall be set to DISABLED.

Refer to 9.10.4 for further information.

11.5.15 ENABLE POWER CYCLE NOTIFICATION

"powerCycleNotification" shall be set to ENABLED.

Refer to 9.13.2 for further information.

11.5.16 DISABLE POWER CYCLE NOTIFICATION

"powerCycleNotification" shall be set to DISABLED.

Refer to 9.13.2 for further information.

11.5.17 SET EVENT PRIORITY (*DTR0*)

The command shall be discarded if *"DTR0"* is not in the range [2,5].

"eventPriority" shall be set to *"DTR0"*.

11.6 Device queries

11.6.1 General

Device queries are used to retrieve device property values from a control device. The addressed control device returns the queried property value in a backward frame.

Unless explicitly stated otherwise in the description of the particular device query, the following holds:

- the query shall be ignored if so required by the provisions of 9.6;
- the query shall apply to device variables.

When applicable, the query shall be discarded if any of the parameter values (in *"DTR0"*, *"DTR1"* and *"DTR2"*) are outside the range of validity of the addressed device variables, as given in Table 19.

11.6.2 QUERY DEVICE CAPABILITIES

The answer shall be a combination of control device capabilities.

Refer to 9.17.1 for further information.

11.6.3 QUERY DEVICE STATUS

The answer shall be the status, which is formed by a combination of control device properties.

Refer to 9.17.2 for further information.

11.6.4 QUERY APPLICATION CONTROLLER ERROR

The answer shall be the detailed error information regarding an application controller:

- if an error in the application controller has occurred (as indicated by *"applicationControllerError"*), but the device is not able to give detailed error information: MASK;
- if an error in the application controller has occurred (as indicated by *"applicationControllerError"*), and the device is able to give detailed error information: error number [0,254];
- if no application controller error has occurred: NO.

Detailed error information is manufacturer specific and should be described in product documentation.

Refer to 9.16 for further information.

11.6.5 QUERY INPUT DEVICE ERROR

The answer shall be the detailed error information regarding an input device:

- if an error in the input device has occurred (as indicated by *"inputDeviceError"*), but the device is not able to give detailed error information: MASK;
- if an error in the input device has occurred (as indicated by *"inputDeviceError"*), and the device is able to give detailed error information: error number [0,254];
- if no input device error has occurred: NO.

Detailed error information is manufacturer specific and should be described in product documentation.

Refer to 9.16 for further information.

11.6.6 QUERY MISSING SHORT ADDRESS

The answer shall be YES if *"shortAddress"* is equal to MASK and NO otherwise.

NOTE Since the control device answers only if no short address is stored, the use of the command is useful only in broadcast mode or when device group addressing is used.

11.6.7 QUERY VERSION NUMBER

The answer shall be "*versionNumber*".

See 4.2 and Table 19 for more information.

11.6.8 QUERY CONTENT DTR0

The answer shall be *"DTR0"*.

11.6.9 QUERY NUMBER OF INSTANCES

The answer shall be *"numberOfInstances"*.

Refer to 9.5 for further information.

11.6.10 QUERY CONTENT DTR1

The answer shall be *"DTR1"*.

11.6.11 QUERY CONTENT DTR2

The answer shall be *"DTR2"*.

11.6.12 QUERY RANDOM ADDRESS (H)

The answer shall be *"randomAddress[23:16]"*.

11.6.13 QUERY RANDOM ADDRESS (M)

The answer shall be *"randomAddress[15:8]"*.

11.6.14 QUERY RANDOM ADDRESS (L)

The answer shall be *"randomAddress[7:0]"*.

11.6.15 READ MEMORY LOCATION (*DTR1, DTR0*)

The query shall be discarded if the memory bank identified by *"DTR1"* is not implemented.

If executed, the answer shall be the content of the memory location identified by offset *"DTR0"* within memory bank *"DTR1"*.

The control device shall answer NO if the addressed memory location is not implemented.

NOTE 1 This allows gaps in the memory bank implementation.

If the addressed offset is below location 0xFF in the bank, the control device shall increment *"DTR0"* by one.

NOTE 2 This allows efficient multi-byte reading within a transaction.

Refer to 9.11.5 for further information.

11.6.16 QUERY APPLICATION CONTROLLER ENABLED

The answer shall be YES if *"applicationActive"* is TRUE, NO otherwise.

Refer to 9.10.1 for further information.

11.6.17 QUERY OPERATING MODE

The answer shall be *"operatingMode"*.

Refer to 9.10.5 for further information.

11.6.18 QUERY MANUFACTURER SPECIFIC MODE

The answer shall be YES when *"operatingMode"* is in the range [0x80,0xFF] and NO otherwise.

Refer to 9.10.5 for further information.

11.6.19 QUERY QUIESCENT MODE

The answer shall be YES if *"quiescentMode"* is ENABLED, and NO otherwise.

Refer to 9.10.4 for further information.

11.6.20 QUERY DEVICE GROUPS 0-7

The answer shall be *"deviceGroups[7:0]"*.

11.6.21 QUERY DEVICE GROUPS 8-15

The answer shall be *"deviceGroups[15:8]"*.

11.6.22 QUERY DEVICE GROUPS 16-23

The answer shall be *"deviceGroups[23:16]"*.

11.6.23 QUERY DEVICE GROUPS 24-31

The answer shall be *"deviceGroups[31:24]"*.

11.6.24 QUERY POWER CYCLE NOTIFICATION

The answer shall be YES if *"powerCycleNotification"* is ENABLED, and NO otherwise.

Refer to 9.13.2 for further information.

11.6.25 QUERY EXTENDED VERSION NUMBER(*DTR0*)

The answer shall be the version number of the relevant part of the IEC 62386-3xx series where xx is given by *DTR0*.

The answer shall be:

- if the Part 3xx given by *DTR0* is not implemented: NO;
- if the Part 3xx given by *DTR0* is implemented: the version number of the Part 3xx.

Refer to the relevant part of the IEC 62386-3xx series for further information.

11.6.26 QUERY RESET STATE

The answer shall be YES if *"resetState"* is TRUE, and NO otherwise.

Refer to 9.17.2 for further information.

11.6.27 QUERY APPLICATION CONTROLLER ALWAYS ACTIVE

The answer shall be YES if "*applicationControllerAlwaysActive*" is TRUE, and NO otherwise.

Refer to 9.10.2 for further information.

11.6.28 QUERY FEATURE TYPE

See 11.9.14 for command execution and 9.2 for information on device features.

11.6.29 QUERY NEXT FEATURE TYPE

See 11.9.15 for command execution and 9.2 for information on device features.

11.6.30 QUERY EVENT PRIORITY

See 11.9.13 for command execution.

11.7 Instance control instructions

Instance control instructions are used to modify property values of an instance of an input device.

Unless explicitly stated otherwise in the description of the particular instance control instruction, the following holds:

- the instruction shall be ignored if so required by the provisions of 9.6;
- the input device shall not reply to the instruction;
- the instruction shall apply to instance variables.

NOTE This document does not describe any instance control instructions. However, the above requirements do apply to instance instructions described in the relevant parts of the IEC 62386-3xx series.

11.8 Instance configuration instructions

11.8.1 General

Instance configuration commands are used to change the configuration and/or the mode of operation of an instance within the input device. For this reason an instance configuration instruction shall be discarded, unless it is accepted twice according to the requirements specified in IEC 62386-101:2022, 9.4.

Unless explicitly stated otherwise in the description of the particular instance configuration instruction, the following holds:

- the instruction shall be ignored if so required by the provisions of 9.6;
- the input device shall not reply to the instruction;
- the instruction shall apply to instance variables.

11.8.2 ENABLE INSTANCE

"instanceActive" shall be set to TRUE.

Refer to 9.10.3 for further information.

11.8.3 DISABLE INSTANCE

"instanceActive" shall be set to FALSE.

Refer to 9.10.3 for further information.

11.8.4 SET PRIMARY INSTANCE GROUP (*DTR0*)

The instance shall have the primary membership to an instance group assigned or removed, by setting *"instanceGroup0"* to *"DTR0"*.

The command shall be discarded if *"DTR0"* is not in the range [0,31] and different from MASK.

Refer to 9.5.5 for further information.

11.8.5 SET INSTANCE GROUP 1 (*DTR0*)

The instance shall have an additional membership to an instance group assigned or removed, by setting *"instanceGroup1"* to *"DTR0"*.

The command shall be discarded if *"DTR0"* is not in the range [0,31] and different from MASK.

Refer to 9.5.5 for further information.

11.8.6 SET INSTANCE GROUP 2 (*DTR0*)

The instance shall have an additional membership to an instance group assigned or removed, by setting *"instanceGroup2"* to *"DTR0"*.

The command shall be discarded if *"DTR0"* is not in the range [0,31] and different from MASK.

Refer to 9.5.5 for further information.

11.8.7 SET EVENT SCHEME (*DTR0*)

The instance shall, if the provisions identified in 9.7.3 allow so, apply a new event addressing scheme for subsequent "INPUT NOTIFICATION (*device/instance, event*)" events by setting *"eventScheme"* to *"DTR0"*.

NOTE Circumstances can dictate that the operation cannot be granted by the receiving instance, meaning that the answer to the corresponding "QUERY EVENT SCHEME" can be different from the event scheme requested here.

The command shall be discarded if *"DTR0"* is not in the range [0,4].

Refer to 9.7.3 for further information.

11.8.8 SET EVENT PRIORITY (*DTR0*)

The instance shall apply a new message priority for subsequent "INPUT NOTIFICATION (*device/instance, event*)" events by setting *"eventPriority"* to *"DTR0"*.

The command shall be discarded if *"DTR0"* is not in the range [2,5].

Refer to 9.14 for further information.

11.8.9 SET EVENT FILTER (*DTR2:DTR1:DTR0*)

The instance shall set *"eventFilter[23:0]"* to [*"DTR2:DTR1:DTR0"*].

Refer to 9.7.4 for further information.

11.8.10 SET INSTANCE TYPE (*DTR0*)

The instance shall discard this command if any of the following conditions are true:

- configuration of the instance type is not supported by the instance, or
- *"DTR0[7:5]"* is non-zero, or
- *"DTR0[4:0]"* is not a supported instance type.

If executed, *"instanceType"* shall be set to *"DTR0[4:0]"*.

NOTE Application controllers can query the instance configuration after execution of this command, to check which instance configuration was activated.

A change of *"instanceType"* shall cause the instance variables of the addressed instance to be set to their reset values, and can cause the device to re-start with all other variables being set to the power-on values. In the case the device re-starts, the system start-up timing shall be met (see IEC 62386-101:2022, 4.11.6).

11.8.11 SET INSTANCE CONFIGURATION (*DTR0, DTR2:DTR1*)

The instance shall discard this command if any of the following conditions are true:

- configuration of *"instanceConfiguration[DTR0]"* is not supported by the instance, or
- *"DTR0"* < 191, and points to a value of *"instanceConfiguration[]"* that is not defined in the corresponding part of the IEC 62386-3xx series, or
- *"DTR0"* is 191, and *"DTR2:DTR1"* is not 0x55CC.

NOTE 1 *"DTR0"* values in the range [192, 255] allow manufacturer-specific instance configuration.

If executed and *"DTR0"* is not 191, then *"DTR2:DTR1"* shall be written to *"instanceConfiguration[DTR0]"*, with any unused bits of the 16-bit value discarded.

If *"DTR0"* is 191, and *"DTR2:DTR1"* is 0x55CC, then all implemented locations of *"instanceConfiguration[]"* shall be set to their factory default values.

A change of *"instanceConfiguration[DTR0]"* can cause the device to re-start with all variables being set to the power-on values. In the case the device re-starts, all further frames may be discarded until the re-start has completed. The system start-up timing shall be met (see IEC 62386-101:2022, 4.11.6).

NOTE 2 Application controllers can send query commands to determine if the device is ready to receive further frames, so allowing for the possibility of the device re-starting.

11.9 Instance queries

11.9.1 General

Instance queries are used to retrieve instance property values from an instance of an input device. The addressed input device returns the property value queried in a backward frame.

Unless explicitly stated otherwise in the description of the particular instance query, the following holds:

- the query shall be ignored if so required by the provisions in 9.6;
- if the query addresses multiple instances within the input device, the query shall be answered as if each instance is a logical device (see IEC 62386-101:2022, 9.6.2);
- the query shall apply to instance variables.

11.9.2 QUERY INSTANCE TYPE

The answer shall be *"instanceType"*.

Refer to 9.5.3 for further information.

11.9.3 QUERY RESOLUTION

The answer shall be *"resolution"*.

Refer to 9.8.2 for further information.

11.9.4 QUERY INSTANCE ERROR

The answer shall be detailed error information:

- if an error has occurred (as indicated by *"instanceError"*), but the instance is not able to give detailed error information: 0;
- if an error has occurred (as indicated by *"instanceError"*), and the instance is able to give detailed error information: error number [1,255];
- if no error has occurred: NO.

NOTE Detailed error information is instance type specific and is described in the IEC 62386-3xx series.

Refer to 9.16 for further information.

11.9.5 QUERY INSTANCE STATUS

The command queries the status of a combination of instance properties.

Refer to 9.17.3 for further information.

11.9.6 QUERY INSTANCE ENABLED

The answer shall be YES, if *"instanceActive"* is TRUE in at least one of the addressed instances, and NO otherwise.

Refer to 9.10.3 for further information.

11.9.7 QUERY PRIMARY INSTANCE GROUP

The answer shall be *"instanceGroup0"*.

Refer to 9.5.5 for further information.

11.9.8 QUERY INSTANCE GROUP 1

The answer shall be *"instanceGroup1"*.

Refer to 9.5.5 for further information.

11.9.9 QUERY INSTANCE GROUP 2

The answer shall be *"instanceGroup2"*.

Refer to 9.5.5 for further information.

11.9.10 QUERY EVENT SCHEME

The answer shall be *"eventScheme"*.

Refer to 9.7.3 for further information.

11.9.11 QUERY INPUT VALUE

The instance shall latch a new *"inputValue"* and reply with the most significant byte of the latched input value.

If the query addresses multiple instances within the input device, it shall be discarded.

Refer to 9.8.3 for further information.

11.9.12 QUERY INPUT VALUE LATCH

The instance shall reply with the next byte from a latched *"inputValue"*.

Following the least significant byte of the latched input value, the answer shall be NO, until a new "QUERY INPUT VALUE" has been executed.

If the query addresses multiple instances within the input device, it shall be discarded.

Refer to 9.8.3 for further information.

11.9.13 QUERY EVENT PRIORITY

The answer shall be *"eventPriority"*.

Refer to 9.14 for further information.

11.9.14 QUERY FEATURE TYPE

The answer shall be:

- if no feature 3xx is implemented: 254;
- if one device/instance feature is supported: the device/instance feature number;
- if more than one device/instance feature is supported: MASK.

Refer to 9.5.4 for further information.